Automatisches Deployment und adaptive Platzierung ubiquitärer Anwendungen

Dissertation

zur

Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

vorgelegt von
Enrico Seib
aus Rostock

Rostock, 6. Dezember 2016

urn:nbn:de:gbv:28-diss2017-0066-9

Gutachter: Prof. Dr.-Ing. Gero Mühl
Universität Rostock

Prof. Dr. rer. nat. Clemens H. Cap
Universität Rostock

Prof. Dr. rer. nat. Odej Kao
Technische Universität Berlin

Tag der Einreichung: 6. Dezember 2016
Tag der Verteidigung: 4. Mai 2017

Zusammenfasssung

Vernetzte Anwendungen und Geräte formen als intelligente Umgebung ein Ensemble, welches dem Nutzer vielfältige Informationen und Dienste anbietet, jedoch auch komplex und unübersichtlich ist. Weiterhin können Anwendungen und Geräte dem Ensemble beitreten oder aus diesem austreten, sodass dieses einer ständigen, dynamischen Veränderung unterliegt. Die miteinander verbundenen und interagierenden Geräte sind zudem nicht homogen bezüglich ihrer Ausstattung, ihres Betriebssystems und der verfügbaren Ressourcen. Sie bieten damit unterschiedliche Möglichkeiten für die Ausführung von Anwendungen. Um die Kommunikation und Kooperation unterschiedlicher Geräte und Anwendungen zu ermöglichen, kommt eine Middleware zum Einsatz, welche einheitliche Kommunikationsschnittstellen für die Anwendungen anbietet und die internen, lokalen Arbeitsweisen in Richtung der Anwendung verbirgt.

Publish/Subscribe bietet als ereignisbasiertes Kommunikationsparadigma die Möglichkeit, Geräte und Anwendungen entkoppelt miteinander kommunizieren und kooperieren zu lassen. Die Entkopplung der Geräte und Anwendungen erfolgt auf drei Ebenen: in Raum, Zeit und im Kontrollfluss. Vor allem die aus der Entkopplung durch ereignisbasierte Kommunikation erwachsene Möglichkeit, Geräte und Anwendungen jederzeit in das Ensemble einbinden zu können und ausscheiden zu lassen, prädestiniert Publish/Subscribe für dynamische Ensembles. Die Umsetzung als Middleware eröffnet zudem die Möglichkeit, Kommunikationsfunktionalität als einheitliche Schnittstelle anzubieten, womit die Middleware von der zugrunde liegenden Hardware abstrahiert. Dem Anwendungsentwickler bietet sich damit eine gemeinsame, einheitliche Ausführungsplattform für Anwendungen. Die mittels einer Publish/Subscribe-basierten Middleware aufgebauten Netzwerke können jedoch schnell unübersichtlich werden, zudem bleiben die unterschiedlichen Verarbeitungsressourcen bei Geräten und Kommunikationsverbindungen nach wie vor bestehen.

Dem Nutzer ist es aufgrund der Vielzahl von Anwendungen und Geräten und Kommunikationsverbindungen nicht möglich, diese zu überblicken und gemäß der Anwendungsanforderungen eine ausreichend gute Anwendungsplatzierung zu finden. Werden durch eine ungeschickte Anwendungsplatzierung einzelne Geräte oder Kommunikationsverbindungen überlastet, kann dies zu Einbußen bei der Qualität der bereitgestellten Dienste führen. Die Fähigkeit von Publish/Sub-

scribe, eine Vielzahl heterogener Geräte und Anwendungen in einem Netzwerk zu einem Servicenetz zusammenzufügen wird an dieser Stelle zu einem Problem, wenn wie bisher üblich, die Nutzer die gewünschten Anwendungen selbst deployen, aber die Anwendungsanforderungen und die im Netzwerk verfügbaren Ressourcen nicht überblicken. Selbst wenn der Nutzer trotz der unvollständigen Informationen eine sinnvolle Platzierung erreicht hat, ist diese durch die dynamische Veränderung des Geräteensembles und der Anwendungen jedoch einer begrenzten Lebensdauer unterworfen. Die Anwendungsplatzierung muss daher automatisch und laufend adaptiv ausgeführt werden.

Der in dieser Arbeit vorgestellte Ansatz löst das Problem der initialen und laufenden Platzierung unter Berücksichtigung von Anwendungsanforderungen und verfügbaren Ausführungs- und Kommunikationsressourcen selbst in dynamischen Umgebungen. Im ersten Aufgabenfeld, der initialen Anwendungsplatzierung, wird zunächst eine *gültige Anwendungsplatzierung gefunden* und die Anwendung ausgeführt. Dazu bedient sich der Ansatz der in Publish/Subscribe genutzten Primitive der Ankündigung, Subskription und Notifikation. Es wird ein Verteilungsprotokoll vorgestellt, welches neben den Anforderungen und verfügbaren Ressourcen die eindeutige Identifikation von Geräten und Anwendungen ermöglicht und damit die Entkopplung der Anwendungen und Geräte durch Publish/Subscribe zum Teil rückgängig macht.

Im zweiten Schritt erfolgt die *laufende und adaptive Anpassung der Anwendungsplatzierung* an Änderungen des Verhaltens von Geräten und Anwendungen. Aufgrund der Komplexität des Platzierungsproblems, selbst bei statischen Problemen liegt sie im NP-schweren Bereich, wird auf eine *Heuristik mit lokalem Wissen* zurückgegriffen. Die Platzierungsanpassung erfolgt auf der Grundlage eines Kostenmodells und eines definierten Platzierungsziels. Die identifizierten tatsächlichen Kosten und die alternativen Ausführungskosten werden in ein Kräftemodell überführt, wobei durch die Differenz beider Kosten Kräfte modelliert werden, die die Anwendungsplatzierung beeinflussen. Je nach Art der einwirkenden Kräfte wird eine oder eine Folge von Basisoperationen auf der Anwendung ausgeführt und die Anwendungsplatzierung gezielt verändert.

Die Implementierung des Ansatzes der automatischen initialen und laufenden Platzierung erfolgt als funktionale Erweiterung der Middleware REBECA, die als einheitliche Ausführungsumgebung mit den entsprechenden *Plugins* die Platzierungsanpassung steuert. Mit der gezeigten Umsetzung des Platzierungsansatzes wird gezeigt, dass er einsetzbar ist und das Potential von Kosteneinsparungen bietet. Zum Nachweis, dass der Verteilungsalgorithmus signifikant Kosten einspart, wurden unterschiedliche Anwendungsplatzierungsszenarien entwickelt. Mit Hilfe von Simulation und der um die Plugins der Platzierungsanpassung erweiterten Middleware wird anschließend nachgewiesen, dass bei den meisten Ressourcenverteilungen signifikant Kosten eingespart werden.

Abstract

As smart environments, networked applications and devices constitute ensembles which provide manifold information and services to the user while becoming more and more complex at the same time. Beside the behavior, joining or leaving of devices and applications add to the already existing complexity and lead to continuous changes of the ensemble. These interacting devices are heterogeneous in their features, resources and operating systems. Due to these differences, they offer different opportunities for executing applications and fulfill their requirements. To enable different, homogeneous devices and applications to communicate and cooperate with each other, a middleware is used that provides obvious and uniform communication interfaces for the application and hides internal local processes at the same time.

Publish/subscribe as an event-driven communication paradigm provides the possibility of decoupled communication of devices and applications, in space and time as well as with regard to control flow, by using events. The decoupling due to event-driven communication allows easy integration and removal of devices and applications to and from the ensemble, respectively. Therefore, publish/subscribe is ideally suited for the communication in dynamic ensembles. Moreover, the implementation of publish/subscribe into a middleware provides a uniform run-time environment for applications to the application developers. However, a network based on publish/subscribe middleware can become large, complex and confusing. Furthermore, the unequal distribution of resources regarding devices and communication connections still remains.

From the user's perspective, it is nearly impossible to keep track of the possibilities and limitations of the multitude of devices and communication connections, and to find an adequate application placement according to the applications demands. The overload of individual devices and communication connections may cause serious problems and result in the loss of provided services. The publish/subscribe middleware's ability to integrate a large number of heterogeneous devices and applications into a single network becomes a problem, as the user is responsible for the deployment of the desired applications without knowledge about available resources and application demands. Even if the user was able to find an appropriate solution, its endurance would be restricted due to the dynamic changes of the system. Thus, the initial as well as ongoing applica-

tion placement has to be carried out automatically. The approach presented in this work solves the problem of initial and ongoing placement of applications and application components taking into account the applications requirements as well as the available resources, even in dynamic environments. First of all, the initial placement is realized. For this purpose, a *valid placement for application execution* is found first, which means that application requirements are fulfilled considering the limited resources. To realize this, the approach uses the primitives of publish/subscribe including announcements, subscriptions and notifications. Furthermore, a distribution protocol is presented which offers the distinct identification of requirements, available resources and devices and works opposite to the loose coupling of publish/subscribe.

In a second step, the *ongoing adaptation* of application placement is performed. By surpassing the meet of application demands and available resources, the focus lies on the reduction of the running costs triggered by the application's execution and communication in a dynamic environment. Due to the complexity of the application placement problem as NP-hard, a *heuristic*, that uses *local knowledge* for the decision-making process, is used and presented. This process is based on a cost model and a defined placement goal which is cost efficiency. The difference between the identified actual costs and the estimated costs from the cost model is derived into a force model. The forces affect the application placement, such that the operating force triggers a sequence of base operations on the applications which leads to an adapted application placement.

The implementation of the presented approach for automatic initial and ongoing application placement is carried out as a functional extension of the middleware Rebeca, which itself acts as a uniform execution environment that controls the application placement. The additional functionalities are integrated as *plugins*. The implemented and integrated functionalities serve as a proof of the presented concept and as a basis for a function and feasibility-oriented evaluation. The evaluation considers several different resource distribution scenarios and proofs that the newly added plugins of Rebeca in conjunction with simulation techniques lead to significant cost savings in most scenarios.

Vorwort

Danksagung

Auch wenn ich die vorliegende Dissertation allein verfasst habe, gibt es doch Personen, ohne deren Betreuung, Hinweise und Unterstützung diese Arbeit nicht möglich gewesen wäre.

Zunächst danke ich meinem Betreuer Herrn Prof. Dr.-Ing. Gero Mühl für die Inspiration und Betreuung in den zurückliegenden Jahren. Ein besonderer Dank gilt auch Herrn Dr.-Ing. Helge Parzyjegla für die Unterstützung und die wertvollen Hinweise. Ebenso möchte ich mich bei Herrn Prof. Dr. rer. nat. Clemens H. Cap und Herrn Prof. Dr. rer. nat. Odej Kao dafür bedanken, dass sie sich bereiterklärt haben, meine Dissertation zu begutachten. Ein weiterer Dank gilt auch den Professoren, Mitarbeitern und Stipendiaten des Graduiertenkollegs „Multimodal Smart Appliance Ensembles for Mobile Applications“, inbesondere dessen Leiter Herrn Prof. Dr.-Ing. Thomas Kirste. Meinen Kollegen vom Lehrstuhl „Architektur von Anwendungssystemen“ Nikolaus Jeremic, Matthias Prellwitz und Christian Wernecke danke ich für die angenehme Arbeitsatmosphäre und den Hilfskräften Steffen Steiner und Stephan Paul für ihre Implementierungstätigkeiten.

Die Promotion war ein Zeitraum, den ich ohne die Motivation und die Unterstützung vieler lieber Menschen kaum erfolgreich hätte zu Ende bringen können. Für die Geduld, Unterstützung, Ablenkung und Motivation danke ich meiner Verlobten Frau Dr. rer. nat. Katja Will sowie meinen Eltern Linhard und Martina Seib für all ihre Geduld und Unterstützung über die Jahre meines Studiums und der Promotion.

Ein letzter Dank gilt meinen Freunden und denen, die mich geprägt haben und nicht mehr unter uns sind.

Publikationen

Die Grundlagen des in Kapitel 3 dieser Arbeit vorgestellten Konzepts und die Vorarbeiten zur adaptiven, verteilten Detektion komplexer Ereignismuster als Anwendungsbeispiel aus Kapitel 5 wurden bereits veröffentlicht. In der ersten Publikation [193] dazu wurde das Konzept der verteilten, adaptiven Platzierung komplexer Ereignismustererkennungen als eine Anwendungsmöglichkeit selbstorganisierten Verhaltens eingeführt. In der späteren Veröffentlichung [194] liegt der Fokus auf der verteilten, selbstorganisierten Detektion komplexer Ereignismuster und es werden erste Simulationsergebnisse präsentiert. In dieser zweiten Veröffentlichung wird allerdings schon die konzeptionelle Erweiterung des Ansatzes auf allgemeine Anwendungskomponenten skizziert.

Meine Forschung entwickelte sich aus dem Teilprojekt *Model-Driven Development of Self-organising Control Applications* (*MODOC*) des *Organic Computing* (*OC*)-Schwerpunktprogramms der Deutschen Forschungsgemeinschaft. Im Abschlussbericht des *OC*-Schwerpunktprogramms findet sie sich als ein Aspekt des Teilprojekts *MODOC* [173] wieder. Diese Veröffentlichung entstand in Zusammenarbeit mit den anderen Mitgliedern der Arbeitsgruppe Helge Parzyjegla, Arnd Schröter, Sebastian Holzapfel, Matthäus Wander, Jan Richling, Arno Wacker, Hans Ulrich Heiß, Gero Mühl und Torben Weis.

Am Anfang des Kapitels 5 wird in Abschnitt 5.2 besonders auf die Modellierung komplexer Ereignismuster eingegangen. An dieser Stelle gab es Anknüpfungspunkte mit dem Lehrstuhl Softwaretechnik der Universität Rostock. So entstand zusammen mit Jens Brüning, Peter Forbrig und Michael Zaki eine weitere Publikation [27].

Neben den begutachteten Veröffentlichungen wurde das Forschungsvorhaben jährlich in einem Beitrag innerhalb des Veröffentlichungsbands der deutschen Graduiertenkollegs des Fachbereichs Informatik aufgeführt [10, 117, 211].

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung und Motivation

Inhalt

1.1 Ubiquitäre Anwendungen und intelligente Umgebungen

Intelligente Umgebungen bestehen aus miteinander vernetzten Geräten wie Sensoren, Aktoren und Computern, welche eingebettet in einer physikalischen Umgebung gemeinsam Dienste für Nutzer erbringen (z.B. diesen mit gewünschten Informationen versorgen) [181], wobei die eingebetteten Geräte zusammen ein Geräteensemble bilden. Die vom Nutzer gewünschten Dienste werden durch Anwendungen oder durch deren Zusammenarbeit erbracht. Die Ausführung der Anwendungen ist jedoch nicht länger auf ein Gerät oder eine Gruppe bestimmter Geräte beschränkt, sondern Anwendungen sollen auf Wunsch der Nutzer auf jedem Gerät des Geräteensembles ausgeführt werden können. Anwendungen sollen also ubiquitär verfügbar sein.

Ubiquitäre Anwendungen finden in zunehmendem Maße Eingang in das tägliche Leben. So werden Anwendungen „mobil“ und immer häufiger nicht länger nur auf PCs ausgeführt, sondern auch auf tragbaren Geräten wie Smartphones. Die Nutzer erwarten, dass die durch die Programme bereitgestellte Funktionalität möglichst zu jeder Zeit verfügbar ist, unabhängig von der genutzten Hardware und dem Standort der Nutzer. Ähnlich wie die Geräte selbst, soll auch die Funktionalität dem Nutzer jederzeit zur Verfügung stehen. So werden Smartphones als Teil der persönlichen Ausstattung wahrgenommen. Sie begleiten und versorgen den Nutzer mit allerlei Informationen aus dem aktuellen Kontext, mit Informationen zu bestimmten Orten und Plätzen, Hotels und Restaurants oder Straßenkarten und Fahrplänen des Nahverkehrs. Für die Erkennung des aktuellen Kontextes und der daraus angepassten Dienste werden Informationen benötigt, welche die aktuelle Position des Nutzers in einem Raum bestimmen, aber auch Informationen die von umgrenzenden Geräten bereitgestellt werden.

Neben der Einbindung portabler Geräte werden ubiquitäre Anwendungen auch im Umfeld der Vernetzung von Fahrzeugen und mit anderen Geräten eingesetzt. Die von Fahrzeugen bzw. ihren Sensoren gewonnenen Daten können so bspw. in einem Verbund heterogener Geräte für intelligente Verkehrsleitsysteme genutzt werden. Darüber hinaus ist es möglich, Anwendungen so zu gestalten, dass sie sowohl auf der Softwareplattform des Fahrzeugs, als auch auf einem Laptop oder Smartphone ausgeführt werden können. So ist es gerade im Motorradbereich üblich, Motorradtouren im Voraus auszuarbeiten und anschließend abzufahren. Bisher ist es jedoch nötig, die Routenauswahl am PC zu erstellen und anschließend auf das Gerät zu übertragen. Im Vergleich zur ebenfalls möglichen Routenplanung am Navigationsgerät selbst ist dies die komfortablere Vorgehensweise. Eine ubiquitäre Anwendung könnte den Schritt zwischen Ausarbeitung am PC und der Übertragung überflüssig machen. Außerdem können Informationen anderer Fahrer, bspw. über Streckensperrungen, Unfälle und sonstige Hindernisse dynamisch mit in die Routenplanung eingebracht werden. Eine weitere Anwendung ist der Zugriff auf Fahrzeuginformationen und -einstellungen. Wenn es das Fahrzeug zulässt, kann der Nutzer mittels Anwendungen Fahrzeugeinstellung

und -abstimmung (z.B. der Motorkennlinie oder der Federelemente) nicht mehr am Fahrzeug selbst, sondern von einem persönlichen Gerät aus vornehmen, wobei er die zu erwartenden Straßenverhältnisse mit einbezieht.

Ein vielzitiertes Anwendungsbeispiel ist auch das intelligente Haus bzw. die intelligente Wohnung. Hier sind in einem Haus/einer Wohnung eine Vielzahl von Sensoren untergebracht, welche für jeweilige Beobachtungspunkte, z.B. Temperatur, Luffeuchtigkeit, Sonneneinstrahlung und Lichtstärke oder von Fenstern und Türen, die Zustände erkennen und propagieren. Diese Informationen werden nun so dargestellt, dass der Nutzer sie bspw. mittels seines Smartphones oder Rechners auswerten kann. Mithilfe geeigneter Anwendungen und Schnittstellen kann er anschließend die Steuerung von Elementen wie der Verschattung und Belichtung, Heizung und Klimatisierung sowie der in einem Haus vorhandenen Geräte wie Radio und Fernsehgerät oder von Einbruchmeldeanlagen übernehmen. Die Gerätesteuerung kann explizit durch den Nutzer für jedes Gerät vorgenommen werden, was jedoch für den Nutzer selbst schnell unübersichtlich werden kann. Durch eine zwischengeschaltete Anwendung, welche aufgrund von bekannten Vorlieben des Nutzers oder erkannten Intentionen die Regelung der Geräte übernimmt, wird die Steuerung der intelligenten Umgebung nutzerfreundlicher und wirkt sich auch positiv auf die Akzeptanz der Umgebung selbst aus.

1.1.1 Anwendungsgebiete intelligenter Umgebungen

Neben den bereits angesprochenen Szenarien wie der Vernetzung von Fahrzeugen mit mobilen Geräten und dem Informationsaustausch sowie der Regelung von Gebäudesystemen und Unterhaltungselektronik in Wohnungen, werden im Folgenden weitere Anwendungsszenarien betrachtet, die die Zusammenarbeit verschiedener Geräte und Anwendungen in unterschiedlichen Umfeldern beleuchten.

Medizinische Assistenzsysteme

Der Einsatz intelligenter Umgebungen ist im Bereich des Krankenhaus- und Pflegesektors für die Assistenz verschiedener Nutzergruppen sinnvoll. Zunächst wird in der vorliegenden Arbeit der Einsatz von Geräten und Anwendungen im Hinblick auf die betroffenen Patienten bzw. Pflegebedürftigen selbst betrachtet, anschließend wird daran erarbeitet, wie weitere Nutzergruppen von der Zusammenarbeit unterschiedlicher Anwendungen und Geräte profitieren können.

Menschen, die in Pflegeheimen oder Anlagen des betreuten Wohnens leben, benötigen in vielerlei Hinsicht Assistenz. So wird je nach Erkrankung und deren Stadium mehr oder weniger medizinische Assistenz benötigt. Möglicherweise muss ein Bewohner einer solchen Einrichtung ständig überwacht werden. Dies geschieht üblicherweise durch das Pflegepersonal mittels Kameras oder durch an Patienten angeschlossene Geräte, deren Beobachtungen ebenfalls durch das diensthabende Personal ausgewertet werden. Neben der Überwachung der Patienten mit dem Ziel einer schnellen Reaktion bei Notfällen wird auch Assistenz

für Bewohner oder Patienten benötigt, die noch weitgehend selbstständig ihr Leben bestreiten, jedoch auf Unterstützung beim Erledigen täglicher Aufgaben (z.B. beim Kochen) angewiesen sind.

Im ersten Fall spielt die Erkennung von Notfällen eine entscheidende Rolle, sodass Mediziner in einem solchen Fall schnell reagieren können. Die Erkennung von Notfällen ist jedoch nicht trivial. So können einerseits bereits Über- bzw. Unterschreitungen einzelner Indikatoren auf einen Notfall hinweisen, während Notfälle andererseits nur im Zusammenspiel verschiedener, einzeln gesehen unkritischer, Indikatoren erkennbar sind. Diese nur im Verbund unterschiedlicher Informationsquellen detektierbaren Notfallsituationen benötigen zur Wahrnehmung Anwendungen, welche die einzelnen Informationen sammeln und nach Mustern auswerten können. Voraussetzung dafür ist jedoch, dass die Geräte und Anwendungen, welche einzelne Indikatoren liefern, über eine „gemeinsame Sprache“ verfügen und miteinander verbunden sind. Wenn jedoch, gerade im Gesundheitssektor, proprietäre Systeme miteinander kommunizieren müssen, ist dies ohne Frage eine Herausforderung, da zunächst eine gemeinsame Kommunikationsstrategie und Kommunikationsplattform gefunden werden müsssen.

Gleichzeitig muss beachtet werden, dass Geräteensembles nicht über ihre gesamte Lebensdauer aus denselben, fest miteinander verbundenen, Geräten bestehen. So ist während der Aufenthaltsdauer von Patienten oder Heimbewohnern die technische Ausstattung der Zimmer mit Geräten den jeweiligen Bedürfnissen der Patienten anzupassen. Dies ist nötig, da nicht zuletzt aus wirtschaftlichen Gründen nicht in jedem Zimmer eine komplette Austattung mit allen verfügbaren Geräten möglich und auch nicht sinnvoll ist. So bedingt der Einzug einer neuen Person einen persönlichen Gerätebedarf, ebenso wie ein veränderter Gesundheitszustand bestimmte Geräte erfordert. Daher muss eine flexible Infrastruktur aufgebaut werden, die es ermöglicht, Geräte je nach Bedarf dem Ensemble hinzuzufügen oder sie aus diesem zu entfernen.

Neben der schnellen Alarmierung bei Notfällen können Assistenzsysteme auch bei anderen Situationen eine wichtige Rolle spielen, bspw. bei der Unterstützung von Demenzpatienten. Prinzipiell basiert die Assistenz auch hier einerseits auf der Erkennung von Notfällen und andererseits auf der Erkennung der Intention von Nutzerverhalten. Die Erkennung von Notfällen beruht wiederum auf der Kombination aus Einzelindikatoren und Mustern von Informationen. Durch die Detektion von Umweltzuständen sowie der Detektion von Nutzerabsichten und Nutzerwünschen wird dem Anwender gezielt Hilfestellung gegeben, bspw. durch das Einblenden einer Wegbeschreibung oder eines Kochrezeptes auf einem Smartphone, Tablet-PC oder Fernsehgerät.

Assistenzsysteme dienen, neben Patienten oder Bewohnern der Einrichtungen, auch der Qualitätssicherung und der Vereinfachung von Abrechnungssystemen. Im Sinne der Qualitätssicherung geben Assistenzsysteme wertvolle Hinweise zu Patienten wie Unverträglichkeiten und Allergien, aber auch aktuelle Medikationen und bereits durchgeführte Tätigkeiten. Auf diese Weise können Informationsdefizite, z.B. bei wechselnden Mitarbeitern, überwunden und nicht sinnvolle

Arbeiten vermieden werden. Auch ohne papiergebundene Akten, allein durch Anzeigen auf einem im Raum vorhandenen Fernsehgerät oder dem Smartphone der Pflegekräfte, lassen sich unnötige Doppelbehandlungen (z.B. nochmalige Blutabnahme) vermeiden. Da Tätigkeiten im Zusammenhang mit dem Patienten detektiert oder zumindest dokumentiert werden, können die gewonnenen Informationen für die Abrechnung der durchgeführten Leistungen und gleichzeitig als deren Nachweis im Sinne der medizinischen Dokumentation dienen.

Alltägliche Assistenz

Die verfügbaren Informationen in Assistenzsystemen sind vielfältig und können so für unterschiedliche Nutzergruppen - vom Patienten über Pflegekräfte bis hin zur Qualitätssicherung und Abrechnung - eingesetzt werden. Der Weg zur alltäglichen Assistenz ist nicht mehr weit. Ein Beispiel dafür ist die „Media-follow-me“-Anwendung in einem vernetzten, intelligenten Haus. Hier sind innerhalb eines Hauses verschiedene Sensoren angebracht, die in der Lage sind zu erkennen, ob und welche Personen sich in einem Raum aufhalten. Bewegt sich nun eine Person durch das Haus und hört bspw. ein bestimmtes Radioprogramm, wird in jedem Raum, den die Person betritt, das gewünschte Programm abgespielt. Assistenz kann im Hintergrund ausgeführt werden, sodass dem Nutzer aufgrund seiner vorher propagierten Präferenzen und erkannten Intentionen im Zusammenspiel mit Umweltsituationen diese zuteil wird. Dabei beruht die Assistenz, neben der Erkennung von Mustern und Intentionen, vor allem auf der Vernetzung unterschiedlicher Geräte und Anwendungen.

1.1.2 Kommunikation in intelligenten Umgebungen

Geräteensembles in intelligenten Umgebungen sind in ihrer Struktur nicht starr, sondern ändern sich im Laufe der Zeit. So werden bei der Vernetzung von Fahrzeugen laufend neue Verbindungen zwischen Fahrzeugen auf- und wieder abgebaut, beim intelligenten Haus werden Geräte durch Nutzer in den Verbund hinein- und wieder hinausgetragen und selbst beim Kranken- und Pflegesektor wird das Geräteensemble gemäß den Bedürfnissen der Patienten verändert.

Dynamische Umgebungen, bei denen Geräte und Anwendungen geplant dem Netzwerk beitreten oder es verlassen, sind an dieser Stelle nicht zu verwechseln mit Fehler- oder Ausfalltoleranz. Die Dynamik der Zusammensetzung der Geräteensembles zieht jedoch Konsequenzen auf der Anwendungsebene nach sich. Gehört ein Gerät nicht mehr zum Ensemble, ist auch eine darauf ausgeführte Anwendung nicht mehr verfügbar. Dies kann dazu führen, dass der Nutzer direkt betroffen ist und nicht mehr die gewünschten Dienste erhält oder dass indirekt andere Anwendungen die gewünschten Zulieferungen entweder nicht ganz oder nicht mehr in der gewünschten Qualität erhalten. Neben der Dynamik aufgrund ein- und austretender Geräte in das bzw. aus dem Ensemble kann auch den Anwendungen selbst eine Dynamik innewohnen. Anwendungen

können im Laufe ihrer Ausführungsdauer ihr Verhalten ändern, sodass sie bspw. aufgrund höherer Eingangslast auch eine höhere Verarbeitungslast aufweisen. Ist dies bei ausreichenden Ressourcen grundsätzlich unproblematisch, kann es jedoch bei knappen Ressourcen zu Engpässen bei der Verarbeitungs- und Kommunikationsbandbreite führen.

Die Geräte innerhalb der intelligenten Umgebungen sind zudem heterogen. So ist im Ensemble eine weite Bandbreite von Geräten anzutreffen. Dazu gehören u.a. (mobile) Sensoren, Smartphones, (Tablet-) PCs, Fahrzeuge oder leistungsfähige Server. Die im Ensemble zusammengeführten Geräte unterscheiden sich bezüglich ihrer Architektur, ihrer Leistungsfähigkeit, dem verwendeten Betriebssystem, der Verfügbarkeit verschiedener Dienste oder der Nähe zu bestimmten Geräten und Anwendungen.

Kommunikation und Kooperation in intelligenten Umgebungen müssen so aufgebaut sein, dass sie einerseits von der bestehenden, heterogenen Gerätearchitektur und den Geräteressourcen abstrahieren, andererseits aber auch dazu in der Lage sind, die Dynamik des Geräteensembles zu unterstützen. Die Dynamik ist sowohl der teilnehmenden Geräte des Ensembles, als auch dem Verhalten der ausgeführten Anwendungen geschuldet.

Die Kommunikation in heterogenen dynamischen Umgebungen muss dabei sicherstellen, dass ihre Strukturen so aufgebaut sind, dass Geräte unabhängig von ihrer technischen Ausstattung und den durch sie ausgeführten Anwendungen dem Informationsaustausch beitreten können.

Eine Möglichkeit der Kommunikation ist die Darstellung und der Austausch von Informationen in Form von *Ereignissen*. Ereignisse, im Englischen häufig als „Happenings of Interests“ bezeichnet, spiegeln Informationen über die Veränderungen von Zuständen der Umwelt, von Geräten oder Anwendungen wider, welche durch Sensoren detektiert werden. Diese Veränderungen werden in Form von Ereignisnachrichten, im Folgenden nur als Ereignis bezeichnet, propagiert. Durch den Austausch von Ereignissen können Anwendungen miteinander kommunizieren, ohne dass konkrete Geräteadressen und Ports bekannt sein müssen. Auf diese Weise wird die Möglichkeit einer inhaltsbasierten, entkoppelten Kommunikation geschaffen.

Eine Umsetzungsvariante ereignisbasierter Kommunikation ist *Publish/Subscribe*. Hier emittieren *Produzenten* Ereignisse, die mittels eines Notifikationsdienstes an alle weitergeleitet werden, die sich im Vorfeld für diesen Typ von Ereignissen subskribiert haben (*Subskribenten*). Die Subskribenten spezifizieren ihre Interessen in Form von gewünschten Ereignistypen und ggf. Einschränkungen bezüglich des gewünschten Inhalts in Form von *Subskriptionen*. Der zwischen Produzenten und Susbribenten geschaltete *Notifationsdienst* ist für die Weiterleitung der Ereignisse an die interessierten Subskribenten zuständig. Bei inhaltlicher Übereinstimmung von abgegebenen und veröffentlichten Subskriptionen werden diese zu den interessierten Anwendungen weitergeleitet.

Publish/Subscribe hat sich in der Vergangenheit im Umfeld intelligenter Umgebungen als geeignetes Kommunikationsparadigma bewährt. Zur Überwindung der im Geräteensemble bestehenden Heterogenität, insbesondere im Hinblick auf die eingesetzte Plattform (Hardware- und Betriebssystemebene), ist der Einsatz einer Middleware sinnvoll, welche zwischen der Anwendungs- und Betriebssystemebene agiert und von der zugrunde liegenden Plattform abstrahiert. Durch die Implementation von Publish/Subscribe durch die Middleware können Produzenten und Subskribenten lose miteinander gekoppelt werden und Anwendungen in einem heterogenen Geräteensemble kommunizieren.

1.2 Kooperation in intelligenten Umgebungen

Durch die Kommunikation mittels einer Publish/Subscribe-basierten Middleware können Anwendungen in einem heterogenen Geräteensemble miteinander interagieren und kooperieren. Um den Nutzern in intelligenten Umgebungen Unterstützung bzw. Assistenz bieten zu können, ist die Ausführung unterschiedlicher kooperierender Anwendungen nötig. Dabei sind die Anwendungen, ähnlich der sie ausführenden Geräte, heterogen. Das bedeutet, sie unterscheiden sich hinsichtlich ihres Verhaltens (z.B. Kommunikationsintensität und Ressourcenkonsumption) und somit auch anhand ihrer Anforderungen an ihre jeweilige Ausführungsumgebung, bspw. bezüglich Rechenzeit, Speicherkapazität oder der Nähe zu bestimmten Geräten oder anderen Anwendungen.

Wenn davon ausgegangen werden kann, dass die intelligente Umgebung eine gemeinsame Ausführungsplattform für alle Anwendungen bietet, sind alle Anwendungen auf jedem Gerät der Umgebung ausführbar. Aufgrund der heterogenen Geräte und der unterschiedlichen Anforderungen der Anwendungen an die ausführenden Geräte, sind Anwendungen jedoch nicht von jedem Gerät sinnvoll ausführbar. Wobei sinnvoll bedeutet, dass Anforderungen der Anwendung nicht oder teilweise nicht erfüllt werden. Somit sind die Anforderungen, die Nutzer an die Anwendungen stellen, nicht in jedem Fall erfüllbar. Das bedeutet letztendlich, dass in einer intelligenten Umgebung die vom Nutzer gewünschten Anwendungen, wenn diese den definierten Ansprüchen zu genügen haben, nicht zwangsläufig auf dem von ihm bevorzugten Gerät ausgeführt werden können.

Da Anwendungen, egal wo sie im Geräteensemble ausgeführt werden, über Ereignisse entkoppelt miteinander kommunizieren und durch die einheitliche Ausführungsplattform prinzipiell von jedem Gerät ausgeführt werden können, spielt die Platzierung von Anwendungen innerhalb des Geräteesembles eine entscheidende Rolle für die Einhaltung der geforderten Eigenschaften von Anwendungen gegenüber den Nutzern. Welche Anwendung nun durch welches Gerät ausgeführt wird, ist wiederum abhängig von den Anforderungen der Anwendungen an das ausführende Gerät mit seinen verfügbaren Ressourcen.

Die Platzierung von Anwendungen bzw. ihre Zuordnung zu den ausführenden Geräten ist jedoch nicht trivial, wenn sie einem Ziel, bspw. der Erreichung einer

bestimmten Dienstgüte oder einer ausgeglichenen Ressourcenauslastung, dient. Sie ist es vor allem deshalb nicht, weil es durch die begrenzten Ressourcen der ausführenden Geräte und die unterschiedlichen Anforderungen zu überlappenden Anforderungen, Überlastungen und Verdrängungen von bereits ausgeführten Anwendungen kommen kann. Im Grunde stellt sich die Anwendungsplatzierung als ein Platzierungsproblem dar, welches bereits in einem statischen Umfeld als NP-schwer nachgewiesen wurde [20].

Nach derzeitigem Kenntnisstand werden Anwendungen in den meisten Fällen auf einem Gerät in unmittelbarer Nähe des Nutzers oder, durch einen Administrator gesteuert, an einer beliebigen Stelle im Geräteensemble deployt. Dieses Vorgehen ist sicher ein probates Mittel um Anwendungen auszuführen, bei knappen Ressourcen und Anforderungen an und durch die Anwendungen jedoch nicht mehr Mittel der Wahl. Da außerdem, gerade mit zunehmender Größe des Geräteensembles, die Platzierungsmöglichkeiten immer vielfältiger und gleichzeitig unübersichtlicher werden, ist eine „sinnvolle“ Verteilung von Anwendungen durch manuelle Zuordnung mitunter nicht erreichbar. Doch selbst wenn eine sinnvolle Zuordnung von Anwendungen innerhalb des Ensembles gelungen ist, ist sie durch die im System vorhandene Dynamik nicht dauerhaft. Neben dem ohnehin NP-schweren Platzierungsproblem in einem statischen Umfeld kommt nun eine zusätzliche Schwierigkeitsstufe durch die Dynamik dazu.

Ziel der Arbeit ist die Vorstellung eines Ansatzes, der Anwendungen in einem dynamischen Umfeld sinnvoll verteilt, sodass Anforderungen von Anwendungen innerhalb des zugrunde liegenden Geräteensembles möglichst vollständig erfüllt werden. Auf diese Weise wird dem Nutzer eine bestmögliche Qualität der ausgeführten Anwendungen ermöglicht. Neben dem Finden einer gültigen, initialen Platzierung liegt das Hauptaugenmerk dabei auf der Adaption der Platzierung an die dynamischen Veränderungen innerhalb des Geräteensembles.

1.3 Anwendungsplatzierung in intelligenten Umgebungen

In der Vergangenheit wurde viel Forschungsarbeit im Umfeld der Kommunikation von Anwendungen und Geräten in heterogenen Umgebungen geleistet. Hierbei wurden insbesondere die Architektur und das Zusammenwirken von Kommunikationsmechanismen in heterogenen Geräteensembles untersucht und entwickelt. So ist Publish/Subscribe bspw. durch adaptive Auswahl von Routingalgorithmen in der Lage, diese an eine sich ändernde Gerätestruktur anzupassen.

Der Bereich der adaptiven Anwendungsplatzierung wurde bisher nur wenig beleuchtet. Dabei fehlt vor allem eine Betrachtung, wie Anwendungen in intelligenten Umgebungen sinnvoll zerlegt und verteilt werden und anschließend als Komponenten miteinander kooperieren können.

Die Umsetzung der intelligenten Platzierung bringt eine Reihe von Herausforderungen mit sich. Während bei der initialen Platzierung von Anwendungen vor allem der Abgleich von Anwendungsanforderungen und verfügbaren Ressourcen ein Rolle spielt, muss aufgrund der Dynamik in der Umgebung und im Anwendungsverhalten die Platzierung kontinuierlich den Anforderungen und den sich ändernden Bedingungen angepasst werden. Durch die Anpassung wird die Qualität der Anwendungsausführung verbessert und dem Wesen intelligenter Umgebungen angeglichen.

1.3.1 Derzeitige Situation

Bisher werden Anwendungen für die persönliche Assistenz hauptsächlich auf dem Gerät deployt, auf dem auch die Interaktion mit dem Nutzer stattfindet. Dies resultiert vor allem aus der Tatsache, da der Nutzer meist selbst für die Zuordnung von Anwendungen zu den ausführenden Geräten sorgt. Diese Vorgehensweise liegt maßgeblich daran, dass die Anforderungen von Anwendungen entweder nicht bekannt oder im voraus nicht abschätzbar sind und der Nutzer darüber hinaus nicht das nötige Wissen über freie Ressourcen hat. Denn besonders bei großen Ensembles kann der Nutzer die Eigenschaften einzelner Geräte nicht mehr überblicken. Und auch wenn anstelle der späteren Nutzer die Platzierung durch einen Administrator gesteuert wird, bestehen bei immer größeren Geräteensembles die gleichen Probleme durch Informationsdefizite. Neben der Platzierung ist auch das Deployment selbst nicht problemlos durchführbar, da Anwendungen auf unterschiedlichen Geräten und verteilt ausgeführt werden. Selbst bei gleichen Ausführungsplattformen sind mitunter benötigte Bibliotheken nicht vorhanden und müssen (notfalls manuell) nachgeladen werden. Manuelles Deployment wird möglicherweise dadurch erschwert, dass für das Deployment von Anwendungen auf beliebigen Geräten entsprechende Zugriffsrechte fehlen.

Wird, aller Probleme zum Trotz, eine gültige initiale Platzierung von Anwendungskomponenten gefunden, um die Anwendungen dort ausführen zu können, so ist die gefundene Lösung durch die Dynamik des Systems und der Anwendungen in absehbarer Zeit veraltet. Wenn Geräte aus dem Ensemble ausscheiden bzw. nicht mehr verfügbar sind, gilt dies auch für die durch sie ausgeführten Anwendungen. Neben diesen direkten Auswirkungen sind auch indirekte Auswirkungen möglich, können sich doch durch hinzugekommene oder fehlende Geräte die Kommunikationswege ändern, sodass mit Auswirkungen auf die Dienstgüte von Anwendungen zu rechnen ist. Auch das veränderte Verhalten von Anwendungen kann sich durch gesteigerten Erreignisfluss auf die Auslastung von Kommunikationssourcen auswirken und dadurch indirekt zu möglichen Engpässen und der Nichterreichung von Anforderungen führen.

Die genannten Unzulänglichkeiten sind nicht nur auf das Gebiet der dynamischen, heterogenen Geräteensembles beschränkt, sie sind in diesem Umfeld jedoch besonders augenscheinlich. Die vorliegende Arbeit beschäftigt sich daher

mit der Entwicklung und Darstellung von Verfahren zur automatischen Platzierung von Anwendungen in einem heterogenen, dynamischen Umfeld.

1.3.2 Probleme derzeitiger Lösungen

Die Platzierung von Anwendungen zu ausführenden Geräten ist ein Problem, bei dem auszuführende Aufgaben zu Ressourcen zugeordnet werden. Die Zuordnung von Anwendungen zu Geräten ist ein Planungsproblem, auch bekannt als *CPP* (*Component Placement Problem*) [126]. Bereits in einem nicht-dynamischen Umfeld liegt das Problem in der Klasse der NP-schweren Probleme [20]. Ein probates Mittel ist daher der Einsatz einer Heuristik. Im vorliegenden Anwendungsfall bietet es sich an, das Problem mit zwei Heuristiken anzugehen. So wird zunächst eine Heuristik benötigt, die den Abgleich von Anforderungen und verfügbaren Ressourcen sowohl für das initiale Deployment als auch für spätere Platzierungsentscheidungen übernimmt. Weiterhin wird eine Heuristik benötigt, die bestimmt, wann aufgrund der Dynamik eine Anpassung der Anwendungsplatzierung notwendig wird und welche Aktion dazu auszuführen ist.

Kein automatisches initales Deployment. Im ersten Problem, der Zuordnung von Anwendungen zu Ausführungsumgebungen, wird zunächst von vergleichbaren Geräten mit derselben Ausführungsumgebung ausgegangen. Die automatische Zuordnung von Anwendungen zu ausführenden Geräten wird häufig auf der Grundlage ausformulierter Constraints und Ressourcen mit Methoden der künstlichen Intelligenz vorgenommen [126]. Anhand dieser maschinenlesbaren Informationen agiert ein Algorithmus, der meist für einen bestimmten Anwendungszweck und ein bestimmtes Anwendungsmodell entworfen wurde. Als Erweiterung dieses Ansatzes und zur Umgehung von Entwicklungsaufwand, existieren inzwischen generalisierte Anwendungsbeschreibungsmodelle, die mittels einheitlicher Planungsalgorithmen ausgeführt werden können [129]. Wenn eine einheitliche Anwendungsausführungsumgebung vorliegt, lasssen sich Anwendungen auch an einer Stelle in einem Anwendungsrepository initialisieren und bei Bedarf an die gewünschte Stelle verschieben [166]. Leider haben die genannten Arbeiten den Nachteil, dass eine Adaption an sich ändernde (Umwelt-) Bedingungen oder sich änderndes (Anwendungs-) Verhalten nicht vorgesehen ist.

Keine dynamische Anpassung. Wie im vorigen Abschnitt zu lesen war, existieren zwar Ansätze für eine auf Anforderungen und verfügbare Ressourcen abgestimmte Platzierung von Anwendungen, allerdings sind diese nicht in der Lage, adaptiv auf veränderte Bedingungen zu reagieren. So werden die Anwendungen darüber hinaus als Ganzes platziert und ausgeführt, was wenig flexibel ist. Es muss daher möglich sein, Anwendungen möglichst flexibel zu platzieren, wobei eine flexible Platzierung sowohl hinsichtlich des Ausführungsortes, als auch hinsichtlich der Anwendung und ihrer Bestandteile gelten muss. Grundidee der adaptiven, verteilten Anwendungsausführung ist das Verschieben von

Anwendungskomponenten zu einem anderen Ausführungsort. Im Bereich dieser Anwendungsmigration finden sich Arbeiten zur nahtlosen Migration von Java-Anwendungen und -Prozessen, welche sich jedoch hauptsächlich mit dem Aufbau der sie ausführenden virtuellen Maschinen beschäftigen [24, 49, 145]. Sie befinden sich außerhalb der von uns gewünschten Abstrahierungsebene der Middleware. Darüber hinaus können Anwendungen nur als Ganzes verschoben werden, eine Aufspaltung in Komponenten ist nicht vorgesehen. Dieser Nachteil gilt auch für Ansätze, bei denen Anwendungen zwar zu verschiedenen Geräten repliziert werden können, die Verschiebung der Komponenten jedoch gleichsam einen Teil der Middleware nach sich zieht, in dem die Anwendungen gerade ausgeführt werden [108]. Dieses Vorgehen ist mit einem entsprechend hohen Overhead verbunden.

Um sich adaptiv veränderten Umwelt- und Anwendungseigenschaften anpassen zu können, muss die bestehende Situation allerdings zunächst reflektiert und mit alternativen Platzierungen verglichen werden. Diese Eigenschaft fehlt allerdings einigen Ansätzen. Bei dem in dieser Arbeit beschriebenen Verfahren werden die Reflektivität und Vergleichbarkeit allerdings vorausgesetzt.

Wurde in der Vergangenheit die Anpassung der Anwendungsplatzierung vor allem auf Grundlage der Verschiebung von Anwendungen als Gesamtes realisiert, bieten die Zerlegung von Anwendungen in ihre Komponenten und deren separate Platzierung eine größere Flexibilität. Das Zusammenspiel von der Zerlegbarkeit von Anwendungen und ihrer individuellen Platzierung wurde in der Vergangenheit wenig beachtet. Ein weiterer Schwachpunkt vorhandener Anwendungen ist die fehlende selbstorganisierte Auslösung von Operationen. Das bedeutet, dass die Entscheidung, wann welche Operation auf der Anwendung ausgeführt wird, durch eine zentrale Instanz getroffen werden muss. Besonders bei großen Geräteensembles kann sowohl eine Überlastung der Entscheidungsinstanz, als auch eine zusätzliche Beanspruchung der Kommunikationskanäle auftreten.

1.4 Beiträge der Arbeit

Die Betrachtung der momentanen Situation der Anwendungsverteilung hat gezeigt, dass das manuelle Deployment von Anwendungen in ubiquitären Systemen aufgrund der mangelnden Kenntnisse der Nutzer über Anforderungen und Ressourcen innerhalb der Geräteensembles nicht zu einem gewünschten guten Ergebnis führt. Die Vielzahl möglicher Anforderungen und verfügbarer Ressourcen erschweren allerdings auch die automatische Platzierung von Anwendungen innerhalb des Geräteensembles erheblich. Dazu kommt, im betrachteten Anwendungsfall, die dem Geräteensemble inhärente Dynamik aufgrund hinzukommender oder ausscheidender Geräte und die Veränderung von Anwendungsverhalten. In diesem Umfeld zeigt die vorgestellte Arbeit mögliche Lösungen auf, mit denen sowohl initiale als auch adaptive Anwendungsplatzierungsentscheidungen automatisiert werden können. Es werden Verfahren präsentiert, bei denen beim initialen Deplyoment Anwendungsanforderungen und die verfügbaren Ressourcen abgeglichen werden und Anwendungen automatisch deployt werden können.

Hauptbeitrag der Arbeit liegt in der Vorstellung einer adaptiven Platzierung von Anwendungen und Anwendungskomponenten gemäß aktueller Umweltsituationen und Anwendungsverhalten. Die Platzierungsentscheidungen werden auf der Grundlage eines ebenfalls vorzustellenden Kräftemodells getroffen, welches die auf die Anwendungsausführung einwirkenden Kosten als Kräfte einwirken lässt.

Die Beiträge der Arbeit verbessern die Handhabung ubiquitärer Anwendungen in heterogenen Geräteensembles. Sie erreichen dies dadurch, dass Anwendungen in einem heterogenen Umfeld anforderungsgerecht platziert und ausgeführt werden. Dabei spielen Anwendungsanforderungen und verfügbare Ressourcen eine wichtige Rolle, aber ebenso die auf die Anwendungsausführung einwirkenden Kosten. Durch die adaptive Verteilung der Anwendungen und Anwendungskomponenten innerhalb des dynamischen, heterogenen Geräteensembles stellt die Arbeit einen wichtigen Beitrag zur Verbesserung der Nutzbarkeit von ubiquitären Anwendungen in intelligenten Ensembles dar.

Selbstorganisierte Anwendungsverteilung. Kernstück der Arbeit ist ein Konzept zur selbstorganisierten Verteilung von Anwendungen und Anwendungskomponenten. Eingebettet als Dienst in die Middleware umfasst das Konzept sowohl die automatische initiale Platzierung und das Deployment als auch die selbstorganisierte, adaptive Anpassung der Anwendungsplatzierung.

Automatische initiale Platzierung und Deployment. Ein Teil des Konzepts beschreibt die bedarfsgerechte und ressourcenabhängige initiale Platzierung von Anwendungen bzw. Anwendungskomponenten in intelligenten Geräteensembles. In der vorliegenden Arbeit werden hierzu Verfahren vorgestellt, die beschreiben, wie Anwendungsanforderungen und verfügbare Ressourcen miteinander in Einklang gebracht und nach welchen Kriterien Platzierungsentscheidungen getroffen werden. Darüber hinaus umfasst das Konzept auch das Deployment von Anwendungen in den sie ausführenden Geräten.

Adaptive Anwendungsverteilung. Nach erfolgreicher initialer Platzierung ist die Anwendungsverteilung jedoch, aufgrund der im System und in den Anwendungen vorhandenen Dynamik, den Anforderungen und den dann aktuell verfügbaren Ressourcen bei Veränderungen laufend anzupassen, soll die Anwendungsausführung den formulierten Anforderungen gerecht werden. Im ersten Schritt wird ein Monitoring benötigt, welches einerseits die verfügbaren Ressourcen ermittelt, sowie andererseits ein Vergleich mit den Zielen der Anwender bzw. der Anwendungen durchführt. Sobald die gestellten Anforderungen nicht mehr ausreichend erfüllt werden oder die beanspruchten Ressourcen nicht mehr mit den vorhandenen übereinstimmen, werden Anwendungen adaptiv dort platziert und verschoben, wo die Anforderungen besser erfüllt werden. Das in dieser Arbeit vorgestellte Konzept dient damit der Sicherstellung der Anforderungserfüllung. Über diesen Umweg gelingt es, die Anforderungen der Nutzer an die Anwendungen in intelligenten Systemen zu erfüllen und qualitative Ansprüche gegenüber

den Anwendungen zu gewährleisten. Die Sicherstellung der Akzeptanz intelligenter Umgebungen durch adaptive Anwendungsplatzierung ist der Kernaspekt der vorliegenden Arbeit.

Das vorgestellte Konzept ermöglicht, Anwendungen anhand der verfügbaren Ressourcen und gestellten Anforderungen verteilt innerhalb des Geräteensembles der intelligenten Umgebung auszuführen. Dabei werden Anwendungen, soweit möglich, in Komponenten zerlegt und adaptiv im Geräteensemble verteilt ausgeführt. Die Verteilung der Anwendungen erfolgt auf Basis der Anforderungen an die Anwendungen und der verfügbaren Ressourcen anhand einer Heuristik auf der Grundlage eines Kräftemodells.

Neben der Entscheidung, ob und welche Aktion auf der Anwendung oder Anwendungskomponente ausgeführt wird, beschreibt das Konzept auch das Zusammenwirken der Kommunikationsinfrastruktur und der Operationen auf den Anwendungen. Insbesondere bei der Einbettung in eine ereignisbasierte Kommunikationsinfrastruktur beinhaltet das Konzept auch die Einhaltung einer konsistenten Reihenfolge von Ereignissen.

Simulative Evaluation mittels verteilter Detektion komplexer Ereignismuster. Das Konzept zur selbstorganisierten Anwendungsverteilung wird anhand der verteilten Detektion komplexer Ereignismuster evaluiert.

Zur Evaluation der in der Arbeit beschriebenen initialen Verteilung und der adaptiven Anwendungsplatzierung wird das Konzept in Rebeca umgesetzt. Dazu wird ein Anwendungscontainer in Rebeca integriert, der die zur Ausführung benötigten Ressourcen als Ausführungsumgebung bereitstellt. Weiterhin integriert der Anwendungscontainer auch einen Monitoringdienst, der die laufende Auslastung von Ressourcen einerseits, sowie die Anforderungen der Anwendungen andererseits ermittelt und somit die Grundlagen für die Platzierungsentscheidungen liefert.

In der Evaluation des vorgestellten Konzeptes wird mittels Simulation überprüft, ob die erwarteten Verbesserungen bezüglich der Anwendungsplatzierung erreicht werden. Das der Simulation zugrunde liegende Szenario basiert auf der Verteilung von Detektoren komplexer Ereignismuster in einem Netzwerk von Sensoren und Aktuatoren, wobei die sie ausführenden Geräte mit jeweils begrenzten Ausführungskapazitäten ausgestattet sind. Durch eine geschickte Verteilung von Detektoren und Teildetektoren werden möglichst wenige Ereignisse durch das Netzwerk geleitet und somit Kapazitäten eingespart.

Als Beispiel für die Zerlegung von Anwendungen in Komponenten und deren Verteilung im Geräteensemble ist das Szenario gut geeignet, lässt sich das vorgestellte Konzept doch für Anwendungen im Allgemeinen ebenso anwenden wie für spezialsierte Anwendungen aus dem Bereich der Detektion komplexer Ereignismuster. Gleichzeitig erfordert die Platzierung von Detektoren komplexer Ereignismuster ausreichend unterschiedliche Anforderungen, um die bedarfsge-

rechte Verteilung von Komponenten und die Auswirkungen des Platzierungskonzepts beurteilen zu können.

1.5 Aufbau der Arbeit

Nach diesem einleitenden Kapitel folgt in Kapitel 2 die Darstellung der grundlegenden Techniken und Konzepte, auf denen die vorliegende Arbeit aufsetzt. Das in dieser Arbeit vorgestellte Konzept zur Verteilung von Anwendungen basiert auf bereits vorhandenen, in eine Middleware integrierte Dienste. Einer dieser Dienste ist die Bereitstellung von Schnittstellen und einer Infrastruktur zur Kommunikation von Anwendungen. Ein solcher Kommunikationsdienst wird mit einer ereignisbasierten Middleware auf der Grundlage von Publish/Subscribe vorgestellt. Im weiteren Verlauf dieser Arbeit wird beschrieben, welche Techniken und Vorgehensweisen für das Anwendungsdeployment und die Anwendungsplatzierung im Allgemeinen existieren. Anschließend daran wird diskutiert, auf welchen Grundlagen Möglichkeiten zur Selbstorganisation im Umfeld ubiquitärer Anwendungen vorhanden sind.

Kapitel 3 beinhaltet den Hauptteil der Arbeit, nämlich die Vorstellung des Konzepts zum automatischen Deployment und der selbstorganisierten, adaptiven Platzierung von Anwendungen und Anwendungskomponenten. Nach einer einleitenden Motivation zur Notwendigkeit eines solchen Konzepts liegt der Fokus zunächst auf dem Anwendungsmodell in Abschnitt 3.3. Nach der Diskussion weiterer Anwendungsmodelle wird das dem Platzierungsansatz zugrunde liegende Anwendungsmodell vorgestellt. Hauptbestandteil des anschließenden Abschnitts 3.4 ist die Beschreibung des Konzepts für die automatische initiale Anwendungsplatzierung. Der Darstellung der laufenden Platzierung ist der darauf folgende Teil 3.5 des Kapitels gewidmet. Beide Teilabschnitte beschreiben ausführlich die jeweiligen Umgebungen, Ziele, die verwandten Arbeiten und die auf die Platzierung einwirkenden und resultierenden Kräfte.

Kapitel 4 legt die Umsetzung des Konzepts in die bereits bestehende Middleware REBECA dar. Nachdem zunächst das grundlegende Konzept von REBECA erklärt wird, folgen die Implemtierungsbeschreibungen des Anwendungscontainers und der Dienste für das automatische Deployment und die selbstorganisierte Platzierung ubiquitärer Anwendungen.

Kapitel 5 widmet sich der fallbasierten Umsetzung des vorgestellten Konzepts anhand der verteilten Detektion von komplexen Ereignismustern. Nach einer kurzen Motivation werden zunächst komplexe Ereignismuster und ihre Detektion im Allgemeinen vorgestellt, wobei an dieser Stelle Probleme bestehender Ansätze sowie die Notwendigkeit einer selbstorganisierten verteilten Detektion erläutert werden. Nachdem der Anwendungsfall des vorgestellten Konzepts ausführlich beschrieben wurde, widmet sich der folgende Teilabschnitt der Vorstellung der Simulationsumgebung und der Einführung der genutzten Beurteilungskriterien zur Bewertung der Nützlichkeit des vorgestellten Konzepts. Im dann folgenden

Teilabschnitt werden verschiedene Experimente zur initialen Verteilung und der laufenden Optimierung der Detektorplatzierung durchgeführt. Das Kapitel endet mit einer Diskussion der Ergebnisse.

Den Abschluss der Arbeit bildet Kapitel 6 mit der Beurteilung der Ergebnisse der vorgestellten Arbeit und einem Ausblick auf offene und zukünftige Forschungen.

Kapitel 2

Grundlagen

Inhalt

2.1 Einleitung

In Kapitel 1 wurde gezeigt, dass Kommunikation und Kooperation von Geräten und Anwendungen, besonders in heterogenen Umgebungen, essentiell wichtig sind. Gleichzeitig ist die Zuweisung von Anwendungen zu den sie ausführenden Geräten entscheidend für die Leistungsfähigkeit des gesamten Ensembles. So kann eine ungünstige Verteilung von Anwendungen, deren Ausführung Ressourcen unterschiedlich stark beansprucht, in Verbindung mit heterogenen Geräten zu Überlastungen oder Unterauslastungen führen. Dies führt zu Einbußen in der Leistungsfähigkeit des Gesamtnetzwerks, wobei hier weitere (Dienst-) Güteanforderungen an dieser Stelle noch gar nicht betrachtet wurden. Unterliegt das Geräteensemble zudem laufenden Veränderungen und muss die Anwendungsausführung daran angepasst werden, besteht auch die Notwendigkeit, die Anwendungsplatzierung laufend dem Umfeld und gemäß der geltenden Anforderungen anzupassen. Die laufende Platzierung benötigt auch einen Ausgangspunkt, eine bestehende initiale Platzierung. Sowohl die initiale, als auch die laufende Platzierung von Anwendungskomponenten sind, da es sich um eine Vielzahl von Anwendungen und vor allem möglichen Ausführungsorten handelt, für den Nutzer allein aufgrund der Vielzahl von Möglichkeiten nicht durchführbar.

Es wird daher ein Ansatz benötigt, der die Anwendungsplatzierung sowohl initial und angepasst an die Bedürfnisse von Anwendungen und die verfügbaren Ressourcen durchführt, als auch laufend den Umweltbedingungen an den Anforderungen und ggf. den Beschränkungen der Anwendungen anpasst, wobei beides automatisch durchzuführen ist. Die Darstellung eines solchen Ansatzes ist ein Hauptbestandteil dieser Arbeit. Zur Einordnung dieses Ansatzes in die Kommunikation und Kooperation von Geräten und Anwendungen in heterogene Geräteensembles dient das vorliegende Kapitel. Es enthält zunächst Techniken und Konzepte zur Kommunikation von Anwendungen in verteilten Umgebungen. Der Schwerpunkt der Betrachtung liegt dabei auf ereignisbasierter Kommunikation mittels Publish/Subscribe, welches sich als Kommunikationsparadigma insbesondere für ubiquitäre Anwendungen eignet. Anschließend wird ein Überblick über grundlegende Konzepte und Techniken des initialen Anwendungsdeployments sowie der laufenden Anpassung des Anwendungsdeployments gegeben. Da der in dieser Arbeit vorgestellte Ansatz zum initialen und laufenden Deployment von Anwendungen, auf den besonders in Kapitel 3 eingegangen wird, ohne Interaktion sowie Intervention mit bzw. durch einen Nutzer auskommt, wird in diesem Kapitel zusätzlich eine Einführung in die Grundlagen und Techniken der Selbstorganisation gegeben.

2.2 Publish/Subscribe

Ereignisbasierte Kommunikation (engl. *event-based communication*) beruht auf dem Austausch von Informationen über *Ereignisse* (engl. *events*), welche wahrnehmbare Geschehnisse von Interesse darstellen. Die Informationen über geschehene Ereignisse werden als *Notifikationen* verbreitet. Gegenüber älteren Kommunikationsparadigmen wie *Anfrage und Antwort* (engl. *request/reply*) bietet ereignisbasierte Kommunikation bspw. den Vorteil, dass kein Polling von Informationen nötig ist. Dies ergibt sich aus der Arbeitsweise des Kommunikationsparadigmas, da die Verteilung von Informationen nicht auf Initiative des bzw. der Empfänger durchgeführt, sondern durch das Auftreten eines Ereignisses getrieben wird. Es wird daher auch von *ereignisgetriebener* Kommunikation gesprochen. Empfänger von Informationen sind in der Lage, ihren Informationsbedarf selbst zu spezifizieren und selbstfokussiert zu verwalten. Tritt ein Ereignis in einem System auf und wird es detektiert, erfolgt anschließend die Veröffentlichung von einem *Ereignisproduzenten* in Form einer Notifikation. Entspricht diese einer abgegebenen Spezifikation von einem oder mehreren Interessenten, wird sie an die Interessenten weitergeleitet. Tritt kein Ereignis auf, werden keine Informationen veröffentlicht. Die in den Notifikationen verpackten Informationen enthalten neben den eigentlichen Nutzdaten über das Ereignis selbst häufig weitere Informationen zum Kontext der Daten, bspw. Informationen zum detektierenden Sensor wie Zeit- und Ortsinformationen.

Eine konkrete Umsetzung ereignisbasierter Kommunikation ist *Publish/Subscribe*. In Anlehnung an das sog. *Beobachter-Muster* (auch bekannt als *Observer bzw. Listener Pattern*) aus der Softwaretechnik [75] dient es der Weiterleitung von Informationen über die Änderung eines bestimmten Objekts. Wie in der allgemeinen ereignisbasierten Kommunikation existieren auch bei Publish/Subscribe unterschiedliche Rollen von Teilnehmern. So werden die Ereignisproduzenten als *Publisher* und die Ereigniskonsumenten als *Subscriber* bezeichnet. Publisher emittieren die beobachteten bzw. detektierten Ereignisse in Form von *Ereignisnotifikationen*, bzw. kurz *Notifikationen*. Diese werden nur an die Subscriber weitergeleitet, welche ihr Interesse an solchen Notifikationen in Form von *Subskriptionen* im Vorfeld spezifiziert haben. Die Verbindung zwischen Produzenten und Empfängern, also die Weiterleitung (engl. *routing*) von Notifikationen, obliegt dem *Notifikationsdienst*. Dieser übernimmt die publizierten Notifikationen und ist dafür verantwortlich, dass sie zu allen daran interessierten Konsumenten gelangen. Der Notifikationsdienst stellt damit sicher, dass Notifikationen und Subskriptionen inhaltlich übereinstimmen. Nicht übereinstimmende Notifikationen werden nicht weitergeleitet und herausgefiltert. Daher wird der Test auf die inhaltliche Übereinstimmung im Umfeld von Publish/Subscribe auch als *Filterung* (engl. *filtering*) bezeichnet.

Durch die Anordnung von Publishern, Subscribern, dem Notifikationsdienst und der ereignisgetriebenen Verarbeitung sind Ereignisproduzenten und Ereigniskonsumenten voneinander entkoppelt, was zunächst einmal asynchrone Kommunikation über räumliche und zeitliche Ebenen hinweg ermöglicht. Letzteres setzt al-

lerdings das Zwischenspeichern von Notifikationen durch den Notifikationsdienst voraus. Die Entkopplung von Sendern und Empfängern von Notifikationen bietet Vorteile bezüglich der Skalierbarkeit und bietet sich als Kommunikationsparadigma insbesondere in dynamischen Umgebungen an. Gleichzeitig ist es durch die Entkopplung jedoch schwieriger, die Einhaltung bestimmter Anforderungen von Anwendungen gegenüber den sie ausführenden Geräten zu garantieren. Dazu gehören bspw. Anforderungen nach einer verlustlosen Kommunikation und Einschränkungen bezüglich der Sichtbarkeit und die Sicherheit der Kommunikation. Insbesondere die Defintion von Sichtbarkeitsbereichen ist Forschungsgegenstand neuerer Arbeiten [169].

Zur Umsetzung des Notifikationsdienstes existieren unterschiedliche Möglichkeiten, wobei grundsätzlich nur zwei Funktionalitäten implementiert werden müssen, nämlich (1) das Bekunden von Interesse durch einen Subscriber und (2) die Publikation einer Ereignisnotifikation durch den Publisher. Mögliche Erweiterungen sind (3) das Widerrufen einer nicht mehr gültigen Subskription durch den Subscriber oder (4) die Bekanntgabe von zukünftig produzierten Ereignissen durch den Publisher mittels Ankündigungen (*Advertisements*) und analog zum Widerruf von Subskriptionen auch (5) der Widerruf von Ankündigungen.

Eine mögliche Umsetzung des Notifikationsdienstes mit asynchroner Kommunikation und der beschriebenen Funktionalität ist der zentrale Ansatz. Hier werden an eine zentrale Instanz des Notifikationsdienstes alle Subskriptionen und Notifikationen geleitet. Die Instanz filtert die eingehenden Notifikationen gemäß der Subskriptionen an einem Punkt und leitet sie in Richtung der Interessenten weiter. Anwendung finden solche Systeme häufig dort, wo strenge Anforderungen an Ausfallsicherheit, Konsistenzsicherung und Transaktionssicherheit gelten und gleichzeitig kein hoher Datendurchsatz gefordert wird [60]. Als Beispiele nennen Eugster et al. [60] die Arbeiten [132] und [26], die jeweils auf eine zentrale Datenbankarchitektur aufbauen. Nachteilig bei diesem Ansatz ist, dass es an zentraler Stelle zu einem Flaschenhals kommen kann, sowohl auf Ebene der Notifikationsverarbeitung, als auch auf der Ebene der Weiterleitung von Subskriptionen und Notifikationen. Der zentrale Ansatz skaliert nicht und ist nur bis zu einer begrenzten Anzahl von Kommunikationsteilnehmern sinnvoll umsetzbar.

Eine Weiterentwicklung des zentralisierten Ansatzes ist die Umsetzung mittels eines Bus-Systems. Im Unterschied zum zentralen Ansatz sind die Geräte, egal ob in der Rolle als Publisher oder Subscriber, nicht mit einer zentralen Instanz, sondern über ein Bussystem, d.h. einen gemeinsam genutzten Kanal, miteinander verbunden. In diesen Kanal hinein werden alle Notifikationen publiziert. Da jedes Gerät Zugriff auf den Kanal hat, erhält jeder Subscriber jede Notifikation. Eine Filterung findet dabei nicht statt. Da nur ein gemeinsamer Kanal existiert, kann auch hier ein Flaschenhals entstehen, das Verfahren skaliert ebenfalls nicht. Während beide Verfahren bei überschaubaren Netzwerken mit wenigen Teilnehmern im Sinne des Trade-offs zwischen Weiterleitungsaufwand und Flexibilität noch als sinnvoll erachtet werden können, führen eine Vielzahl von Publishern und Subscribern schnell zu einer Überlast an den zentralen Komponenten.

2.2.1 Verteilter Notifikationsdienst

Eine Möglichkeit der Ineffizienz durch mangelnde bzw. zu späte Filterung zu begegnen ist ein verteilter Notifikationsdienst. So lässt sich die Effizienz vor allem durch frühzeitiges Filtern und die Nicht-Weiterleitung von nicht nachgefragten Notifikationen steigern. Darüber hinaus bietet sich hier die Möglichkeit, die Anzahl der Instanzen des Notifikationsdienstes den Anforderungen anzupassen.

In einer verteilten Umgebung wird der Notifikationsdienst durch ein Netzwerk von miteinander verbundenen *Brokern* realisiert, welche die Notifikationen an die Konsumenten weiterleiten. Dazu verwaltet jeder Broker die Interessen der ihm angeschlossenen *lokalen Klienten* sowie die der Nachbarbroker mit Hilfe von *Routingtabellen*. Hierin finden sich jeweils Abbilder der aktuellen aus der jeweiligen Richtung stammenden Subskriptionen, so dass jeder Broker anhand der Routingtabelle entscheidet, in welche Richtung die ankommenden Notifikationen weitergeleitet werden. Dies geschieht durch Vergleich der eingehenden Notifikationen mit den in den Routingtabellen aufgeführten Subskriptionen. Die Gesamtheit der Routingtabellen spiegelt die im System vorhandenen Subskriptionen wider. Da grundsätzlich neue Broker hinzugefügt und Broker aus dem Brokernetzwerk ausscheiden können, bietet der verteilte Notifikationsdienst mit seiner Skalierbarkeit einen geeigneten Ausgangspunkt für die Kommunikation in dynamischen Umgebungen.

Wie die Routingtabellen angepasst werden, d.h. wie und welche Subskriptionen in die Routingtabelle aufgenommen werden und wie die Weiterleitung von Notifikationen konkret abläuft, entscheidet der *Routingalgorithmus*. Hier reicht die Spannweite möglicher Routingalgorithmen vom Fluten aller eingehenden Notifikationen an alle Nachbarbroker, über die selektive Weiterleitung der Notifikationen anhand verschiedener (inhaltlicher) Überdeckungsmethoden bis hin zu den auf den Anwendungszweck zugeschnittenen Mechanismen. Welche Techniken dazu existieren, ist Thema der folgenden Abschnitte 2.2.2 bis 2.2.3

2.2.2 Selektion von Notifikationen

Die Interessen von Subscribern werden in Form von Filtern spezifiziert, anhand dieser Filter werden abgegebene Subskriptionen und Notifikationen miteinander verglichen und bei Übereinstimmung die Notifikation ausgeliefert/weitergeleitet. Neben der interessensgerechten Weiterleitung von Notifikationen ermöglicht der Einsatz von Filtern den sparsamen Umgang mit Kommunikationsressourcen, da bei entsprechender Verteilung von Filtern frühzeitig nicht nachgefragte Notifikationen nicht weitergeleitet werden müssen und so Kommunikationsbandbreite eingespart werden kann. In der Vergangenheit haben sich, häufig in Abhängigkeit von der zugrunde liegenden Infrastruktur, Anwendungsumgebung und dem Anwendungsziel, unterschiedliche Filtermechanismen entwickelt, welche im Folgenden vorgestellt werden.

Kanalbasierte Filterung. Die *kanalbasierte Filterung* (engl. *channel-based filtering*) basiert auf der Idee, dass Notifikationen eines bestimmten Typs kategorisiert und in dafür festgelegten Kanälen publiziert werden. Subscriber abonnieren sich also nicht für die Notifikationen, sondern für Notifikationskanäle. Sie erhalten dann alle in diesem Kanal veröffentlichten Notifikationen. Publisher und Subscriber sind über einen gemeinsamen Kanal miteinander verbunden und nicht völlig voneinander entkoppelt. Die Effizienz des Ansatzes ist vor allem dadurch bestimmt, wie differenziert die Kanäle voneinander sind. Fasst ein Kanal eine Vielzahl unterschiedlicher Notifikationstypen zusammen, erhalten die Subscriber auch alle in diesem Kanal veröffentlichten Notifikationen. Zusätzlicher Filteraufwand (seitens der Subscriber) bzw. Weiterleitungsaufwand (seitens der Kommunikationsinfrastruktur) für eigentlich nicht gewollte Notifikationen sind dann die Folge, falls sich Subscriber nur für eine Teilmenge der in dem Kanal veröffentlichten Notifikationen interessieren.

Die kanalbasierte Filterung kann unterschiedlich umgesetzt werden. So können Kanäle bspw. mit einem Objekt gleichgesetzt werden, sodass sich Subscriber, die sich für alle Änderungen eines Objekts subskribieren, auch alle diesbezüglichen Notifikationen erhalten. Alternativ können sich Subscriber auch für Notifikationen, die von einem bestimmten Port eines Gerätes veröffentlicht werden, subskribieren. Kanalbasierte Kommunikation lässt sich bspw. durch IP-Multicast realisieren. Eine weitere praktische Umsetzung der kanalbasierten Filterung findet sich darüber hinaus in der Spezifikation des *Corba Event Service* [45].

Themenbasierte Filterung. Die *themenbasierte Filterung* beruht auf der Annahme, dass Ereignisse einem Thema zugeordnet werden können. Bis hierher gleicht die themenbasierte Filterung der kanalbasierten. Zusätzlich ermöglicht die themenbasierte Filterung jedoch eine hierarchische Kategorisierung, sodass Subscriber ihr Interesse allgemeiner fassen oder genauer spezifizieren können, um so den gewünschten Inhalt der ausgelieferten Ereignisse einzugrenzen. Diese Art der Filterung wird ebenfalls in produktiven Systemen eingesetzt. Durch die themenbasierte Filterung lassen sich unterschiedliche Messwerte für Raumtemperaturen bspw. dem Thema „Umgebungsbedingungen“ zuordnen, der Kurs der Aktie der Lufthansa dem Thema „Aktienkurse“. In den genannten Beispielen lassen sich die Interessen an „Umgebungsbedingungen“ („EnvironmentalConditions“) und „Aktienkurse“ („StockMarket“) genauer nach Themen wie „Temperatur“ („EnvironmentalConditions.Temperatures“), „Luftfeuchtigkeit“ („EnvironmentalConditions.Humidity“) und „Sonneneinstrahlung“ („EnvironmentalConditions.SolarRadiation“), respektive „Aktienkurse von Automobilunternehmen“ („StockMarket.Automotive“) oder „Aktienkurse von Fluggesellschaften“ („StockMarket.Airlines“) eingrenzen. Sind Subscriber allgemein an allen Umweltbedingungen wie Temperatur, Luftfeuchtigkeit und Sonneneinstrahlung interessiert, reicht es aus, wenn sie sich für das Thema „EnvironmentalConditions“ subskribieren. Interessieren sie sich lediglich für Temperaturen, ist eine Subskription auf „EnvironmentalConditions.Temperatures“ ausreichend. Analog gilt dies für das Interesse an Aktienkursen im Allgemeinen („StockMarket“)

und Luftverkehrsaktien („StockMarket.Airlines“). Eine weitere mögliche feingranularere Strukturierung wären in den Beispielen Subskriptionen auf einzelne Unternehmensaktien („StockMarket.Airlines.Lufthansa“) oder die Temperaturwerte einzelner Räume („EnvironmentalConditions.Temperatures.Rooms“). Die veröffentlichten Subskriptionen fassen in jedem Fall alle dem Thema zugeordneten Sub-Themen mit ein. So erhalten bspw. alle an „Umgebungsbedingungen“ („EnvironmentalConditions“) interessierten Klienten neben Ereignissen zur Raumtemperatur auch Informationen zur Luftfeuchtigkeit („EnvironmentalConditions.Humidity“) und der Lichteinstrahlung („EnvironmentalConditions.Radiation“), während Klienten, die sich für „Aktienkurse“ subskribieren, neben Ereignissen für Aktien von Fluggesellschaften auch Aktienkurse von Fahrzeugbauern („StockMarket.Automotive“) und Energieversorgern („StockMarket.UtilityProv“) etc. erhalten.

Generell sind Themen aussagekräftiger als Kanäle, jedoch besteht noch immer eine Bindung zwischen Publishern und Subscribern über die Themen, eine komplette Entkopplung findet auch hier nicht statt. Ein weiteres Problem ist die Aufteilung des Subskriptionsraums in Themen, dies ist generell nur entlang einer Dimension möglich. Da die Themen und ihre Unterteilung durch die Publisher bestimmt werden, hat eine Umstrukturierung der Themen immer auch weitreichende Folgen für die Subscriber.

Die Umsetzung der themenbasierten Filterung ist ebenfalls komplizierter als die kanalbasierte, insbesondere wenn Muliticast-Mechanismen genutzt werden. Vor allem die Implementierung von Sub-Themen ist nicht trivial. Möglichkeiten sind bspw., dass für jedes Thema und jeden Sub-Themenbaum eine eigene Multicastgruppe existiert, dass Notifikationen mehrfach veröffentlicht werden, dass sich Subscriber für Gruppen von Themen subskribieren oder dass Multicastgruppen für jedes Thema existieren. Diese Aufzählung ist nicht vollständig, weist aber auf die Vielfalt der Umsetzungsmöglichkeiten hin.

Typbasierte Filterung. Die *typbasierte Filterung* ähnelt in ihrem Grundprinzip der themenbasierten Filterung, betrachtet Notifikationen jedoch als Objekte, welche in einer Typhierarchie ihrer Klassen eingebettet sind [61]. Die Grundidee einer Hierarchie findet sich neben der themenbasierten Filterung auch in der typbasierten Filterung wieder. Typbasierte Filterung eröffnet gegenüber der themenbasierten Filterung die Möglichkeit der Mehrfachvererbung, was einerseits die Ausdrucksfähigkeit der Filterung verbessert, andererseits in der praktischen Umsetzung mit Problemen verbunden sein kann. In der typbasierten Filterung lässt sich bspw. „Temperatur“ als Untermenge von „Umgebungsbedingungen“ („EnvironmentalConditions.Temperatures“) zusammenfassen. Diese stellen dann gleichzeitig die Teilmenge einer „Endogenen Zustandsbeschreibung“ („EndogenStateDescr.Temperatures“), z.B. eines Kernreaktors, dar.

Inhaltsbasierte Filterung. *Inhaltsbasierte Filterung* erlaubt die Definition von Filtern auf dem gesamten Inhalt von Ereignissen, unabhängig von einer durch

den Publisher vorgenommenen Einstufung in Typen, Themen oder Kanäle [153]. Sie ist die ausdrucksstärkste Art der Filterung. Filter sind im Zusammenhang mit der inhaltsbasierten Filterung als Funktion zu verstehen, welche Notifikationen hinsichtlich von Bedingungen auswertet und im Ergebnis einen Boolschen Wert zurückgibt. Die jeweiligen Filter sind Kombinationen einzelner Teilfilter (auch Filterprädikate genannt), welche logisch miteinander verknüpft den Funktionswert des Filters ergeben. Die Umsetzung der inhaltsbasierten Filterung ist im Vergleich zu den bisher genannten Filtertechniken am komplexesten. So ist eine Umsetzung in IP-Multicast nicht ohne weiteres möglich, wächst die Anzahl der Multicastgruppen doch exponentiell zur Anzahl der möglichen Filter. Inhaltsbasierte Filterung findet sich auch im produktiven Einsatz wieder, bspw. im *Corba Notification Service* [44] und im *Java Message Service* (JMS) [144].

Allerdings ist die Wirksamkeit der inhaltsbasierten Filterung prinzipiell dadurch eingeschränkt, dass Publisher und Subscriber sich sowohl über das Daten-, als auch über das Filtermodell geeinigt haben [153] müssen. Mühl et al. [153] nennen eine Bandbreite von möglichen Ausgestaltungen der inhaltsbasierten Filterung, dazu zählen das Matching von Templates [47], einfache Vergleiche [31], erweiterbare Filterausdrücke [150], Name/Wert-Paare, Ausdrücke in XPath für XML [5] sowie mobiler Programmcode [54].

2.2.3 Routing von Notifikationen und Subskriptionen

Die Weiterleitung der Subskriptionen und Notifikationen obliegt, als Mittler zwischen Publishern und Subscribern, dem Notifikationsdienst. In verteilten Umgebungen wird dieser von Teilinstanzen des Notifikationsdienstes, den Brokern, umgesetzt und auf den am Kommunikationsnetz beteiligten physikalischen Geräten des Geräteensembles ausgeführt. Die Subscriber veröffentlichen ihr Interesse an bestimmten Typen von Notifikationen über den Umweg des nächstgelegenen Brokers des Notifikationsdienstes. Auch die Publisher veröffentlichen die von ihnen detektierten Ereignisse als Notifikationen. Die Broker agieren in beiden Fällen als Stellvertreter ihrer *Klienten*. Grundsätzlich erfolgt die Weiterleitung von Notifikationen anhand der in den Routingtabellen festgehaltenen Subskriptionen. Die Entscheidung darüber, an welcher Stelle im Brokernetzwerk sich Notifikationen und die in den Subskriptionen veröffentlichten Filter treffen, bestimmt der eingesetzte *Routingalgorithmus*. Da sich der eingesetzte Routingalgorithmus auf inhaltsbasierte Filterung stützt, lässt sich hier auch von inhaltsbasiertem Routing sprechen.

Die für die Weiterleitung von Notifikationen und Subskriptionen genutzten Routingalgorithmen sind vielfältig. Ihr Einsatz erfolgt in Abhängigkeit vom Einsatzzweck und der Geräteinfrastruktur. So führt eine möglichst genaue Beschreibung von Interessen zu einer Menge von Notifikationen, die den gewünschten Notifikationen entspricht. Damit geht ein erhöhter Aufwand für die Filterung nicht benötigter Notifikationen am Broker einher, die Weiterleitung von unerwünschten Notifikationen wird jedoch unterbunden. Hier kann Kommunika-

tionsbandbreite eingespart werden. Um jedoch aus Sicht der Broker Filteraufwand und damit die Auslastung von Rechenkapazität zu vermeiden, sind weniger genau beschriebene Filter gewünscht. Algorithmen zur Filterung von Notifikationen sind also immer ein Trade-Off zwischen Filteraufwand und Weiterleitungsaufwand. Routingalgorithmen lassen sich folgendermaßen unterscheiden [152]:

Flooding. Beim *Flooding* (Fluten) werden Notifikationen kontrolliert an alle Broker im Brokernetzwerk gesendet. Subskriptionen sind nicht notwendig, da keine Filterung innerhalb des Notifikationsdienstes stattfindet. Jeder Broker muss alle eingehenden Notifikationen weiterleiten, die jeweiligen Routingtabellen umfassen lediglich die Subskriptionen der lokalen Klienten, welche mit den eingehenden Notifikationen verglichen und bei zutreffenden Filterbedingungen an die Klienten weitergeleitet werden.

Falls alle Klienten im Netzwerk an allen publizierten Notifikationen interessiert sind, ist das Flooding vorteilhaft, da keine, in diesem Fall nicht notwendigen Routingtabellen für Subskriptionen vorgehalten und Notifikationen nicht gegen Filterausdrücke ausgewertet werden müssen. Dies ist jedoch ein Sonderfall. Sind die Konsumenten hingegen jeweils nur an einer Teilmenge der Notifikationen interessiert, führt die Weiterleitung von nicht benötigten Ereignissen zu einer übermäßigen, vergeudeten Nutzung von Kommunikationsressourcen und unnötigen Vergleichsoperationen.

Simple Routing. Beim *Simple Routing* wird, im Gegensatz zum Flooding, das Brokernetz nicht mit Ereignissen, sondern mit Subskriptionen (und dem Widerruf von Subskriptionen) geflutet. Die Broker verfügen bei dem Simple Routing-Ansatz über globales Wissen über alle im Netzwerk vorhandenen Interessen, ausgedrückt in Subskriptionen. Jeder Broker führt eine Routingtabelle mit den Subskriptionen seiner Klienten, sowie eine Tabelle mit allen anderen im Netzwerk vorhandenen Subskriptionen, zusammen mit den Angaben über ihre Herkunft (d.h. unter Angabe des Nachbarbrokers aus dessen Richtung sie stammen). Die eingehenden Notifikationen werden mit den vorhandenen Subskriptionen aus der Routingtabelle verglichen und bei entsprechender Überdeckung von Subskription und Notifikation in Richtung der Filterherkunft weitergeleitet. Einzige Voraussetzung für das Simple Routing ist die eindeutige Identifizierbarkeit der aus den Subskriptionen abgeleiteten Filter.

Da Filter zwar eindeutig sind, jedoch nicht zusammengefasst werden, wächst die Größe der Routingtabellen linear mit der Anzahl der aktiven Subskriptionen und der Anzahl der Broker. Zudem muss jede Routingtabelle nach einer Änderung der Subskriptionsmenge an Brokern aktualisiert werden, Simple Routing skaliert im Gegensatz dazu nicht.

Identity-based Routing. Der Grundgedanke bei der Entwicklung des *identitätsbasierten Routings*, wie auch des Covering- und Merging-based Routings,

besteht darin, Gemeinsamkeiten zwischen den formulierten Ereignisfiltern zusammenzufassen und für die Einsparung von Einträgen in den jeweiligen Routingtabellen auszunutzen und neben der Einsparung von Routingtabelleneinträgen auch die Anzahl von Vergleichen zwischen Notifikationen und Filtern zu reduzieren. Welches Routingverfahren dabei am vorteilhaftesten ist, kann nicht pauschal gesagt werden. Dies hängt immer von der zugrunde liegenden Infrastruktur, der Anzahl von Publishern und Subscribern, der Anzahl unterschiedlicher Notifikationen und der Selektivität der Filterausdrücke ab.

Beim identitätsbasierten Routing werden identische Filter nicht weitergeleitet. Das bedeutet, sobald ein neuer Filter bei einem Broker von einem seiner Nachbarbroker eintrifft, wird dieser nur weitergeleitet, falls nicht schon ein identischer Filter existiert. Auch der Widerruf einer bestehenden Subskription wird nur dann weitergeleitet, falls kein identischer Filter aus einer anderen Richtung existiert.

Covering-based Routing. Das *Covering-based Routing* ist als eine Weiterentwicklung des identitätsbasierten Routings anzusehen. Es vermeidet die Weiterleitung von Filtern, wenn diese eine Teilmenge eines bestehenden Filters darstellen. Um festzustellen, ob ein Filter Teilmenge eines anderen Filters ist, werden Überdeckungstests (engl. *covering tests*) genutzt. Die jeweiligen Routingtabellen enthalten dabei alle nichtüberdeckten Filter in Richtung der Nachbarbroker. Die Weiterleitung der Subskriptionen erfolgt wie beim identitätsbasierten Routing schrittweise mit der Überprüfung auf Überdeckung an jedem Broker. Die Weiterleitung von widerrufenen Subskriptionen ist hier allerdings komplizierter, müssen doch beim Widerruf einer überdeckenden Subskription die vormals überdeckten Subskriptionen wiederhergestellt und weitergeleitet werden.

Merging-based Routing. Wie auch bei beiden vorangegangenen Routingverfahren ist es das Ziel des *Merging-basierten Routings*, Routingeinträge und Filteroperationen einzusparen. So ermöglicht das Merging-basierte Routing die Zusammenfassung von Filtern zu einem gemeinsamen, überdeckenden Filter. Erreicht also ein Filter einen Broker, versucht dieser mit bereits vorhandenen Filtern eine Obermenge zu bilden. Dabei wird zwischen perfektem und imperfektem Merging unterschieden. Während perfektes Merging die Filter ohne überstehende Elemente zusammenfügt, wird beim imperfekten Merging in Kauf genommen, dass Notifikationen weder von einer, noch von der anderen Ausgangssubskription benötigt werden. Der Ressourcenverbrauch wird durch deren Weiterleitung dabei für die Einsparung von Routingeinträgen und weniger Vergleichen in Kauf genommen. Die Vorteile, die Merging-basiertes Routing in Bezug auf die Einsparung von Subskriptionen bietet, können sich jedoch unter Umständen umkehren. So ist es möglich, dass beim imperfekten Merging mehr überlappende als zusammengefasste Subskriptionen entstehen und sich der Weiterleitungsaufwand der Filter erhöht und das Routing im schlimmsten Fall zum Covering-basierten Routing entartet. Kommt es dagegen beim imperfekten Merging zu einer Vielzahl von Ausnahmen, mutiert das Routing in Richtung des Floodings [153]. Wie auch

beim überdeckungsbasierten Routing ist der Widerruf von Subskriptionen nicht trivial, müssen doch die noch aktiven Subskriptionen neu publiziert werden.

Erweiterungen vorhandener Routingverfahren

Abseits der vorgestellten Routingverfahren existieren eine Reihe von Erweiterungen, die für einige besondere Netzwerkkonfigurationen weitere Vorteile in puncto Effizienz bewirken. Davon werden im Folgenden die wichtigsten drei Erweiterungen [153] vorgestellt.

Advertisements. Die bisher vorgestellten Routingalgorithmen leiten (bis auf das Flooding) die veröffentlichten Subskriptionen an alle Broker des Netzwerks weiter, egal ob aus der Umgebung des Brokers bzw. des dortigen Teilnetzes Notifikationen zu erwarten sind. Ziel von *Ankündigungen* (engl. *advertisements*) ist es, Subskriptionen nur in die Bereiche des Netzwerks zu leiten, aus denen auch mit passenden Notifikationen zu rechnen ist. Ankündigungen werden von Publishern im Vorfeld, also vor der ersten Publikation von Notifikationen, veröffentlicht und beschreiben die zukünftig zu erwartenden Notifikationen. Die Broker verwalten in einer separaten Routingtabelle die Ankündigungen und leiten anhand dieser Einträge die eingehenden Subskriptionen weiter, wenn Ankündigungen und Subskriptionen sich überdecken. Dieses Verfahren verläuft analog zur Weiterleitung von Notifikationen bei entsprechend vorliegenden Subskriptionen. Die Einsparungen durch weniger Subskriptionseinträge in den Routingtabellen sowie weniger Vergleiche von Subskriptionen und Notifikationen werden durch das Führen einer zweiten Routingtabelle für die Ankündigungen erkauft, wodurch Ankündigungen als ein zusätzliches Optimierungswerkzeug in Netzwerken mit einer Vielzahl unterschiedlicher Typen von Subskriptionen und Notifikationen verbunden sind. Beim Einsatz von Ankündigungen ist außerdem zu beachten, dass Ankündigungen und Subskriptionen nicht zwangsläufig in einer bestimmten Reihenfolge vorliegen und daraus Race Conditions erwachsen können. Auch beim Widerruf von Subskriptionen und Ankündigungen ist auf einen korrekten Ablauf zu achten.

Hierarchische Routingalgorithmen. Die in Abschnitt 2.2.3 vorgestellten Algorithmen zum Routing von Subskriptionen und Notifikationen basieren auf einem *Peer-to-Peer* Netzwerk mit gleichberechtigten Brokern, die gegenseitig Filter austauschen. Im Gegensatz dazu sind beim hierarchischen Routing die Broker nicht gleichberechtigt, sondern hierarchisch angeordnet. Dabei existiert eine Rangordnung, an deren Spitze der Wurzelbroker (engl. *root*) steht. Beim hierarchischen Routing werden eingehende Notifikationen schrittweise in Richtung des Wurzelbrokers weitergeleitet und auf dem Weg dorthin gegen abgegebene Subskriptionen überprüft. Damit werden Subskriptionen und deren Widerrufe nur in Richtung des Wurzelbrokers geleitet. Im Vergleich zu Peer-to-Peer-Netzen

ermöglichen hierarchische Routingansätze durch die Weiterleitung der Notifikationen und Subskriptionen in Richtung des Wurzelbrokers eine Verkleinerung der Routingtabellen, da in den individuellen Routingtabellen nur die Filter des jeweiligen Teilbaums verwaltet werden. Allerdings verringert sich der Vorteil kleinerer Tabellen bei zunehmender Anzahl unterschiedlicher Subskriptionen.

Hierarchische und Peer-to-Peer-basierte Routingmechanismen lassen sich kombinieren. Dabei werden in der Regel einzelne Bereiche des Netzwerks in einer Baumstruktur organisiert und innerhalb dieser Bäume hierarchische Routingalgorithmen angewandt, während die Bäume (bzw. ihre Wurzelknoten) in einem Peer-to-Peer-Netzwerk miteinander verbunden sind. Diese Zusammenarbeit von hierarchischem und Peer-to-Peer-basiertem Routing wird auch als *hybrides Routing* bzw. als *hybrider Routingmechanismus* bezeichnet.

Rendezvous-basiertes Routing. Grundsätzlich werden in allen inhaltsbasierten Routingverfahren bestimmte Pfade für die Weiterleitung von Notifikationen von den Ereignisproduzenten zu den Subscribern aufgebaut. Bei den bisher genannten Routingverfahren wurden diese Pfade schrittweise anhand der abgegebenen Subskriptionen und der genutzten Routingalgorithmen im Netzwerk propagiert. Dabei werden die Subskriptionen im gesamten Netzwerk verbreitet. Beim *Rendezvous-basierten Routing* wird eine andere Arbeitsweise genutzt, denn es existiert ein Broker der als „Treffpunkt“ für alle Notifikationen und passenden Subskriptionen agiert. Zu diesem Treffpunktbroker (oder „Rendezvous-Broker“) werden also sowohl die Subskriptionen zu diesem Notifikationsstypen geleitet, als auch die passenden Notifikationen ausgehend vom Treffpunktbroker weitergeleitet. Die Subskriptionen hinterlassen in Richtung der Treffpunktbroker Routinginformationen in Richtung der jeweiligen Vorgänger. Dadurch werden Routingpfade etabliert, die für die spätere Auslieferung der Notifikationen zu den Subscribern genutzt werden. So existiert für jeden Notifikationstyp ein gemeinsam genutzter Verteilungspfad. Durch diesen wird sichergestellt, dass jede Notifikation an interessierte Subscriber ausgeliefert wird.

Damit sich Subskriptionen und Notifikationen jedoch überhaupt treffen können, muss die Adresse des Treffpunktbrokers innerhalb des Brokernetzwerks global bekanntgegeben werden. Mögliche Flaschenhälse sind zu vermeiden, aber gleichzeitig soll die Verteilung der Treffpunktbroker (bspw. mit der Anzahl der Notifikationstypen) skalieren. Solch eine Verteilung lässt sich mit Hilfe einer verteilten Hash-Tabelle (engl. *distributed hash table*, *DHT* [179]) realisieren. Mit deren Hilfe lässt sich aus dem eindeutigen Namen eines Notifikationstyps ein Schlüssel für die Adresse des Treffpunktbrokers bestimmen.

Prinzipiell müssen alle Notifikationen zunächst zum Treffpunktbroker geleitet werden. Werden jedoch auf dem Pfad dorthin alle Broker betreten, an die sämtliche Subskriptionen verwaltet und weitergeleitet werden, so wird die Notifikation nicht weitergeleitet und verworfen, ein Umstand der zu Überlastungen der Treffpunktbroker beitragen kann [153]. Eine Lösung dieses Problems kann durch zusätzliche Ankündigungen und damit weiteren Routinginformationen in

den Brokern erreicht werden, sodass Notifikationen nicht mehr zwangsläufig den Treffpunktbroker erreichen müssen [153].

2.2.4 Publish/Subscribe Middleware

Middleware beschreibt ein Konzept, welches als zusätzliche Abstraktionsebene als Verbindung zwischen Geräten in einer verteilten, heterogenen Umgebung fungiert. Diese Ebene ist eine zusätzliche Softwareschicht, welche zwischen der Betriebssystemebene und der verteilten Anwendung agiert und somit einheitliche, homogene Schnittstellen in Richtung der Anwendungsebene präsentiert, gleichzeitig die vorhandene Komplexität der unterhalb der Middleware liegenden Schichten den oberen Schichten gegenüber verbirgt. Das Konzept der Middleware ist ein verbreitetes Mittel zur Abstraktion von Kommunikation und wird oft und erfolgreich in komplexen, verteilten Systemen eingesetzt [153].

Der Fokus der Middlewarekonzepte hat sich im Lauf der Jahre allerdings verschoben. So gingen Middlewarekonzepte zunächst von Daten und Diensten an festen Standorten aus, bspw. als Sammlung von Objekten oder Daten innerhalb einer Datenbank. Feste Standorte von Daten bzw. Kollektionen ermöglichen eine Kommunikation mit festgelegten Anfrage- und Antwortstrukturen. Daraus entwickelten sich Middlewarearchitekturen mit klarer Delegation von Funktionalitäten. Diese werden von Geräten und Anwendungen gezielt aufgerufen und in die eigene Funktionalität eingebunden. Grundsätzlich wird bei dieser Art der Interaktion von Request/Reply-basierter Interaktion gesprochen, in Bezug auf die sich herausgebildeten Middlewarearchitekturen von Client/Server-Middlewarearchitekturen. Beispiele für solche Architekturen finden sich in *Remote Procedure Calls* (*RPC*) [18, 158, 159], aber auch bei Web-Services mit dem *Simple Object Access Protocol* (*SOAP* [25]) [153].

Auch wenn die bei den Client/Server-Middlewarearchitekturen eingesetzten Mechanismen bekannt sind und sich bewährt haben, stoßen sie bei zunehmender Dynamik in den Systemen an ihre Grenzen. So führt die direkte und oftmals synchrone Kommunikation zwischen aufrufenden und aufgerufenen Geräten zu einer engen Kopplung, was dem Kommunikationsmodell bei hinzukommenden oder ausscheidenden Geräten in dynamischen Umgebungen nicht entgegenkommt. Zudem müssen Vorschriften für die Dienstgüte von bereitgestellten Diensten an die Zulieferdienste entsprechend weitergeleitet werden. Werden dabei Dienste und Daten abgefragt, kann dies bspw. in zu hoher oder niedriger Frequenz durchgeführt werden. Führt eine zu hohe Frequenz zur Verschwendung von Ressourcen, ist bei einer zu niedrigen Frequenz mit einer wachsenden Latenz zu rechnen, außerdem sind aufrufende Geräte von der Dienstebereitstellung der aufgerufenen Geräte und somit von deren Existenz abhängig. Zu den genannten [153] nachteiligen Aspekten von Client/Server-Middlewarearchitekturen kommt ein dort ebenfalls genannter Nachteil hinzu, dies ist die Vermischung von Kontrollflüssen mit deren Steuerung und der eigentlichen Anwendungslogik.

Die Notwendigkeit, asynchrone und entkoppelte Interaktion zwischen Geräten zu ermöglichen, führte bspw. zur Erweiterung bestehender Middlewarearchitekturen wie *Corba* oder *J2EE*, welche auch in den Teilabschnitten des folgenden Abschnitts 2.2.5 genauer erklärt werden.

2.2.5 Klassische Middlewarearchitekturen

Die Notwendigkeit, heterogene Geräte miteinander durch eine Middleware zu verbinden und miteinander kommunizieren zu lassen, gilt durch ihre inhärente Dynamik und Heterogenität in besonderem Maße für ubiquitäre Umgebungen. So haben sich in der Vergangenheit vielfältige Middlewarekonzepte herausgebildet, wovon einige im Folgenden vorgestellt werden. Dabei liegt der Fokus zunächst auf Architekturen, deren Entwicklung durch die Industrie vorangetrieben wurden und eine weite Verbreitung gefunden haben. Die Betrachtung von Entwicklungen aus dem Forschungsbereich insbesondere für ubiquitäre Systeme findet sich dagegen anschließend in Abschnitt 2.2.6.

Corba Event- und Notification Service. Unter der Maßgabe eine Middleware für die Interoperabilität von Geräten und Anwendungen zu entwickeln und bereitzustellen, entstand *Common Object Request Broker Architecture* (*Corba*). *Corba* ist in seinem Grundaufbau eine verteilte, objektorientierte Middleware auf Basis einer Client/Server-Architektur, die durch die sog. *Object Management Group (OMG)* entwickelt und standardisiert wurde. Der Standard ist plattform- und sprachunabhängig formuliert und umfasst Protokolle und Dienste, die konkrete Implementierung ist jedoch den Entwicklern einzelner Systeme überlassen.

Grundbestandteil von Corba ist der sogenannte *Object Request Broker (ORB)*, welcher als Vermittler und ausführende Einheit entfernter Methodenaufrufe in Form eines Dienstes agiert. So erfolgt der Zugriff auf entfernte Objekte nicht direkt über die bereitgestellten Schnittstellen, sondern durch den Umweg über den ORB mit Hilfe von Proxys. Die Proxys (bzw. Stellvertreterobjekte) kapseln die (Request/Reply-) Kommunikationsvorgänge und bieten nach außen die gleichen Schnittstellen wie das stellvertretene Objekt. Aufrufer und Aufgerufene müssen daher nicht selbst das Kommunikationsmanagement übernehmen.

Wie in vielen Bereichen der Informatik üblich, haben einmal eingesetzte Techniken oftmals eine lange Lebensdauer, der Einsatz relationaler Datenbanksysteme in Unternehmen ist ein klassisches Beispiel hierfür. Corba fand und findet eine weite Verbreitung als klassische Request/Reply-basierte Middleware, besonders in Bereichen wie Telekommunikation und Finanzanwendungen. Trotzdem hat sie nach Ansicht einiger Autoren ihren Höhepunkt überschritten, wie bereits vor fast einem Jahrzehnt angemerkt wurde [94]. Trotz ihrer Nachteile ist Corba durch ihre Sprachen- und Plattformunabhängigkeit, ihre bereits enthaltenen Standardfunktionen zur Kommunikation und ihre Flexibilität noch immer weit verbreitet und dient zudem als Basis für Erweiterungen und aufbauende Entwicklungen.

Eine Erweiterung von Corba ist der *Corba Event Service* [43], welcher asynchrone Kommunikation bei loser Kopplung unterstützt und damit den Schwächen von Corba in heterogenen Umgebungen entgegenwirkt. Beim Corba Event Service existieren für jedes Objekt zwei Rollen, nämlich (Ereignis-) Erzeuger und (Ereignis-) Verbraucher, auch als Supplier und Consumer bezeichnet. Zwischen Erzeuger und Verbraucher agiert als Vermittler ein Ereigniskanal, auf den die Supplier und Consumer mittels synchroner Methoden zugreifen. Der Ereigniskanal selbst unterstützt unterschiedliche Kommunikationsmethoden und Datenmodelle und erweitert dabei die ursprüngliche Corba-Spezifikation. Grundsätzlich werden durch den Corba Event Service zwei unterschiedliche Kommunikationsmethoden unterstützt, nämlich sowohl das *Push-* als auch das *Pull-*Modell. Beim Push-Modell startet der Supplier die Kommunikation durch Abgabe eines Ereignisses in den Kommunikationskanal, während bei dem Pull-Modell der Verbraucher die Kommunikation durch Anforderung an den Erzeuger initiiert. Der Ereigniskanal als zentrales Objekt verhält sich wie ein Proxy-Objekt mit Schnittstellen zur Entkopplung von Erzeuger und Verbraucher.

Der Ereigniskanal erlaubt dem Erzeuger die Weiterleitung von Ereignissen an mehrere Verbraucher, ohne dass der Erzeuger diese kennen muss. Darüber hinaus können Push- und Pull-Modell miteinander kombiniert werden, sodass bspw. Verbraucher über ihr spezifiziertes Pull-Modell mit Ereignissen versorgt werden, obwohl die Ereignisse durch den Erzeuger über Push eingespeist wurden.

Prinzipiell werden durch Corba einfache, nicht-typisierte sowie typisierte Ereignisse unterstützt. Nicht-typisierte Ereignisse werden grundsätzlich immer weitergeleitet, wobei jedoch Typinformationen verloren gehen. Typisierte Ereignisse dagegen erfordern vor der Weiterleitung die Abstimmung von Erzeugern und Verbrauchern bezüglich der Ereignisschnittstellen und der Weiterleitungsmethode. Trotz der Erweiterung Corbas durch den Event Service sind einige bestehende Probleme noch immer ungelöst. So können zwar Ereignisse lose gekoppelt ausgetauscht werden, allerdings erhalten Verbraucher sämtliche Ereignisse eines Ereigniskanals sobald sie sich beim Kanal angemeldet haben, selbst wenn ihr eigentliches Interesse sich nur auf einen Teilbereich der im Kanal weitergeleiteten Ereignisse bezieht. Da Ereignisse nicht gefiltert werden können, müssen sie zwangsläufig weitergeleitet werden, was zu einer unnötigen Belastung der Kommunikationskanäle führen kann. Des Weiteren unterstützt der Corba Event Service keine Maßnahmen zur Sicherung der Dienstgüte, sondern behandelt alle Ereignisse mit gleicher Priorität.

Die angeführten Probleme wurden auch durch die OMG erkannt, welche ihrerseits mit der Einführung des *Corba Notification Service* [44] reagierte. Durch die Erweiterung der Ereignisse (und Ereigniskanäle) können nun Ereignisse gefiltert werden, dazu werden die bisher verwendeten primitiven Ereigniskanäle erweitert. Die ebenfalls erweiterten, nun strukturierten Ereignisse bestehen aus zwei Teilen, einem Header mit Meta-Informationen über Ereignistyp, Anwendungsdomäne und Ereignisnamen, sowie einem Body, der die eigentlichen Nutzinformationen beinhaltet. Sowohl Header als auch Body können im Prinzip un-

begrenzt viele Name/Wert-Paare enthalten, die als Basis für Filterungen von Ereignissen dienen können. Als weiteren Aspekt bietet der Corba Notification Service die Möglichkeit, die Dienstgüte der Ereignisauslieferung anhand unterschiedlicher Parameter zu beeinflussen, womit auch dem zweiten kritischen Aspekt des Corba Event Service entgegengetreten wird. Schließlich sind Verbraucher nun in der Lage, mit Hilfe einer Sprache eigenständig Filterausdrücke auf den Name/Wert-Paaren zu formulieren, wobei diese sowohl auf den Inhalt, als auch auf Dienstgüteinformationen hin angewendet werden.

Daneben bietet der Corba Notification Service die Möglichkeit, ein Repository für die im System veröffentlichten Ereignisse anzulegen. In diesem Verzeichnis werden die Metadaten der Ereignisse abgelegt, sodass Verbraucher sich einen Überblick über die vorhandenen Ereignisse machen und entsprechend genaue Anforderungen erstellen können.

Java Message Service. Neben Corba wurde es in der Vergangenheit auch nötig, die *Java Platform Enterprise Edition* (*J2EE*)-Spezifikation um asynchrone Aufrufe und Notifikationsdienste zu erweitern, woraus der *Java Messaging Service (JMS)* [113] entstand. Diese Spezifikation beschreibt diejenigen Schnittstellen (*Application Program Interface (API)*) und Protokolle, mit denen Anwendungen bzw. Anwendungskomponenten asynchron Nachrichten senden, empfangen und verarbeiten. Die konkrete Implementierung obliegt, wie schon bei Corba, den jeweiligen Herstellern, welche als *JMS Provider* bezeichnet werden.

JMS unterstützt grundsätzlich zwei unterschiedliche Kommunikationsarten, dies sind Punkt-zu-Punkt Kommunikation und Publish/Subscribe. Bei der Punkt-zu-Punkt Kommunikation interagieren Sender und Empfänger mit Hilfe von Nachrichtenschlangen, welche durch einen JMS-Server betrieben und verwaltet werden. Eine direkte Kommunikation der Partner ist nicht vorgesehen, die Kommunikation ist in dieser Hinsicht entkoppelt. Bei der zweiten Umsetzungsmöglichkeit, dem Publish/Subscribe, erfolgt ein themenbasierter, entkoppelter Austausch von Nachrichten zwischen Sender und Empfänger über den JMS-Server. Dieser verwaltet mehrere, unterschiedliche Themen. Die Klienten können zu den jeweiligen Themen Nachrichten publizieren und sich für Themen subskribieren.

Auch die in JMS genutzten Nachrichten bestehen aus einem Header und einem Body. Der Header enthält alle vordefinierte Felder für bestimmte Informationen, wie bspw. eine eindeutige Identifizierung, das Ziel, die Prioritätsstufe oder einen Zeitstempel und kann darüber hinaus um weitere Felder individuell erweitert werden. Der Body ist wiederum Träger der Nutzdaten, dabei werden eine Reihe unterschiedlicher Datentypen und -formate unterstützt.

JMS verfügt bei der von ihr unterstützten themenbasierten Publish/Subscribe-basierten Kommunikation über die Möglichkeit, eine inhaltliche Filterung der Nachrichten in begrenztem Maße durchzuführen. Dies geschieht mit Hilfe eines *Message Selectors*, der die Spezifikation von Interessen in Form eines Filters auf dem Nachrichtenheader erlaubt, wobei Filterausdrücke auf einer in *SQL92* [50]

beschriebenen Syntax beruhen. Der Einsatz von Filtern auf dem Body ist nicht möglich, jedoch können Filterausdrücke auf selbstdefinierten Header-Feldern angewandt werden.

Die Nachrichten werden in JMS durch die Subscriber synchron und asynchron konsumiert und werden mit den von Corba bekannten Push- und Pull-Methoden vom Erzeuger zum gewünschten Verbraucher verschickt. Auch bei der Bestätigung über die Zustellung von Nachrichten verfügt JMS über unterschiedliche Möglichkeiten, nämlich über automatisierte oder klientenspezifische Bestätigungen. Die Nachrichten können volatil oder persistent sein. Persistente Nachrichten werden (genau) einmal zum Konsumenten zugestellt und beim JMS-Dienstanbieter solange zwischengespeichert, bis der Empfänger sie abholt. Volatile Nachrichten werden hingegen verworfen, falls kein Empfänger verfügbar ist.

Ähnlich wie bei den volatilen und persistenten Nachrichten, erlaubt JMS den Klienten ihr Interesse in Form von dauerhaften und nicht-dauerhaften Subskriptionen zu publizieren. Die nicht-dauerhaften Subskriptionen sind an die Lebenszeit der Subskribenten gebunden, verfallen also beim Ausfall/Ausscheiden der Subskribenten aus dem Netzwerk. Die dauerhaften Subskriptionen werden dagegen auf einem Server gespeichert und bleiben aktiv, selbst bei inaktiven Subskribenten. Die zwischenzeitlich aufgelaufenen Nachrichten werden dann solange gespeichert, bis der Subscriber wieder mit dem Netzwerk verbunden ist bzw. die Lebenszeit der Nachrichten noch nicht abgelaufen ist.

Eine weitere Besonderheit von JMS ist die Umsetzung des Transaktionskonzepts, damit ermöglicht JMS die Zusammenfassung von Veröffentlichungen und Konsumptionen verschiedener Nachrichten zu einer logischen Einheit, welche entweder in Gänze oder gar nicht abgearbeitet werden.

Wie auch Corba wird JMS produktiv in der Industrie eingesetzt. Ein häufiger Einsatzzweck von JMS ist die Integration unterschiedlicher Komponenten und vorhandener Services zu einer gemeinsamen, verlässlichen, ereignisgetriebenen Infrastruktur. So ermöglicht JMS eine gemeinsame Schnittstelle für Java-Anwendungen mit anderen produktiven Systemen unterschiedlicher Hersteller. Allerdings leiden die in JMS beschriebenen Spezifikationen unter fehlender technischer Genauigkeit und ungenauen Implementierungsvorschriften.

2.2.6 Verwandte Arbeiten

Wie im vorherigen Abschnitt 2.2.5 beschrieben wurde, haben sich im Bereich der Middleware einige Standards in der Industrie herausgebildet. Im Umfeld ubiquitärer Anwendungen in heterogenen Umgebungen gibt es derzeit keinen Middlewarestandard, sondern vielmehr eine Reihe von Forschungsprototypen. Einige dieser Prototypen werden im Folgenden allgemein und im Hinblick auf ihre Eignung zur Unterstützung ubiquitärer Anwendungen in heterogenen Umgebungen vorgestellt.

Rebeca. Der Forschungsprototyp, auf dem diese Arbeit aufbaut, ist die *Rebeca Event-Based Electronic Commerce Architecture* (REBECA)-Middleware. Sie wurde ursprünglich für den Einsatz in verteilten, elektronischen Geschäftsanwendungen entwickelt, wie sie beim elektronischen Handel mit Wertpapieren oder bei der elektronischen Informationsverteilung vorkommen [23]. REBECA als ein verteilter Notifikationsdienst implementiert als Middleware einen ereignisbasierten Kommunikationsdienst mittels Publish/Subscribe, womit sie in ihrer Funktionalität mit den weiteren Middlewareentwicklungen dieses Abschnitts 2.2.6 vergleichbar ist, sich jedoch in einigen Aspekten von den anderen vorgestellten Entwicklungen unterscheidet. So basiert REBECA auf (1) einer *formalen Spezifikation* [170], welche das Verhalten der Notifikationen eindeutig beschreibt; (2) einem *erweiterbaren Daten- und Notifikationsmodell*, das die Erweiterung des vorgegebenen Datenmodells (bestehend aus Name/Wert-Paaren) um weitere Datenformate und Definitionen von Randbedingungen zulässt; (3) einer *Erweiterungsfähigkeit des Routingframeworks* um hierarchische und Peer-to-Peer-basierte Routingalgorithmen [149, 150] und der Möglichkeit, Ankündigungen zuzulassen; sowie (4) der Möglichkeit zur Nutzung von Scopes zur Beschränkung der Sichtbarkeit von Notifikationen [65, 68, 153, 169].

Die Architektur von REBECA besteht aus Klienten und dem Notifikationsdienst, wobei Klienten als Produzenten und Konsumenten von Notifikationen agieren können. Aufbauend auf ein physisches Netzwerk formen die Broker, die als Prozesse auf physikalischen Knoten ausgeführt werden, ein Overlay-Netzwerk in Form eines azyklischen Graphen, mit den Brokern des Notifikationsdienstes als Knoten des Graphen und logischen Verbindungen zwischen den Brokern als Kanten des Graphen, welche die Kommunikation zwischen den Brokern im Overlaynetz darstellen. Im physischen Netzwerk werden diese Verbindungen als *Transmission Control Protocol/Internet Protocol* (*TCP/IP*)-Verbindungen umgesetzt. Als Alternative zu TCP/IP-Verbindungen können jedoch auch Nachrichten als IP-Multicast genutzt werden. Azyklische Brokernetze können allerdings problematisch werden, da Überlastungen bestimmter Knoten oder Kommunikationsverbindungen zu einem Engpass im Gesamtsystem ausarten können. Gleiches gilt bei Ausfall von Brokern oder Kommunikationsverbindungen. Möglichkeiten, diesen Problemen mit Redundanz zu entgehen, sind bereits in REBECA integriert worden [46, 176, 200]. Die ursprüngliche Entwicklungsrichtung von REBECA als Plattform für die Kommunikation in verteilten Umgebungen hat sich im Laufe der Zeit dahingehend gewandelt, dass sie zunehmend als Grundlage für die Forschung an inhaltsbasierten Routingmechanismen genutzt wurde und wird. Als Forschungsschwerpunkt [66] wurde in REBECA die Integration unterschiedlicher Routingkonzepte und die Erweiterbarkeit des Daten- und Notifikationsmodells in den Fokus gerückt. Gerade im Zusammenhang mit dem spezifizierten und voraussehbaren Verhalten hat sich REBECA zu einer geschätzten prototypischen Umgebung entwickelt. So unterstützen Erweiterungen Konzepte für mobile Klienten und Anwendungen [155, 213], die Integration von historischen Daten und Puffern [40], Konzepte zur Sichtbarkeit und Strukturierung [64, 169], Konzepte für Programmierabstraktionen [201], Konzepte zur modellgetriebenen Entwick-

lung [168, 171], Konzepte zur Integration und Evaluation von Rekonfigurationsmechanismen [172], Konzepte zum adaptiven Routing [190, 192], Konzepte zur Umsetzung von Ideen zur Selbstorganisation und Selbststabilisierung in Brokernetzen [110, 111], sowie Methoden zur analytischen Beurteilung von Routingalgorithmen [154, 191].

REBECA unterstützt eine beliebige Menge von Ereignisattributen, die in Form von Name/Wert-Paaren gekapselt sind. Die Datentypen der Attributwerte sind vielfältig und reichen von primitiven Datentypen (Ganzzahlen, Fließkommazahlen, Zeichen, Zeichenketten, Boolsche Werte etc.) bis zu komplexen Datentypen. Darüber hinaus sind neben den vordefinierten auch eigene Datentypen definier- und integrierbar. Bedingung ist jedoch, dass die neu eingeführten Datentypen durch die Routingalgorithmen bezüglich der Filterung und Weiterleitungsoptimierungen unterstützt werden können. Die Filterausdrücke selbst sind beliebig miteinander kombinierbar, solange sie Boolsche Kombinationen unterschiedlicher Prädikate sind, wobei die einzelnen Prädikate Bedingungen auf einzelnen Ereignisattributen definieren. Die Verantwortung für die Erstellung von Ereignisattributen und Filtern liegt beim Nutzer und damit auch die Fähigkeit, die Filter durch die Routingalgorithmen so zu nutzen, dass Optimierungen bezüglich der weiterzuleitenden Notifikationen erreichbar werden.

In Bezug auf die eingesetzten Routingalgorithmen ist REBECA ebenfalls flexibel. So werden neben dem Fluten (*Flooding*) auch das *Simple Routing*, das Identitätsbasierte Routing, das Überdeckungsbasierte Routing sowie das *Merging*-basierte Routing unterstützt [151]. Die Flexibilität von REBECA geht an dieser Stelle sogar soweit, dass unterschiedliche Routingmechanismen jeweils für Subskriptionen und Ankündigungen gleichzeitig genutzt werden können, woraus sich bei der Vielzahl unterstützter Routingtechniken eine hohe Anzahl von Kombinationsmöglichkeiten ergibt. Anhand der Spezifikation von REBECA mittels sequentieller Pfade und linearer Temporallogik konnte im Übrigen nachgewiesen werden, dass die genutzten Routingalgorithmen formal korrekt sind [153].

Ein weiterer in REBECA unterstützter Aspekt ist die Kontrolle der Sichtbarkeit von Notifikationen. So kann eine Menge von Klienten definiert werden, für die Notifikationen sichtbar bzw. unsichtbar sind, d.h. an diese müssen bzw. dürfen die nicht sichtbaren Notifikationen nicht weitergeleitet werden. Ursprünglich wurden *Sichtbarkeitsbereiche* (*Scopes*) als Strukturierungsmittel zur Ordnung Publish/Subscribe-basierter Systeme und zur Organisation der ereignisgetriebenen Systeme in REBECA eingefügt [169]. Welche Ansätze für die Umsetzung von Scopes existieren, wird bspw. in [66] beschrieben. Die Umsetzung von Scopes in REBECA ist u.a. Thema in [169]. Mit Hilfe dieser Scopes lassen sich im Umfeld ubiquitärer Anwendungen bspw. unabhängig von der Infrastruktur Kommunikationsbereiche bestimmter Anwendungen abgrenzen, ohne dass sich diese mit ähnlichen Bereichen und Informationen überschneiden.

Ein anderer integrierter Aspekt ist die Umsetzung von Programmierabstraktionen, also insbesondere das Verbergen der Hardware- und Plattformdetails [201], aber auch der Einsatz von Programmierabstraktionen in ubiquitären Um-

gebungen. Diese erlauben durch Mittel der objektorientierten Programmierung die nahtlose Umsetzung typsicherer Filterausdrücke und Ereignisbehandlungsroutinen, wie bspw. die dynamische Generierung von Anwendungscode für die Definition komplexer Ereignismuster zur Laufzeit.

Ein weiteres Merkmal von REBECA ist die Unterstützung mobiler Klienten, was gerade ein wichtiges Kriterium für den Einsatz in dynamischen, heterogenen Umgebungen ist. Dabei wird Mobilität auf zwei Ebenen, nämlich auf Ebene der Geräte und auf der Anwendungsebene [212], unterstützt. Mobile Klienten sind i.d.R. drahtlos mit einem REBECA-Broker verbunden. Drahtlose Verbindungen sind jedoch bei weitem nicht so verlässlich wie drahtgebundene. Folglich können die Verbindungen häufiger ausfallen oder gar komplett abreißen. Die auf die von den Klienten abgegeben Subskriptionen passenden Notifikationen können dann nicht von den Brokern ausgeliefert werden und verfallen. Falls ein Klient nach einer gewissen Zeit jedoch wieder eine Verbindung zum Broker aufbauen kann, können die zwischenzeitlich verfallenen Notifikationen logische Inkonsistenzen beim Klienten nach sich ziehen. Um dieses Problem zu umgehen, ist REBECA in der Lage, Broker bei aktiven Subskriptionen die (vorübergehend) nicht zustellbaren Notifikationen zwischenzuspeichern zu lassen und bei vorliegender Verbindung auszuliefern. Dieser Service ist ähnlich der Eigenschaft von REBECA, Notifikationen in einer *History* abzulegen [40]. Durch die History können Klienten per Subskription auch auf vergangene Notifikationen zugreifen, sofern diese noch nicht abgelaufen sind. Die betreffenden Notifikationen werden dann erneut ausgeliefert. Darüber hinaus unterstützt REBECA auch Standortwechsel (*Roaming*) von Klienten. Bewegen sich Klienten, müssen neben den gecachten Notifikationen auch die Subskriptionen an den Broker weitergeleitet werden, mit dem der Klient nun verbunden ist. Auch dies unterstützt REBECA, ohne dass dabei die logische Ordnung von Notifikationen gestört wird [213]. Dies geschieht vor allem an Brokern, die in der Lage sind, die Standortveränderungen von Publishern und Subscribern zu erkennen, d.h. Subskriptionen und Notifikationen verlust- und überlappungsfrei im Netzwerk weiterzuleiten. Grundsätzlich vor allem für die verlässliche Kommunikation mittels latent unzuverlässiger Kommunikationsverbindungen gedacht, bietet die Unterstützung mobiler Klienten einen wichtigen, grundlegenden Aspekt für das in dieser Arbeit vorgestellte Verfahren, ermöglicht es doch grundsätzlich die Verschiebung von Anwendungskomponenten unter Beibehaltung der Reihenfolge der Subskriptionen.

Neben physischer Mobilität kann jedoch auch logische Mobilität gefragt sein, nämlich wenn Klienten relative Subskriptionen abgeben, sich also bspw. für Temperaturereignisse aus der Nachbarschaft interessieren. REBECA unterstützt auch die logische Mobilität durch das Konzept eines ortsabhängigen Filters [67], so wird je nach Standort des Klienten die Subskription entsprechend den äußeren Bedingungen angepasst. Möglich werden die ortsabhängigen Filter durch das Hinzufügen weiterer Name/Wert-Paare, welche den Standort der emittierten Notifikationen beschreiben und die Möglichkeit für die Subscriber bietet, lokationsabhängige Filter zu formulieren. Die Auswertung der Name/Wert-Paare am Broker erfolgt analog zu den bisherigen Publish/Subscribe-Schnittstellen,

ermöglicht aber die Auswertung von Filtern wie „in der Nähe von X“ oder „an meinem Standort“, wobei dies stark anwendungsabhängig implementiert ist. Die Möglichkeit, logische Mobilität zu integrieren, scheint auf den ersten Blick bei der bedarfsgerechten Anwendungsverschiebung nicht benötigt zu werden, wenn nach einer Verschiebung von Anwendungen bzw. Anwendungskomponenten das gleiche Ergebnis erreicht wird, als wenn dies ohne Änderung der Anwendungsplatzierung erzielt worden wäre. Wenn allerdings der Ereignisraum aufgespaltet wird, müssen Ereignisse überschneidungsfrei den jeweiligen Anwendungen zugeteilt werden, so dass an dieser Stelle die logische Mobilität in leicht abgewandelter Form zum Einsatz kommt.

Eine weitere wichtige Eigenschaft, gerade in einem dynamischen Netzwerk wie dem ubiquitären Computing, ist die Fähigkeit eines Systems, sich adaptiv und selbstorganisiert zu verhalten, sowie bis zu einem gewissen Grad fehlertolerant agieren zu können. Diese Entwicklungsrichtung wird auch in der Forschung rund um REBECA verfolgt. Grundsätzlich betreffen alle Veränderungen, die mit einem selbstorganisierten Netzwerk, dem adaptiven Verhalten und der Fehlertoleranz zusammenhängen, die Broker und die Verbindungen zwischen ihnen. Prinzipiell können einem dynamischen System Broker hinzugefügt, aus dem (Overlay-) Netzwerk entfernt und Verbindungen zwischen Brokern ausgetauscht werden. Da sich auf diese drei Operationen alle weitergehenden Aspekte zurückführen lassen [169], ist die Rekonfigurationsmöglichkeit eines Brokernetzwerks grundlegende Voraussetzung für Adaptivität, Selbstorganisation und Fehlertoleranz [172]. REBECA integriert Methoden zur Ausführung dieser drei Basisoperationen, unter der Sicherstellung, dass weder die Ordnung von Notifikationen noch die zugesicherte Dienstgüte des Netzwerks beeinträchtigt werden. Durch diese Zusicherungen bietet REBECA die Möglichkeit, sich selbstorganisiert an sich ändernde Umgebungsbedingungen und Ereignismuster anpassen zu können [111]. Ein Beispiel für selbstorganisiertes, adaptives Verhalten ist die Fähigkeit von REBECA, eigenständig zur effizienteren Weiterleitung von Notifikationen beizutragen. Dies geschieht durch die Minimierung von Weiterleitungskosten, die durch kontinuierliche Auswahl günstigerer und zuverlässigerer Verbindungen zwischen Publisher und Subscriber erfolgt. Eine weitere Besonderheit von REBECA ist die Möglichkeit, verschiedene Routingkonfigurationen in unterschiedlichen Bereichen gleichzeitig betreiben zu können. Dieses auch als *hybrides Routing* [190, 192] bezeichnete Konzept bietet den Brokern die Freiheit, den genutzten Routingalgorithmus selbst zu bestimmen und den momentanen Bedürfnissen im betreffenden Teil des Netzwerks anzupassen. Im gemeinsamen Zusammenspiel von angepasstem Routing und angepasster Netzwerktopologie können in der Gesamtbetrachtung Weiterleitungskosten in erheblichem Maße eingespart werden.

Grundsätzlich können Komponenten in dynamischen Umgebungen (temporär) ausfallen. REBECA bietet die Möglichkeit, die Lebensdauer von Subskriptionen zu begrenzen, sodass Klienten ihre Subskriptionen laufend erneuern müssen. So bleibt die Menge der im System vorhandenen Subskriptionen relativ aktuell, ohne dass spezielle Mechanismen benötigt werden, welche die Abwesenheit von Klienten erkennen und deren Subskriptionen zurückrufen, das gleiche gilt

auch für ausgetretene/ausgefallene Broker. Außerdem wird so die Lebensdauer von möglicherweise fehlerbehafteten Ankündigungen und Subskriptionen begrenzt [110]. Dieses Verfahren wird auch als „Soft-State“ bezeichnet. Die zweite Möglichkeit zur Steigerung der Fehlertoleranz besteht darin, dass im Falle eines Brokerausfalls Notifikationen nicht verloren gehen. So speichert ein Broker die von ihm ausgehenden Notifikationen solange, bis er eine Rückmeldung über die erfolgreiche Ankunft der Notifikation bei seinen direkten Nachbarn und deren Nachbarn erhält.

Siena. Eine der ersten prototypischen Implementierungen einer ereignis- und inhaltsbasierten Publish/Subscribe-Middleware ist *Scalable Internet Event Notification Architecture (Siena)* [30, 31]. Hauptaugenmerk bei der Entwicklung von Siena ist die Umsetzung eines verteilten Ereignisnotifikationsdienstes, der internetweit skalierbar ist. Wenn in diesem Zusammenhang auch oft von einem Multi-Broker-Notifikationsdienst gesprochen wird, ist dies im Fall von Siena nicht korrekt, da Broker nicht als Broker, sondern als *Server* bezeichnet werden. Die Klienten (Publisher und Subscriber) sind mit den Servern verbunden, die wiederum die Kommunikation organisieren. Die Server bilden wie auch bei REBECA dabei ein logisches Overlay-Netzwerk, das den Austausch von Subskriptionen und Notifikationen organisiert und durchführt. Das Routing selbst erfolgt auf der Basis von überdeckungsbasiertem Routing (Covering-based Routing) sowohl in hierarchischer und in der Peer-to-Peer-Variante, womit Siena deutlich weniger Routingalternativen und Anpassungsmöglichkeiten bietet als REBECA.

Für das Routing und die Zuordnung von Notifikationen zu den Subskriptionen speichern die Siena-Server Subskriptionen und Ankündigungen so, dass die Beziehungen zwischen den Eintragungen nachverfolgt werden können, also das überdeckungsbasierte Routing auch bei Widerruf einer überdeckenden Subskription weitergeführt wird. Dazu wird ein Geflecht von Filtern und Verweisen angelegt. An dieser Stelle nutzt Siena das Prinzip partiell geordneter Mengen (*Partial Ordered Set (POSET)*). Solch ein partiell geordnetes Mengenkonstrukt von Filtern wird durch jeden Server verwaltet und immer dann aktualisiert, sobald eine neue Subskription bzw. ein Subskriptionswiderruf beim Server eingeht und durch diesen verarbeitet wird. Analog geschieht dies auch für Ankündigungen.

Die lokale Verwaltung von Mengenbeziehungen ist an dieser Stelle zwar aufwändig, ermöglicht allerdings die effiziente Beurteilung, ob ein Matching zwischen Ankündigung, Subskription und Notifikation vorliegt [31]. Für solch eine Verbesserung wird die partiell geordnete Menge traversiert, sodass ein Vergleich am Wurzelknoten (bei dem Filter, der alle anderen überdeckt) beginnt und sukzessive in Richtung der Blätter voranschreitet. Falls ein Filter nicht mehr mit der Notifikation übereinstimmt, trifft dies auch für alle seine Nachfolger zu, eine weitere Traversierung ist dann nicht mehr erforderlich. Um Überdeckungen festzustellen, wurden weitere Methoden entwickelt und integriert, die feststellen, ob eine Übereinstimmung existiert. Dieses oftmals effiziente Verfahren bedient sich der Idee des Auszählens [32, 33, 149]. Das Matching von abgegebenen Filtern und

veröffentlichten Notifikationen beruht in Siena auf der Auswertung des gesamten Inhalts von Notifikationen, welche wiederum aus typisierten Name/Wert-Paaren bestehen. Die Subskriptionen und Ankündigungen bestehen aus Konjunktionen von Attributfiltern, welche selbst wiederum Prädikate auf Ereignisattributen sind. Neben einfachen Filtern, die nur ein Filterausdruck je Attribut enthalten, existieren auch zusammengesetzte Filter (*Composed Filter*), welche mehrere Prädikate je Attributfilter erlauben.

Da Siena als Middleware bis hin zu einer internetweiten Verbreitung skaliert, wurden verschiedene Servertopologien berücksichtigt und integriert. Dazu gehören hierarchische, azyklische, generische und hybride Peer-to-Peer-Topologien, wobei neben den reinen Topologien auch Mischformen, sog. *hybride Servertopologien* möglich sind, welche möglichst die Stärken der Topologien miteinander verbinden. Durch die Option unterschiedliche Topologien zu integrieren, ist das Kommunikationssystem skalierbar und damit eine grundlegende Möglichkeit, einem dynamischen Netzwerk gewachsen zu sein. Allerdings ist die Anpassbarkeit Sienas durch fehlende Adaptivität des Routingalgorithmus eingeschränkt.

Padres. *Padres* [63] wurde mit dem Ziel entwickelt, die Verwaltung von Geschäftsprozessen und die Integration unterschiedlicher Unternehmensanwendungen zu ermöglichen, ist aber genau wie Rebeca, im Gegensatz zu den auch produktiv eingesetzten Systemen wie JMS und Corba, ein Forschungsprototyp. Das Grundgerüst von Padres baut auf Siena auf und ist damit eine verteilte, inhaltsbasierte Middleware, angereichert um verschiedene weitere Funktionalitäten. Dazu gehört auch die Erkennung komplexer Ereignismuster durch die Korrelationen von primitiven Ereignissen oder Ereignsmustern.

Gelten die von einem Klienten abgegebene Subskriptionen grundsätzlich ab der Abgabe, also für zukünftige Ereignisse (eventuell können je nach Routingalgorithmus auch aktuelle, noch zirkulierende Notifikationen weitergeleitet werden), ist für den Zugriff auf historische Notifikationen jedoch häufig eine separate Schnittstelle zu einer Datenbank nötig. Padres vereinfacht diesen Zugriff durch Bindung der Middleware an eine Datenbank und die Umsetzung der Subskriptionen in die Anfragesprache des Datenbankmanagementsystems (DBMS) [63, 133]. Voraussetzung dafür ist jedoch, dass die Einträge in der Datenbank mit entsprechenden Metainformationen bestückt, d.h. beschreibbar sind. Somit bietet Padres den Klienten die gleiche Sprache zum Ausdrücken von Subskriptionen für historische wie für aktuelle sowie zukünftige Notifikationen an. Die Notifikationen in Padres bestehen aus typisierten Name/Wert-Paaren, weisen jedoch keine Struktur (Header/Body) auf. Zur Einteilung der Notifikationen in bestimmte Klassen/Typen ist es daher nötig, dass mindestens ein Attribut der Name/Wert-Paare den Notifikationstyp beschreibt. Padres unterstützt auch Ankündigungen. Darin enthalten sind die Spezifikationen der zu erwartenden Notifikation, wobei bereits an dieser Stelle der zu erwartende Typ genannt und Informationen über den zu erwartenden Wertebereich der Attributwerte gemacht werden.

Das Routing von Notifikationen, Subskriptionen und Ankündigungen erfolgt durch das Brokernetzwerk, welches auch in Padres als ein azyklisches Brokernetzwerk ausgestaltet ist. Das für das Routing genutzte Verfahren basiert grundsätzlich auf dem überdeckungsbasierten (covering-based) Routing. Weitere Routingverfahren, wie bspw. zusammenführendes (merging-based) Routing, können aber ebenfalls genutzt werden.

Zur Überprüfung von abgegebenen Subskriptionen und veröffentlichten Notifikationen auf inhaltliche Überdeckung nutzt Padres *Java Expert System Shell (Jess)* [72]. Jess ist eine deklarative Regel-Engine auf der Grundlage des Rete-Algorithmus [70] zur Verarbeitung von Regeln, welcher selbst ein effizienter Ansatz zur Lösung des many-to-many Matchingproblems [70] ist. Zur Nutzung von Jess werden Ankündigungen und Subskriptionen in einer Rete-konformen Baumstruktur als Matching-Regeln und Matching-Strukturen verwaltet. Die Jess-Engine nimmt die als Fakten dargestellten Notifikationen und wertet sie gegen die Matching-Regeln aus. Dieses Verfahren erlaubt streng genommen nur die Auswertung atomarer Subskriptionen, durch eine Erweiterung des Rete-Algorithmus ist darüber hinaus auch die Subskription und die Auswertung komplexer Ereignismuster möglich.

Padres ist zwar in erster Linie ein Forschungsprototyp, wurde jedoch für einen Einsatz im Unternehmensumfeld spezifiziert. Daher umfasst Padres neben der reinen Weiterleitung von Notifikationen bereits Strategien zur Lastverteilung und der Fehlertoleranz [63]. Fällt bspw. ein Broker durch einen Fehler aus, muss eine neue Overlaystruktur geschaffen werden. Da dies im Fehlerfall Zeit kostet, sind zur schnellen Wiederherstellung im Vorfeld eines Fehlers redundante Pfade und Netzwerkzyklen erlaubt, auf die im Fehlerfall zugegriffen werden kann [135]. In Padres werden generell für jede Ankündigung separate und eindeutig gekennzeichnete Verteilungsbäume konstruiert. Um Rückkopplungen und damit Netzwerkzyklen zu vermeiden, werden Subskriptionen und Notifikationen mit der Kennzeichnung des Verteilungsbaums markiert. So können Weiterleitungen von doppelten Notifikationen und Subskriptionen verhindert und das Brokernetzwerk effizienter ausgenutzt werden. In puncto Lastverteilung besitzen Broker in Padres eine weitere Besonderheit. So sind sie in der Lage, lokal angebundene Subscriber an Nachbarbroker zu verweisen und diese mit ihren Subskriptionen dort verwalten zu lassen [39]. Zur Steuerung der Last sind zudem eine Reihe von Algorithmen verfügbar. Sie steuern brokerseitig bspw. die Auslastung, die Verzögerung bei Matching von Notifikationen, Ankündigungen und Subskriptionen, die Ausgangslast und verschiedenste Kombinationen dieser Eigenschaften.

Falls ein Fehler im Bereich der Broker und ihrer Verbindungen untereinander auftritt, muss dieser in einem produktiven System schnellstmöglich erkannt und behoben werden. Andernfalls würden Fehler Einfluss auf Ergebnisse und Verhalten des Systems ausüben. Fallen bspw. einzelne Broker oder die Verbindung zu ihnen aus, sind sie für ihre Nachbarn nicht mehr erreichbar. In diesem Fall wird durch die benachbarten Broker der Wiederaufbau des Broker-Overlays, allerdings ohne den ausgefallenen Broker und mit direkten Verbindungen, in-

itialisiert. Inwieweit die Broker in der Lage sind, unterschiedlich viele Ausfälle zu tolerieren, hängt maßgeblich von ihrem Wissen über das Netzwerk ab. Um einen gleichzeitigen Ausfall von n Brokern in der Nachbarschaft tolerieren zu können, benötigen die Broker Wissen über alle benachbarten Broker bis zu einer Entfernung von $n+1$ Hops. Nach der erfolgten Netzwerkkonsolidierung durch Neuaufbau der Verbindungen erfolgt das inhaltliche Recovery, welches den Abgleich von Routingtabellen und versendeten Notifikationen umfasst. So werden neben den Routinginformationen für Subskriptionen und Ankündigungen die empfangenen und weitergeleiteten Subskriptionen abgeglichen und so die fehlerhaften und/oder verlorengegangenen Subskriptionen ermittelt und anschließend erneut versendet.

Das Zusammenspiel aller Eigenschaften und Fähigkeiten ermöglicht den Einsatz von Padres als verlässliches und korrekt arbeitendes Kommunikationsmittel. So lässt sich Padres bspw. für die Orchestrierung von Diensten als auch zur Ausführung, Überwachung und Verwaltung von Geschäftsprozessen sowie zur Integration von Anwendungskomponenten einsetzen. In besonderem Maße bedeutsam ist in diesem Zusammenhang die Umsetzung eines *Enterprise Service Bus (ESB)* mittels Publish/Subscribe als Kommunikationsnahtstelle zwischen Middleware- und Webtechnologien, sowie als Zwischenstück zur Transformation von Daten, Dienstebereitstellung und Dienstevermittlung.

Im Vergleich zu Siena ist Padres um wichtige Facetten erweitert worden, vor allem im Hinblick auf Verlässlichkeit, Zuverlässigkeit und der Erkennung komplexer Ereignismuster. Die Flexibilität von Rebeca erreicht es jedoch nicht. Obgleich Anforderungen nach Zuverlässigkeit und Fehlertoleranz außer in der Welt der Geschäftsprozesse auch in verteilten, heterogenen, dynamischen Umgebungen eine Rolle spielen, steht beim ubiquitären Computing ein weiterer Aspekt im Blickpunkt, nämlich die Adaptivität und Selbstorganisiertheit. Trotz der umgesetzten Fehlertoleranz in Padres sind Adaptivität und Selbstorganisiertheit weniger gut integriert als bspw. in Rebeca.

Helferlein. Die bisher angesprochenen Forschungsmiddlewarekonzepte wurden für unterschiedliche Ziele entwickelt und bieten durch ihre Erweiterungen die Möglichkeit, auch im Umfeld von ubiquitären Umgebungen eingesetzt werden zu können. Es existieren jedoch auch Middlewarearchitekturen, die eigens für den Einsatz in einem heterogenen, dynamischen Geräteumfeld mit ubiquitären Anwendungen entwickelt wurden. Im Laufe der Zeit entstanden dazu einige Arbeiten von Kirste et al., unter anderem die Projekte *EMBASSI* und *DynaMITE* [90, 95], deren Fokus vor allem auf der Erkundung und Bereitstellung von Diensten und geeigneter Kommunikationsmechanismen sowie der Integration von Diensten und Geräten in ein bestehendes Netzwerk liegt. Besonders das Projekt DynaMITE als Nachfolger von EMBASSI erleichtert die Interaktion von Anwendungen und Geräten unterschiedlicher Herkunft in einem bestehenden Ensemble dahingehend, dass Geräte spontan über das gesamte Netzwerk hinweg interagieren können. Damit leisten sie einen Beitrag bei der Erkennung

von Nutzerinteraktionen sowie der Unterstützung der Nutzer bei bereits erkannten Zielen. Weiterhin dient DynaMITE als prototypische Implementierung der im Projekt *SodaPop* erarbeiteten Grundkonzepte der bedarfsorientierten kanalbasierten Kommunikation [92].

In der Tradition der o.g. Middlewarearchitekturen und Konzepte für Kommunikationsmechanismen, Nutzerintentionserkennung und Nutzerunterstützung steht *Helferlein*, eine Middleware für intelligente Umgebungen in dynamischen und heterogenen Geräteensembles [11]. Helferlein ist jedoch keine klassisch einsatzfähige Middleware, sondern vielmehr eine Menge miteinander verbundener Teilsysteme zur Unterstützung des raschen Prototypings in Forschung und Lehre. Dabei erfüllt Helferlein die Anforderungen nach (1) Ad hoc-Konnektivität, (2) lose gekoppelten Komponenten, (3) der Umsetzung unterschiedlicher Kommunikationsparadigmen, (4) einer benutzerfreundlichen und lesbaren Protokollbeschreibung, (5) einer persistenten, zustandsbehafteten Repräsentation von Objekten, (6) der Erweiterbarkeit um Objekte und Kommunikationsparadigmen, (7) formal spezifizierten, benutzerfreundlich lesbaren Protokollen sowie (8) einer Plattform- und Programmiersprachenunabhängigkeit [11].

Besonderer Wert wird in Helferlein auf die Integration unterschiedlicher Kommunikationsmechanismen (derzeit Publish/Subscribe und Tuple Spaces, Stand 2015) und auf die Integration unterschiedlicher Objekte (physisch und virtuell) gelegt. Dies ist für den universitären Lehr- und Forschungsbetrieb sinnvoll, werden vor allem die Geräteeinbindung und verschiedene Kommunikationsmechanismen in verteilten, dynamischen Umgebungen erforscht. Dagegen baut Rebeca vor allem auf Publish/Subscribe auf, bietet dafür in puncto Adaptivität eine Vielzahl von Möglichkeiten, bspw. in Bezug auf die Auswahl von Routingalgorithmen oder der Anpassung von Overlaystrukturen. Die Integration von Anwendungen, die mit unterschiedlichen Programmiersprachen erstellt wurden, macht Helferlein hier zu einer flexiblen Middleware und ermöglicht grundsätzlich auch die Zerlegung von Anwendungskomponenten, solange sich diese an bestimmte Schnittstellen halten und die gewünschten Funktionalitäten implementiert sind. Ein Mechanismus zur Verteilung von Anwendungen bzw. Anwendungskomponenten besteht jedoch (auch hier) nicht, wobei die Serialisierung und Deserialisierung einzelner Objekte möglich ist. Jedoch fehlen Verfahren, die sicherstellen, dass während der Serialisierung und Deserialisierung von Anwendungen exakt die gleichen Ergebnisse erzeugt werden, wie die, die ohne die Serialisierung und Deserialisierung der Anwendungen erzeugt worden wären.

2.3 Anwendungsdeployment und Anwendungsmigration

Sowohl das initiale Deployment von Anwendungen als auch das laufende Deployment im Zuge der Anwendungsmigration wurden und werden in der Informatik aus unterschiedlichen Blickwinkeln betrachtet. Das Hauptproblem mit der sich auch diese Arbeit auseinandersetzt, ist die fehlende automatische Zuordnung von Anwendungen zu Ausführungsorten, wobei bei der Zuordnung die Ansprüche der Anwendungen und die angebotenen Ressourcen zu beachten sind.

2.3.1 Anwendungsdeployment

Ist im allgemeinen Sprachgebrauch von der Installation von Software die Rede, wird meist von einem Programm gesprochen, welches von einem Datenträger gelesen oder aus einem Netzwerk heruntergeladen und auf einem System ausführbar gemacht wird. Dies umfasst neben der eigentlich zu installierenden Software selbst auch die Ausführungsumgebung, müssen doch bspw. benötigte Bibliotheken oder unverzichtbare Zusatzprogramme nachinstalliert und die Eintragung der Software in die Registry des Betriebssystems vorgenommen werden. Dieser Prozess läuft häufig automatisch unter dem Einsatz eines *Installers* ab.

Das Deployment von Software beschreibt dagegen einen Prozess, der über die Installation einer Software hinausgeht und in der deutschsprachigen Literatur auch als *Softwareverteilung* bezeichnet wird [13]. Im engeren Sinn beschreibt der Begriff die Verteilung von Software an ausgewiesene Empfänger, wobei zur Software neben der initialen Verteilung auch die Verteilung notwendiger Aktualisierungen gehört. Wird der Begriff der Softwareverteilung weiter gefasst, fallen neben der Bereitstellung und Verteilung auch alle notwendigen begleitenden Tätigkeiten, die für die Verteilung und Installation von Software notwendig sind, an [13].

In der historischen Perspektive wurde das Deployment von Software auf Rechnern in größeren Organisationseinheiten (wie Unternehmen, Behörden etc.) eingesetzt. Die in solchen Organisationseinheiten vorherrschende Architektur bestand (und besteht klassischerweise) vor allem aus Arbeitsplatzrechnern als Clients und zentralen Diensten, bereitgestellt durch Server. Auch in Unternehmen war und ist es wichtig, dass Anwendungen auf dem gleichen Versionsstand sind und miteinander kommunizieren (d.h. häufig Dokumente austauschen) können.

Die Installation von Software und nötiger Aktualisierungen im Unternehmensumfeld ist zwar ähnlich wie im privaten Bereich, allerdings besitzen Nutzer im produktiven Einsatz oftmals nicht die notwendigen Berechtigungen, um die installierte und verfügbare Software zu verwalten. Diese Aufgaben übernehmen daher i.d.R. privilegierte Nutzer (Administratoren). Mit zunehmender Anzahl von Geräten geht auch hier eine Automatisierung, d.h. Unterstützung der Softwareverteilung durch dafür geschaffene, automatisch arbeitende Dienste einher.

Rollen, Rollenmodell und Rollenzuweisung. Die Zuweisung von Anwendungen zu den sie ausführenden Geräten ist gleichbedeutend mit der Zuweisung von Rollen zu einem Objekt. Müssen bei der Zuweisung von Rollen Nebenbedingungen wie Anforderungen von Rollen an die Objekte und Geräte unterschiedlicher Eigenschaften betrachtet werden, wird in der Literatur auch vom „Zuweisungsproblem“ gesprochen, einem grundlegenden kombinatorischen Optimierungsproblem in der Mathematik und der Informatik. Doch zunächst einmal wird an dieser Stelle geklärt, was eine Rolle im Zusammenhang von verteilten Anwendungen in heterogenen Umgebungen darstellt und welche Zuordnungsmechanismen zu Geräten existieren.

Rollen und Rollenmodelle in Bezug auf die Anwendungsverteilung enthalten in Bezug auf das Anwendungsdeployment grundsätzlich zwei Sichtweisen. Das ist einerseits die Ausführung des Deployments und andererseits das Deployment, also die Zuweisung von Funktionalitäten zu Geräten selbst. Bei der Betrachtung des ersten Falls, der Ausführung des Deployments, liegt ein Prozess vor, bei dem eine Reihe von Tätigkeiten von Personen bzw. Personengruppen und auch Geräten ausgeführt wird. So ist ein Anwendungsentwickler bspw. zuständig für die Erstellung und Paketierung der zu verteilenden Software, während der Administrator eines Netzwerks für die Bereitstellung der benötigten Infrastruktur, für die Paketauslieferung und die tatsächliche Verteilung inkl. der damit verbundenen Zugriffsrechte im Netzwerk zuständig ist. Neben der Zuständigkeit bestimmter Mitarbeiter bzw. Gruppen von Mitarbeitern für bestimmte Handlungen verfügen die Gruppen zumeist auch über unterschiedliche (Zugriffs-) Rechte auf verschiedene Ressourcen. Im Zusammenhang mit unterschiedlichen Zugriffsrechten wird in der Literatur häufig von „Nutzerprofilen“ gesprochen. Das Prinzip der Nutzerprofile findet sich außer in der Informatik in vielen Bereichen des Lebens. Allerdings wird hier nicht von Nutzerprofilen gesprochen, sondern von *Rollen*, wobei der Name der Rolle häufig einen Hinweis auf die ausführende Tätigkeit ist. Als Hinweis sei an das klassische Theater mit den Rollen „Held“ und „Antiheld“ in einem Stück gedacht.

Bisher wurde sich dem Begriff der Rolle informell genähert. Was aber genau ist eine Rolle? In der Informatik begann die Beschreibung von Rollen mit dem Aufkommen der objektorientierten Programmierung durch Bachman und Daya [8], wobei Bachmans Forschungen sogar bis ins Jahr 1973 [7] zurückreichen und er damit als Begründer von Rollen in der Informatik gilt. Ins Gespräch gebracht wurde der Begriff jedoch erst 1976 von Falkenberg [62]. Das neuartige am Rollenkonzept war im Vergleich zu den damals verbreiteten Modellen wie dem Netzwerk-Datenmodell, nicht wie bisher die Definition von (Daten-) Typen und Funktionen darauf, sondern die Definition eines Datenmodells aus Sicht von Tätigkeiten/Operationen und der sie ausführenden Objekte. Auch spätere Entwicklungen wie Entwurfsmuster in der Softwareentwicklung greifen Rollen wieder gezielt durch die Definition von Verhaltensmustern auf. Den Begriff Rolle definiert Balzert [13], bezogen auf Mitarbeiter eines Unternehmens, als „[...] eine Zusammenfassung von Aufgaben [...], die in einem bestimmten Kontext (z.B. bei der Durchführung eines Geschäftsprozesses) i.d.R. von einer Person durchgeführt

werden. Rollen müssen Mitarbeitern nicht fest zugeordnet sein. Ein Mitarbeiter kann je nach Situation unterschiedliche Rollen einnehmen – auch mehrere Rollen gleichzeitig.“ Verallgemeinert und übertragen auf Softwaresysteme lässt sich eine Rolle als Klassifikation eines Objektes anhand seiner durchgeführten Aufgaben/Funktionen ansehen. Dabei kann ein Objekt seine Funktion nur solange ausfüllen, wie auch das Objekt selbst existiert und alle nötigen Eigenschaften und Fähigkeiten mitbringt, die für die Erfüllung der Rolle notwendig sind [8]. Rollen agieren zudem nicht autonom, sondern innerhalb eines Beziehungsgefüges im Zusammenhang mit anderen Rollen, nicht aber mit anderen Objekten. Ein Objekt, egal ob Mitarbeiter, Mensch oder Maschine, ist in der Lage, eine oder mehrere Rollen und damit Funktionen auszuführen, allerdings nur solange, wie das Objekt die Anforderungen der Rolle umsetzen kann. Ist diese Bedingung erfüllt, ist die Zuordnung von Rollen zu den sie ausführenden Objekten flexibel.

Rollenzuweisung in Sensornetzen. Im folgenden Abschnitt geht es um den zweiten angesprochenen Blickwinkel auf die Rollenzuweisung, nämlich die Zuweisung von Rollen im Sinne von Funktionalitäten zu Objekten anhand eines konkreten Beispiels, der Rollenzuweisung in Sensornetzen.

Die Ausübung mehrerer Rollen durch ein einziges Objekt ist nicht ungewöhnlich, so ist dies bspw. im Bereich von Sensornetzen verbreitet. Sensornetze sind geprägt durch eine Vielzahl kleiner Geräte, die mit Sensoren, einem Prozessor und eigener Energiequelle ausgestattet sind und drahtlos miteinander kommunizieren. So bieten die Geräte selbst vergleichsweise wenig Rechenleistung und Speicherplatz und sind, aufgrund ihrer übersichtlichen Ressourcen, möglichst energieeffizient zu betreiben. Durch ihre geringe Größe lassen sie sich zudem überall dort einsetzen, wo bspw. eine flächendeckende, aber trotzdem unauffällige Beobachtung einer Umgebung gewünscht ist. So wird bspw. am sog. *Smart Dust*, zu deutsch *intelligenter Staub*, für das Militär geforscht. Die Idee hierbei ist es, ein (gegnerisches) Gebiet mit Hilfe einer Vielzahl kleiner, sich spontan vernetzender Geräte aufzuklären. Dazu werden diese Geräte hinter den feindlichen Linien abgeworfen, dort sammeln sie selbstständig Informationen, tauschen diese spontanvernetzt aus und senden sie schließlich über einen Kommunikationspunkt an die eigene Truppe zurück.

Der intelligente Staub ist in Bezug auf die verteilte Sammlung von Informationen und spontane Vernetzung zwar ein extremes Beispiel, es zeigt jedoch, dass eine Zuweisung unterschiedlicher Aufgaben zu sie ausführenden Geräten notwendig ist. Im einfachen Fall existieren drei unterschiedliche Rollen [71]: (1) Abdeckung eines Umweltbereiches durch einen Sensorknoten, (2) das Clustering von Sensorknoten sowie (3) die Aggregation von Informationen innerhalb des Netzwerks. Die Zuweisung der Rollen an die Geräte im Vorfeld ist wenig sinnvoll, da weder abzusehen ist, ob ein Gerät überhaupt einen Abwurf „überlebt“ und arbeiten kann, noch an welcher Stelle es landet und zukünftig besser Daten sammelt oder sie lediglich an eine Verbindungsstation weiterzuleiten hat. Auch wenn sich die Geräte initial auf dem gleichen Ausstattungsstand befinden, sind sie jedoch

i.d.R. nicht vollständig homogen, sondern unterscheiden sich bspw. hinsichtlich ihrer Position innerhalb des Geräteensembles, in Bezug auf ihre Nachbarn sowie der Position bezüglich der Nähe zu Ereignisquellen oder Steuerungselementen. Denkbar sind außerdem Unterschiede bezüglich der verbliebenen Energiereserven oder der Verfügbarkeit einzelner Komponenten, je nach dem, welche der Geräte nach einem Abwurf noch arbeitsfähig sind.

Aufgrund dieser Unsicherheiten im Vorfeld wird die Zuweisung der Rollen erst zur Laufzeit bestimmt, welche an bestimmte, vorab formulierte Bedingungen, geknüpft ist. So müssen die Geräte, welche einen Teil der Umgebung abzudecken haben, über einen entsprechenden Sensor sowie ausreichende Batteriekapazität verfügen und höchstens einen weiteren Sensorknoten in der Nachbarschaft haben, welcher den gleichen Bereich abdeckt. Die Geräte, die darüber hinaus für das Clustering und die Informationsaggregation zuständig sind, müssen weitreichendere Anforderungen in Bezug auf die vorhandene Batteriekapazität sowie an verfügbarer Rechenleistung erfüllen.

2.3.2 Rollenzuweisung und initiales Deployment

Die Zuordnung von Rollen zu den sie ausführenden Umgebungen ist eine Instanz des Aufgabenzuweisungsproblems (*Task Assigning Problems*), welches bereits seit den 1970er Jahren bekannt ist. Ebenso bekannt ist auch, dass die Mehrzahl der Probleminstanzen in die Aufwandsklasse der NP-schweren Probleme fällt [20]. In seiner ursprünglichen Form beschreibt das Aufgabenzuweisungsproblem die Verteilung von m Aufgaben zu n möglichen Ausführungsorten. Daraus ergeben sich grundsätzlich n^m verschiedene Möglichkeiten. Ziel ist es, eine Verteilung zu finden, die anhand vorher ausformulierter Bedingungen mit zu beachtenden Kosten und einem zu erreichenden Ziel optimal ist [119]. Dabei werden Bedingungen und Kosten in einer Kostenfunktion formuliert, die bspw. die Beschreibung von Ausführungsaufwand, Speichernutzung und Weiterleitungsaufwand als Kosten aufführt. Zudem existiert eine Zielfunktion, welche das Optimierungsziel definiert und die einwirkenden Kosten beschreibt.

Die Möglichkeiten, Lösungen für das Aufgabenzuweisungsproblem zu finden, sind vielfältig. So wurden bereits 1982 drei mögliche Klassifikationen von Lösungsansätzen beschrieben [58]. Mögliche genannte Klassifikationen sind demnach (1) Ansätze aus der Graphentheorie [116, 198], (2) mathematische Lösungen [198] sowie (3) Heuristiken [12, 57, 86]. Während die ersten beiden Gruppen von Lösungen die Möglichkeit bieten, tatsächlich optimale Lösungen zu finden, können Heuristiken dies nicht garantieren, sind jedoch in der Lage, akzeptable Lösungen in einem zeitlich überschaubaren Rahmen zu liefern.

2.3.3 Verwandte Arbeiten

Wie Anwendungscode für unterschiedliche Anwendungen bei gleichberechtigten und heterogenen Geräten verteilt und die Zuweisung von Rollen durchgeführt

werden kann, wird bspw. durch Frank et al. [71] für Sensornetze beschrieben. Das dort vorgestellte Verfahren arbeitet schrittweise, wobei im ersten Schritt der Quelltext der Anwendung verteilt wird. Im einfachsten Fall erhält zunächst jedes Gerät den Anwendungsquelltext, bspw. durch Flooding. Anschließend wird der Quelltext vor Ort kompiliert und damit lokal verfügbar gemacht. Daraufhin erfolgt ein Vergleich der Anforderungen der jeweiligen Anwendung mit den verfügbaren Ressourcen, woraus ein Matchingwert ermittelt wird. Der Matchingwert beschreibt die Übereinstimmung von Anforderungen der Anwendungen und den am Gerät verfügbaren Ressourcen. Anschließend wird der lokal ermittelte Matchingwert mit den Matchingwerten der umliegenden Nachbargeräte verglichen, wobei das Gerät mit dem größten Matchingwert den Zuschlag erhält und die Anwendung ausführt. Die doppelte Ausführung von Anwendungen, aber auch gegenseitige Blockierung wird mit Hilfe selbstorganisierter Zuweisung der Anwendung umgangen. Dazu werden Tabellen und ereignisbasierte Kommunikation genutzt, die Autoren beschreiben jedoch nicht die Kriterien selbst, nach denen die Anwendungen den ausführenden Geräten zugeordnet werden und welche Techniken konkret eingesetzt werden. Stattdessen wird lediglich erwähnt, dass die selbstorganisierte Rollenzuweisung auf der Basis probabilistischer Methoden durchgeführt wird.

Das Verfahren, zunächst den gesamten Anwendungscode zu verschicken, an lokaler Stelle ausführbar zu machen und erst dann zu entscheiden, welches Gerät oder Geräte nun den Zuschlag für die Ausführung bekommen, ist jedoch nicht effizient im Sinne des gesamten Verteilungsprozesses, da zunächst alle bzw. alle Geräte einer definierten Menge von Geräten den auszuführenden Quelltext erhalten, einen Matchingwert ermitteln und erst dann über die Ausführung entscheiden. Auf diese Weise werden Weiterleitungskapazitäten und Rechenleistung zur Berechnung des Matchingwertes gebunden. Stattdessen wäre es sinnvoll, den Versand von Datenmengen bereits im Vorfeld zu begrenzen und den Quelltext nur an die Anwendungen zu versenden, welche als ausführende Geräte in Frage kommen. Weiterhin bleibt unklar, wie eine Mehrfachausführung verhindert wird und an welcher Stelle darüber entschieden wird. Grundproblem bleibt nach wie vor, eine sinnvolle (initiale) Verteilung von Anwendungen, noch dazu in einer heterogenen Umgebung, zu finden.

Ein Weg, die Verteilung des Anwendungsquelltextes an alle bzw. eine unnötig große Menge von Geräten zu umgehen, ist die Nutzung von Constraints, welche Bedingungen der Anwendungen bezüglich der ausführenden Geräte repräsentieren. Im Prinzip ist dies die Weiterführung der Idee des Matchingwerts. So arbeitet das von Kichkaylo et al. [126] vorgestellte Verfahren an der effizienten Verteilung von Anwendungen mittels formulierter Constraints und Methoden der künstlichen Intelligenz. Dabei stützen sich die Autoren auf ein bereits bestehendes System, welches Planungsmethoden künstlicher Intelligenz einsetzt. Aufgrund der Komplexität des Verteilungsproblems von Anwendungen zu ausführenden Geräten und der Vielzahl möglicher Kombinationen von Constraints in verteilten Umgebungen stoßen auch die integrierten Verfahren künstlicher Intelligenz an Grenzen. Die Autoren lösen dieses Problem des eingebun-

denen Planungswerkzeugs derart, dass sie das Verhalten von Komponenten mit Hilfe von Expertenwissen in diskrete Gruppen unterteilen und so eine, der Granularität der diskreten Verhaltensgruppen angepasste, verbesserte Platzierung der Anwendungen erreichen. Voraussetzung für die Anwendung des Verfahrens ist auch in diesem Fall, dass entsprechende Constraints bekannt sind und sich zusammenfassen lassen, sodass das Verhalten von Komponenten mit den Regeln künstlicher Intelligenz analysierbar ist. Einen ähnlichen Ansatz nutzen auch Lacour et al. [129], sie teilen ebenfalls Anwendungen in bestimmte Gruppen ein und verteilen diese dann automatisch in einer Umgebung, hier in einem Computer Grid. Die Art der möglichen Anwendungen ist unterschiedlich, wobei jedoch jede Anwendung einem Programmiermodell (z.B. parallel arbeitende Anwendungen) zugeordnet werden muss. Ausgehend von dieser Einteilung werden die gruppierten Anwendungen je nach Programmiermodell in einem Metamodell zusammengefasst. Die in das Metamodell eingeordneten Anwendungen werden anschließend mit standardisierten, aber ebenso dem jeweiligen Anwendungsmodell angepassten, Deploymentwerkzeugen und einem Planungsalgorithmus automatisch verteilt. Da jedoch nicht alle existierenden und künftigen Programmiermodelle in ein einziges Metamodell überführt werden können, ist dieser Ansatz durch die Autoren nicht weiter verfolgt worden. Daher ist auch dieser Ansatz auf bestimmte Programmiermodelle und Gruppen von Anwendungen begrenzt.

Trotz ihrer Einschränkungen zeigen beide Beispiele, wie durch die Aggregation von verfügbaren Informationen ein gemeinsames Planungswerkzeug für das automatische Deployment von Anwendungen eingesetzt werden kann. Dabei wurde jedoch von Anwendungen ausgegangen, auf denen ohne weiteres Instanzen auf den Geräten im Netzwerk erzeugt und ausgeführt werden können. Was zudem bei beiden Arten von Ansätzen auffällt, ist, dass die Steuerung bzw. die Planung des Deployments zentralisiert durchgeführt werden. Damit verbunden sind benötigtes globales Wissen an einer Stelle im Netzwerk, womit zentrale Speicherung von Informationen, Aufwand für Aktualisierungen und Abhängigkeiten einhergehen. Die Autoren der vorgenannten Arbeiten gehen von heterogenen Geräten in einem gemeinsamen Ensemble aus und ermitteln daraufhin eine Zuordnung von Anwendungen im Geräteensemble. Das dabei ausgewählte Gerät ist jedoch nur für die initiale Platzierung zuständig, eine laufende Anpassung ist nicht vorgesehen. Eine laufende Anpassung ist jedoch bei den beschriebenen Anwendungsgebieten auch nicht nötig, gehen die Autoren von statischen Umgebungen mit konstanten Ressourcen und ebenfalls konstanter Ressourcenauslastung aus.

Sind die Geräte in der ausführenden Umgebung oder die Umgebung insgesamt jedoch einer Dynamik unterworfen, kann dies auf die Anwendungsausführung insoweit Auswirkungen haben, dass sich eine gefundene, ehemals gute oder sogar optimale Lösung grundlegend verschlechtert und eine oder mehrere Anwendungen im Sinne eines definierten Platzierungsziels erneut einem anderen Gerät zugewiesen werden müssen. Daher ist neben der initialen Verteilung auch ein Ansatz nötig, der kontinuierlich die Anwendungsplatzierung überwacht und wenn nötig anpasst.

Ein Ansatz, der sowohl die initiale Platzierung von Anwendungskomponenten als auch eine laufende Adaption der Anwendungsplatzierung ermöglicht, findet sich bei [165]. Das dort vorgestellte Verfahren ist allerdings bezogen auf die Platzierung von Detektoren komplexer Ereignismuster. Dieses Problem und mögliche Lösungsansätze werden im Detail in Abschnitt 5.2.1 genauer erläutert, daher wird an dieser Stelle nur das Grundprinzip vorgestellt.

Der Ausführungsort von Anwendungen wird in einer Umgebung miteinander vernetzter Geräte gesucht. Jedes Gerät kann dabei prinzipiell jede Anwendung ausführen, allerdings sind die Geräte heterogen und unterscheiden sich anhand ihrer verfügbarer Ressourcen und deren Auslastungen, was sich in der individuellen Kostensituation niederschlägt. Das heißt, die Ausführung einer Anwendung ist je Gerät unterschiedlich teuer. Die Informationen über die Kostensituationen der Gräte werden an zentraler Stelle gesammelt und dort in einen Kostenraum übertragen. Dieser Kostenraum repräsentiert damit den aktuellen Kostenzustand der im Ensemble beteiligten Geräte. Neben der aktuellen Kostensituation stehen Informationen über den zu erwartenden Verbrauch von Ressourcen der neu zu platzierenden Anwendungen und der daraus erwarteten Kosten zur Verfügung. Anhand der Abbildung der Geräte im Kostenraum und dem zu erwartenden Konsum von Ressourcen wird anschließend an zentraler Stelle der Ort im Kostenraum gesucht, an dem die zu platzierende Anwendung am günstigsten ausführbar ist. Wird eine Platzierung gefunden, wird der Ausführungsort durch die (Rück-) Abbildung des Kostenraums auf das reale Netzwerk bestimmt. Der Aufbau des Kostenraums erfordert das Wissen um die Kostensituationen der einzelnen teilnehmenden Geräte. Er ermöglicht der „wissenden“ Komponente auch die Auswahl der Instanz, die für das Deployment der Anwendung zuständig ist. Weiterhin bietet der Kostenraum die Möglichkeit, diesen kontinuierlich nach Verbesserungsmöglichkeiten der Anwendungsplatzierung zu durchsuchen. Die initiale, aber auch die kontinuierliche Anpassung erfordern einen aktuellen Kostenraum. Jedoch sind die Haltung und Aktualisierung des Kostenraums selbst wiederum mit Aufwand und damit Kosten verbunden. Insbesondere das Versenden von aktuellen Kosteninformationen der Geräte bedeutet eine zusätzliche Belastung der beteiligten Geräte, Kommunikationskanäle und damit Kosten.

2.3.4 Kontinuierliches (Re-)Deployment

Der Begriff des Deployments als Verteilung von Anwendungen wird häufig im Zusammenhang mit der initialen Platzierung von Anwendungen gebraucht. Das kontinuierliche Deployment, also die andauernde (Neu-)Platzierung von Anwendungen gehört jedoch auch zum Deployment. Ist durch das initiale Deployment eine gültige und möglichst gute Platzierung gefunden worden, so hat diese in einem statischen Umfeld Bestand. Komplexer als im statischen Fall ist die Platzierung von Anwendungen in einem dynamischen Umfeld. In solchen Fällen genügt eine einmalige Zuweisung von Anwendungen zu den sie ausführenden Geräten nicht. Daher sind laufende Optimierungen erforderlich, welche zunächst

die Notwendigkeit von Optimierungsmaßnahmen erkennen und anschließend die Optimierung ausführen.

Mittels kontinuierlicher Platzierung/Platzierungsanpassung lässt sich die Platzierung von Anwendungen bzw. Anwendungskomponenten so gestalten, dass formulierte Anforderungen durch die sie ausführenden Geräte unter Einbeziehung der verfügbaren Geräteressourcen nicht nur einmalig, sondern dauerhaft erfüllt werden. Allerdings ist auch bei dieser Optimierung wieder die Frage zu klären, mit welchem Wissen und durch wen die Ausführung/Überwachung der damit verbundenen Tätigkeiten erfolgt.

Problembeschreibung

Ausgehend vom Problem des initialen Deployments, bei dem einmalig Anforderungen von Anwendungen mit den zur Verfügung stehenden Ressourcen verglichen und die Platzierung anschließend nach bestimmten Kriterien vorgenommen werden, ist in dynamischen Umgebungen zur dauernden Sicherstellung der Anwendungsanforderungserfüllung und Zielerreichung eine adaptive Anpassung der Anwendungsplatzierung notwendig. Grundprinzip bei der adaptiven Verteilung ist eine kontinuierliche Platzierungsanpassung und somit das kontinuierliche Re-Deployment von Anwendungen. Das Problem bei der adaptiven Platzierung ist neben der Suche nach einer gültigen und vorteilhaften Platzierung auch die Erkennung von Situationen/Zuständen, welche eine Adaption nötig machen sowie die nahtlose Weiterführung von Anwendungen bzw. Anwendungskomponenten. Bevor eine Anwendung bzw. eine Anwendungskomponente verschoben wird, müssen diese serialisiert, versendet und auf dem Zielgerät deserialisiert und ausführbar gemacht werden. Bei der Betrachung der Problemklasse stellt sich heraus, dass die Anwendungsplatzierung, wie bereits erwähnt, schon im statischen Fall im NP-schweren Bereich liegt. Damit wird klar, dass auch die Variante im dynamischen Umfeld nicht von geringerer Komplexität ist. Eine Lösung des Problems wird daher mit Hilfe von Heuristiken umgesetzt. Allerdings liegt es in der Natur von Heuristiken, dass diese sich in bestimmten Situationen vergleichsweise gut oder schlechter verhalten. Ein weiterer, bei der adaptiven Anpassung zu beachtender, Aspekt ist die Konsistenz von Anwendungszuständen. Dabei muss gewährleistet werden, dass die bereits erreichten Zustände von Anwendungen auch bei der Neuplatzierung Bestand haben, die Veränderung des Ausführungsortes darf keine Veränderungen bei der Ausführung der Anwendung nach sich ziehen. Anwendungen müssen also nahtlos verschoben werden können.

Verwandte Arbeiten

Die Suche nach gültigen und vorteilhaften Platzierungen unterliegt den gleichen Anforderungen und Einschränkungen wie die Suche nach einer gültigen und vorteilhaften initialen Platzierung. Die Problembeschreibung sowie mögliche Lösungsansätze wurden dazu bereits in Abschnitt 2.3.2 bzw. 2.3.3 beschrieben.

Zusätzlich zum initialen Deployment muss bei der kontinuierlichen Platzierung entschieden werden, wie häufig und in welchem Umfeld nach Verbesserungen gesucht wird, welche Ressourcen und Informationen einzubeziehen sind und welche Instanz über die Zuordnung von Anwendungen zu Geräten entscheidet. Das Problem, welche Informationen für eine geänderte Platzierung genutzt werden und damit in die Platzierungsentscheidung eingehen, ist grundsätzlich ähnlich zu dem Problem, welche Informationen bei der initialen Platzierung in die Platzierungsentscheidung einfließen. Dabei existieren unterschiedliche Herangehensweisen. So ist denkbar, dass bspw. Informationen aus dem näheren Umfeld des bisherigen Ausführungsortes oder demgegenüber global Informationen gesammelt und später ausgewertet werden. Weiterhin lässt sich die Häufigkeit der Vorteilhaftigkeitsüberprüfungen variieren. So lässt sich die Häufigkeit bspw. bedarfsgerecht (z.B. ereignisbasiert beim Eintreten bestimmter Schlüsselereignisse) oder periodisch mit beliebiger Frequenz ausführen. Während die Effizienz der periodischen Suche nach Verbesserungen vor allem mit der Häufigkeit von (für die Effizienz der Platzierung relevanten) Änderungen im System zusammenhängt, erfordert die ereignisbasierte Suche nach Verbesserungen eine Art des Selbstbewusstseins und der Selbsterkenntnis des Systems über Änderungen und deren Folgen für die Platzierung von Anwendungen. Die Erkennung von Zeitpunkten bzw. die Auslösung von Überprüfungen und der Umfang der für die Platzierungsanpassung einbezogenen Daten sind die Voraussetzung für die Anwendung einer Platzierungsstrategie, wie sie auch für das initiale Deployment existiert. Nachdem durch die Platzierungsstratgie entschieden wurde, welche Anwendungen wohin verschoben werden, stellt sich die Frage, wie Anwendungen nahtlos bzw. konsistent, d.h. ohne Verlust von Zuständen und ohne Änderung der Abarbeitungsfolge, an den neu zugeordneten Ausführungsort verschoben werden können.

Die nahtlose Verschiebung von Anwendungen eröffnet grob gesagt zwei Arbeitsrichtungen. Zum einen müssen bereits ausgeführte Anwendungen, sofern sie schon lauffähig zur Verfügung stehen, am bisherigen Standort „eingefroren“, zum neuen Ausführungsort verbracht und dort wieder „erweckt“ werden. Weiterhin muss dafür gesorgt werden, dass die Kommunikationsinfrastruktur so angepasst wird, dass sichergestellt ist, dass keine Informationen während der Anpassung der Platzierung verloren gehen.

Platzierungsanpassung mit der Vereinheitlichung der Ausführungsumgebung. Die Verschiebung von bereits ausgeführten Anwendungen ist, unabhängig vom Einfrieren und der nahtlosen Wiederausführung der Anwendung und der Anpassung der Kommunikationsinfrastruktur, nur dann möglich, wenn Geräte der Anwendung eine einheitliche Ausführungsplattform bieten. Im Falle von heterogenen Umgebungen kann jedoch per se nicht davon ausgegangen werden, dass diese vorhanden ist bzw. schnell genug angepasst werden kann.

Eine nahtlose Migration ist möglich, wenn die Ausführungsumgebung in einem Geräteensemble prinzipiell gleich ist. Benötigt wird also eine einheitliche Softwareschicht oberhalb der Betriebssystemebene, eine Idee, die als Middleware auf-

gegriffen wurde. Doch neben Middlewarekonzepten wie Corba oder JEE können auch weniger umfangreiche zusätzliche Softwareschichten zur nahtlosen Verschiebung von Anwendungen eingesetzt werden, so zum Beispiel durch den Einsatz virtueller Maschinen. Eine der bekanntesten und verbreitetsten Ansätze dafür ist die virtuelle Maschine von Java, die Java Virtual Machine (Java VM bzw. JVM). Von Vorteil ist, dass Java VMs und Java-basierte Programme ohnehin weite Verbreitung gefunden haben und auch ohne eine gemeinsame Kommunikationsmiddleware bereits in vielen Umgebungen mit verteilten Systemen miteinander interagieren. Wohl auch aus diesem Grund existieren zahlreiche Ansätze, die Java-Anwendungen innerhalb eines Geräteensembles nahtlos zu verschieben [24, 49]. Die beiden beispielhaft genannten Ansätze nutzen dazu die bereits von der Java VM mitgelieferten Methoden. Dabei sind die Verschiebungen (auch als Migrationen bezeichnet) neben dem Einsatz für einzelne Anwendungen auch für Prozesse innerhalb von Java-basierten Anwendungen durchführbar [145]. Die hier genannten Ansätze entstammen dem Ende des 20. Jahrhunderts bzw. dem Anfang des 21. Jahrhunderts, somit stellt die nahtlose Migration von Anwendungen auch in heterogenen Umgebungen allein keine Herausforderung mehr dar, sondern kann mit Erweiterungen und geschickter Kombination bereits mit vorhandenen Funktionalitäten der Java VMs gelöst werden. Wird die Verschiebung in Zusammenhang mit der Kommunikation und insbesondere der ereignisbasierten Kommunikation betrachtet, ergeben sich weitere Herausforderungen. Da sich diese Arbeit auf die ereignisbasierte Kommunikation mittels Publish/Subscribe bezieht, liegt es nahe, auch verwandte Arbeiten zur Migration in diesem Umfeld zu betrachten.

Grundlegender Bestandteil der Publish/Subscribe-Kommunikationsinfrastruktur ist der Notifikationsdienst, dieser ist verantwortlich für die Weiterleitung von Notifikationen, Subskriptionen und Ankündigungen sowie deren Matching. In einer verteilten Infrastruktur wird der Notifikationsdienst durch miteinander vernetzte Instanzen (Broker) des Notifikationsdienstes realisiert, jeder Broker unterhält dabei Kontakt zu benachbarten Brokern und seinen lokalen Klienten. Wird davon ausgegangen, dass die lokalen Klienten und der Broker eine Einheit bilden, muss anstatt der Migration von Anwendungen das gesamte Konglomerat von Brokern und von ihnen ausgeführte Anwendungen verschoben werden. Dieser Ansatz wird bspw. durch Huang et al. [108] genutzt und Anwendungen mit den dazugehörigen Brokern als Ganzes in einem mobilen Publish/Subscibe-Umfeld dupliziert. Der Ansatz ist sinnvoll, wenn Broker und Anwendung bzw. mehrere Anwendungen stark aneinander gekoppelt sind. Solch eine Gruppierung von Broker und einer oder mehreren Anwendungen ist prinzipiell möglich, kann ein Gerät doch mehrere Brokerinstanzen und die damit verbundenen Anwendungen ausführen. Natürlich zieht eine enge Bindung von Broker und Anwendungen bei einer Verschiebung oder Replikation einen höheren Overhead nach sich, als wenn lediglich die betroffene Anwendung verschoben wird. Schließlich muss auch bei der engen Bindung zwischen Broker und Anwendung die Serialisierung und Deserialisierung von Anwendungen und Brokern durchgeführt werden. Solange die Publish/Subscribe-Infrastruktur ausreichend schnell angepasst wird, ist auch

eine nahtlose Weiterverarbeitung möglich. Doch der beschriebene Ansatz besitzt neben dem vergleichsweise größeren Overhead weitere, teils gravierende Nachteile. Zum einen beschränkt er sich auf die Replikation von Anwendungen und ist nicht in der Lage, weitere Operationen auf den Anwendungen (z.B. Aufspalten und Deployment von Anwendungskomponenten) durchzuführen, während die Migration von Komponenten über den Umweg der Replikation und das Stilllegen der ursprünglichen Anwendungsinstanz durchgeführt werden kann. Insgesamt gesehen ist die Adaptionsfähigkeit des vorgestellten Verfahrens begrenzt.

Die Nahtlosigkeit von Platzierungsanpassungen ist als Problem bereits mit der Mobilität von Klienten aufgetaucht. Klienten, also die Empfänger der Notifikationen, sind in diesem Modell nicht fest, sondern mobil. Während Notifikationen an fest eingebundene Klienten unmittelbar ausgeliefert werden können, ist dies bei mobilen Klienten nicht so einfach umzusetzen, können sich diese doch vorübergehend oder dauerhaft an einem anderen Platz im Netzwerk und an einem anderen Broker befinden. Ist der Klient nicht an einen Broker angeschlossen, seine Subskriptionen aber noch aktiv, müssen diese vom Broker des mobilen Klienten zwischengespeichert werden. Wird der Klient wieder mit dem Broker verbunden, werden die zwischenzeitlich aufgelaufenen Subskriptionen ausgeliefert. Bewegt sich der Klient dauerhaft an eine andere Position und somit an einen Broker, verwaltet dieser die Subskriptionen des Klienten. Durch den „Umzug“ des Klienten müssen nun geeignete Maßnahmen vollzogen werden, die eine Nachlieferung der zwischenzeitlich aufgelaufenen Subskriptionen und die Aktualisierungen (auch den Widerruf von Subskriptionen am vormaligen Broker) einschließen. Ein Ansatz, der mobile Klienten in ein Publish/Subscribe-System integriert, wird durch Zeidler und Fiege [213] für REBECA beschrieben. Dieser Ansatz ist gewissermaßen eine Grundlage für die in dieser Arbeit betrachteten mobilen Anwendungen, welche im Sinne von Publish/Subscribe die Klienten darstellen. So wird eine grundsätzliche Operation einer Anwendung ermöglicht, nämlich deren nahtlose Migration. Im Sinne der adaptiven Anwendungsanpassung fehlen weitere Aspekte, wie das Selbstbewusstsein zur Erkennung einer benötigten Adaption, das Platzierungsziel sowie weitere Operationen auf Anwendungen, z.B. die Zerlegung oder Replikation.

Wer, mit welchen Informationen zu welchem Zeitpunkt was steuert, ist neben der nahtlosen Anpassung das zweite Problemfeld. Die Anwendung selbst ist dazu in der Lage, wenn sie selbstbewusst ihre umgebende Situation reflektieren kann. Die Adaptionsanweisung kann die Anwendung jedoch auch von außen erreichen. So nutzt der Ansatz von Bacon et al. [9] eine außen stehende Instanz, welche die Adaption von Anwendungen koordiniert und auslöst. Das Konzept dahinter ist „Adaption durch Supervision“.

Ein Deployment von Anwendungen ohne zentral gesammelte Informationen bietet der Ansatz von Herrmann [97]. Statt der zentralen Sammlung von Daten und dem ebenfalls zentralen Abgleich zwischen Anwendungsanforderungen und verfügbaren Ressourcen werden Anwendungen zunächst initial ohne Optimierung platziert. Die Anpassung der Anwendungsplatzierung erfolgt dagegen ad-

aptiv während der Laufzeit der Anwendungen. Die Platzierungsanpassung wird auf der Grundlage eines Kräftemodells durchgeführt. Dieses Kräftemodell beruht auf den Kosten, die durch die Ausführung der Anwendungen auf den jeweiligen Geräten entstehen. Das Kräftemodell zeigt jedoch nicht die Kosten selbst, sondern die Kräfte, die durch die möglichen Kosteneinsparungen durch veränderte Anwendungsplatzierung resultieren können. Die als Kräfte modellierten Einsparungen wirken nun auf die Anwendungsplatzierung ein, indem sie die Anwendung wie Federn in bestimmte Richtungen ziehen. Existiert eine resultierende Kraft, die von ihrem Betrag größer ist als die Beträge der Kräfte aus allen anderen Richtungen, ist die Anwendung in die Richtung der größten Kraft zu verschieben, da hier die größtmögliche Einsparung erreicht wird. Wirken nicht nur eine maßgebliche Kraft, sondern mehrere Kräfte mit gleich großen Beträgen auf die Anwendungsplatzierung ein, kommt statt der Verschiebung von Komponenten zusätzlich eine Zerlegung der Anwendung von Komponenten oder die Aufspaltung des Informationsraums durch Verdopplung der Anwendung in Betracht. Natürlich können Kräfte auch in gegenläufiger Richtung entstehen, sodass Anwendungen bspw. statt in verschiedene Richtungen wieder zu einem Ausführungsort zusammengezogen werden können.

Der Ansatz von Herrmann entstammt, wie auch der von O'Keeffe [165] (siehe Abschnitt 2.3.3), dem Bereich der ereignisbasierten Kommunikation mittels Publish/Subscribe. Dieser Ansatz ist vielversprechend, ermöglicht er doch eine Verteilung von Anwendungen bezüglich einer individuellen Kostensituation und ohne dass globales Wissen zur Verfügung stehen muss. Damit geht die Vermeidung eines Flaschenhalses und die Vermeidung von zusätzlich belastendem Informationsfluss einher, allerdings ist mittels lokalem Wissen grundsätzlich nur ein lokales Optimum erreichbar. Darüber hinaus ist zu beachten, dass eine Verschiebung von Anwendungen und Anwendungskomponenten nur dann sinnvoll ist, wenn der Aufwand für die Verschiebung von Komponenten die gewonnenen Einsparungen rechtfertigt. Daher ist ein Schwellwert von Einsparungen nötig. Erst nach dessen Erreichen findet eine Anpassung der Platzierung statt. Die genannten Eigenschaften bzw. Einschränkungen des Ansatzes sind dadurch bedingt, dass er in einer verteilten Umgebung mit lokalem Wissen und in einer bestimmten Anordnung von Geräten angewendet wird. Der Ansatz ermöglicht es jedoch, auf laufende Veränderungen reagieren zu können. Allerdings löst der Ansatz nicht alle in der Einleitung dieser Arbeit aufgeworfenen Probleme. So ermöglicht er zwar das Deployment von Anwendungen, jedoch nicht initial. Verschiebungen sind zwar möglich, wie die jeweiligen Anwendungen jedoch zu ihrem initialen Ausführungsort gelangen, wird nicht beschrieben. Der Ansatz beschreibt außerdem, wie der Ausführungsort von Anwendungskomponenten verändert werden kann. Dabei wird davon ausgegangen, dass Anwendungen bereits als miteinander kommunizierende Komponenten vorliegen. Werden Anwendungen jedoch in ihrer Gesamtheit betrachtet und deren Ausführungsort verändert, bietet der Ansatz zwar eine Umsetzungsidee, betrachtet dieses Szenario aber nicht. Außerdem wird davon ausgegangen, dass Anwendungen als Komponenten vorliegen und durch jedes am Geräteensemble teilnehmende Gerät ausführbar sind.

2.4 Selbstorganisierte Systeme

Intelligente Umgebungen sind dadurch gekennzeichnet, dass verschiedenste Geräte und Anwendungen miteinander interagieren. Doch je mehr Geräte in einem Ensemble miteinander vernetzt sind, je komplexer werden die Möglichkeiten, die Geräte miteinander zu verbinden und sie miteinander kommunizieren zu lassen. Die zunehmende Komplexität, in der Literatur auch unter dem Begriff *Komplexitätskrise* (engl. *complexity crisis*) [123] bekannt, ist ein großes Problem und zugleich limitierender Faktor für die weitere Entwicklung von Systemen [157], zumal sich das Problem der wachsenden Komplexität bei weiterer Zunahme der Geräte noch verschärft. So stößt eine zentrale Kommunikation durch die Beanspruchung von Kapazitäten (z.B. Kommunikationsbandbreite) in Richtung des zentralen Kommunikationsknotens in absehbarer Zeit an ihre Grenzen. Ebenso anfällig für Überlastungen ist die zentrale Koordination, denn auch hier müssen zunächst die für die Entscheidung relevanten Informationen an zentraler Stelle gesammelt und ausgewertet werden. Es kann rund um die zentrale Schnittstelle dann vergleichsweise schnell zu einer Überlast kommen. Die Alternative zur Zentralisierung ist, auf dezentrale Strukturierung zu setzen, das System ohne zentralisierte Kommunikation und Koordinierung zu betreiben. Kann sich ein System ohne äußeres Zutun „von innen“ heraus ohne zentrale Steuerungskomponente organisieren? Unter der Inkaufnahme, dass dezentrale Entscheidungsbefugnis mit lokalem und bis zu einem gewissen Grad begrenztem Wissen einhergeht, lässt sich gegenüber einer zentralen Koordinierung die Bildung von Flaschenhälsen vermeiden. Die Selbstorganisation ist ein Organisationsprinzip, bei dem in kleinen Einheiten dezentral koordiniert und entschieden wird. Als Einstieg wird im Folgenden zunächst dargestellt, was Selbstorganisation beinhaltet und wie Selbstorganisation im Umfeld einer verteilten, intelligenten Umgebung umgesetzt werden kann.

Populärwissenschaftlich wird die Selbstorganisation häufig im Zusammenhang mit Heilung und Regeneration diskutiert, also mit Vorgängen, die ohne äußeres Zutun von einem System eigenständig ausgeführt werden, um dieses von einem Defekt bzw. technisch ausgedrückt, aus einem fehlerhaften Zustand in einen „Normalzustand“ zu überführen. Anders ausgedrückt beschreiben die Begriffe *Selbstorganistion* bzw. *Selbstmanagement* die Fähigkeit eines Systems, sich ohne eine bestimmte Kontroll- bzw. Steuerungsinstanz organisieren zu können [182]. Die genutzten Prinzipien der Selbstorganisation sind jedoch keine exklusive Entwicklung der Informatik, sondern wurden auch bei natürlich vorkommenden Prozessen und Systemen beobachtet und auf technische Systeme angewandt. Die Anwendungsgebiete selbstorganisierter Systeme in der Informatik sind vielfältig und finden sich bspw. in autonomen Systemen und Netzwerken, Multi-Roboter-Systemen, mobilen Ad hoc-Netzwerken sowie Sensor- und Aktuatornetzen [53].

2.4.1 Entwicklung und Definition von Selbstorganisation

Beschreibungen von Prozessen, die als Selbstorganisation bezeichnet werden, finden sich in vielen Bereichen von Natur- und Ingenieurswissenschaften. So betrachten [101] unterschiedliche Definitionen und Sichtweisen der Selbstorganisation. Das Prinzip der Selbsortganisation findet sich in der Biologie [29] und Chemie [164], in besonderem Maße finden sich Beschreibungen von Selbstorganisation allerdings in der Physik [55, 88]. So wird die Selbstorganisation im physikalischen Kontext als spontane Strukturbildung durch Elemente eines Systems, welche bisher unabhängiges Verhalten gezeigt haben, bezeichnet [55]. Selbstorganisation ist dadurch gekennzeichnet, dass Elemente eines Systems zusammenarbeiten und die Strukturen, die im Laufe des selbstorganisierten Verhaltens entstehen, sich eigenständig herausbilden.

Neben den vielfältigen Blickwinkeln unterschiedlicher Wissenschaftsdisziplinen wird die Selbstorganisation auch in der Informatik in unterschiedlichen Zusammenhängen betrachtet. So integriert das TCP/IP-Kommunikationsprotokoll die Vermeidung von Überlast selbstorganisiert, bspw. mit Hilfe des Slow-Start-Algorithmus und des Congestion-Avoidance-Algorithmus. Ein weiteres, bekanntes Anwendungsgebiet selbstorganisierter Systeme sind Netzwerke. So beschreiben Robertazzi et al. [186] bereits 1986 Grundzüge selbstorganisierender Kommunikationsnetze, ein Höhepunkt der Überlegungen zu selbstorgansierenden Systemen in Sensornetzen fand sich um die Jahrtausendwende [41, 42, 183]. Ein ebenfalls verbreitetes Anwendungsgebiet zur Nutzung von Selbstorganisation ist die selbstorganisierte Mustererkennung, wie sie bspw. Kohonen [127] bereits 1990 vorgestellt hat.

Definition. Eingangs des Kapitels 2.4 wurde die Selbstorganisation nach Prehofer et al. [183] als Fähigkeit eines Systems, sich ohne eine Kontroll- bzw. Steuerungsinstanz organisieren zu können, bezeichnet. Ein System besteht dabei aus einer Vielzahl von Entitäten, welche in einer Struktur geordnet sind und eine Funktionalität bereitstellen. Strukturelle Ordnung bedeutet, dass die Entitäten in einer bestimmten Weise angeordnet sind und miteinander kommunizieren. Unter der Bereitstellung von Funktionalität wird verstanden, dass das Gesamtkonglomerat aus Entitäten, d.h. das System, (mindestens) einen bestimmten Zweck erfüllt [182]. Eine ähnliche Definition bietet auch Dressler [53]. Allerdings baut er explizit auf die Arbeit von Yates et al. [209] auf. Die bereits im Jahre 1987 erschienene Publikation charakterisiert selbstorganisierte Systeme mit den Eigenschaften (1) vollständige dezentrale Kontrolle, (2) wiederkehrende Strukturen und die (3) hohe Komplexität eines selbstorgansierten (Gesamt-) Systems. Dressler erweitert diese Definition um autonom agierende Elemente des Systems (Subsysteme) und um ein beobachtbares, globales, durch ein erkennbares Muster gekennzeichnetes Verhalten des Gesamtsystems. Dabei wird darauf hingewiesen, dass bereits die Ausführung einfacher Regeln auf der Ebene der Subsysteme so weit reichende Konsequenzen haben kann, dass das Verhalten des Gesamtsys-

tems, trotz vorhersehbaren Verhaltens der Subsysteme, nicht vorhersagbar ist [53].

Lange Zeit, und das zeigen auch die o.g. Beispiele, wurden selbstorganisierte Systeme zwar verbal beschrieben, jedoch fehlte es an einer formalen Definition. An einer solchen Definition ließe sich auch entscheiden, ob ein System überhaupt selbstorganisiert arbeitet [157]. Abhilfe schaffen die Beiträge von Herrmann, Mühl und Werner [96, 101]. Hierin definieren die Autoren ein System als selbstorganisiert, (1) wenn es adaptiv ist, sich also eigenständig an Änderungen jeglicher Art anpasst; (2) sich eigenständig strukturiert, was wiederum voraussetzt, dass überhaupt eine Struktur vorhanden ist, welche durch das System eigenständig geschaffen, angepasst und gewartet werden kann; und (3) dezentralisiert arbeitet, was bedeutet, dass die Kontrolle über die adaptive Strukturierung über die Komponenten des Systems verteilt ist. Bis hierhin gleicht sich die Definition mit der von Prehofer et al. weitgehend. Neu ist hierbei jedoch die klare, formale Definition (1) eines selbstorganisierenden Systems als adaptives Softwaresystem, was sich adaptiv in seiner Struktur an seine Umgebung anpasst, wobei die globale Struktur von den Interaktionen von Teilkomponenten abhängt und die Interaktionsregeln ausschließlich auf der Grundlage lokalen Wissens und ohne Bezug auf die globale Struktur ausgeführt werden; (2) der Adaptivität als Abbildung eines Systemzustands und eines Umgebungsszustands zu einem gültigen und akzeptabel guten Systemzustand; (3) der Struktur als Eigenschaft eines Systems mit der Erfüllung von Einschränkungen bezüglich der Freiheitsgrade der Systemkomponenten; (4) eines Softwaresystems als eine Menge wohldefinierter und interagierender Komponenten, der Identifizierung eines Systems über die Menge seiner Komponenten und einer wohldefinierten Funktion, wobei außerdem eine Teilmenge eines Systems als Subsystem definiert ist; (5) einer zentralen Kontrollinstanz eines Systems in seiner Funktion nach außen als Subsystem; (6) der Klasse der selbstorganisierenden Softwaresysteme als Teil der Klasse der Softwaresysteme, wenn sie der Definition von Adaptivität genügen, sich ändernden Bedingungen anpassen und ohne zentrale Kontrollinstanz auskommen.

2.4.2 Selbstorganisation und Self-X

Selbstorganisation steht als Begriff in der Diskussion um sich anpassende Systeme nicht allein, sondern wird oft in Verbindung mit Begriffen wie Adaptierbarkeit, Selbstmanagement, Selbstkonfiguration und Selbstheilung verwendet. Je nach Forschungsbereich werden einige Begriffe häufiger genannt und spielen eine zentrale Rolle, dabei werden unterschiedliche Begriffe synonym verwendet. Es kommt jedoch ebenfalls vor, dass in den Veröffentlichungen unterschiedliche Begriffe genutzt werden. Dies geschieht vor allem, um die eigenen Forschungstätigkeiten abzugrenzen.

Ein im Zusammenhang mit der Selbstorganisation häufig fallender Begriff ist *Self-X*. Gegenüber der Selbstorganisation ist Self-X jedoch keine Eigenschaft eines Systems, sondern eine Sammlung von Begriffen, welche die (Autonomie-)

Eigenschaften eines Systems bezüglich eines Sachverhalts (X) beschreiben. So werden unter Self-X bspw. folgende Eigenschaften zusammengefasst [53]:

- Selbstkonfiguration (engl. *self-configuration*).
  Unter Selbskonfiguration werden all jene Methoden zusammengefasst, die nötig sind, um Konfigurationsänderungen und -anpassungen eines Systems durchzuführen, welche durch geänderte Umweltbedingungen erforderlich geworden sind.

- Selbstmanagement (engl. *self-management*).
  Selbstmanagement beschreibt die Fähigkeit, ein System oder ein Gerät so verwalten zu können, dass es die gemäß einem definierten Ziel gewünschten Eigenschaften aufweist.

- Selbstadaptierbarkeit (engl. *adaptability*).
  Als Selbstadaptierbarkeit wird die Eigenschaft von Komponenten eines Systems beschrieben, die es ermöglicht, sich auf ändernde Umweltbedingungen einstellen zu können.

- Selbstdiagnose (engl. *self-diagnosis*).
  Unter Selbstdiagnose werden die Mechanismen zusammengefasst, die es einem System ermöglichen, selbstständig Überprüfungen seines Zustandes durchzuführen und diese mit Referenzdaten zu vergleichen.

- Selbstschutz (engl. *self-protection*).
  Unter Selbstschutz wird die Eigenschaft eines Systems verstanden, sich und seine Komponenten selbstständig gegen ungewollte oder feindliche Einwirkungen aus der Umwelt schützen zu können.

- Selbstheilung (engl. *self-healing*).
  Unter Selbstheilung werden die Methoden verstanden, die es einem System ermöglichen, Konfigurationen und Ausführungsstrategien so verändern zu können, dass auftretende Fehler im Gesamtsystem kompensiert werden können.

- Selbstreparatur (engl. *self-repair*).
  Die Selbstreparatur ist, ähnlich der Selbstheilung, die Gesamtheit von Methoden, die notwendig ist, ein System aus einem nicht zulässigen wieder in einen konsistenten Zustand zu bringen. Im Gegensatz zur Selbstheilung, in der die Auswirkungen auftretender Fehler im Gesamtsystem zu begrenzen sind, hat die Selbstreparatur vor allem die Reparatur und die Kompensierung von Fehlern einzelner Systemkomponenten zum Ziel.

- Selbstoptimierung (engl. *self-optimization*).
  Unter den Begriff der Selbstoptimierung fällt die Fähigkeit eines Systems, die Ausführung lokaler Operationen so zu verändern, dass diese gemäß eines definierten Optimierungsziels hin agieren.

Meiner Meinung nach treffender ist die Definition von Selbstorganisation, Self-X und die Abgrenzung der beiden Begriffe im Umfeld der *Organic Computing*-Initiative [162] gelungen. Als gemeinsam genutzte Grundlage für die dort realisierten Projekte wird Selbstorganisation als die Eigenschaft eines Systems (bestehend aus einfachen Subsystemen) beschrieben, welches in der Lage ist, hochkomplexe und adaptive sowie robuste und flexible Gesamtsysteme formen zu können. Selbstorganisation beschreibt damit die grundlegende Eigenschaft eines Systems, während Self-X dagegen die Reihe von Eigenschaften eines selbstorganisierenden Systems bezüglich eines bestimmten Sachverhalts, also die des „X“ beschreibt. Dazu gehören dann die Aspekte Selbstkonfiguration, Selbstoptimierung, Selbstheilung, Selbstschutz, Selbstbeschreibung und Selbsterklärung [160, 161, 188, 189].

Als Essenz der genannten Beschreibungen der Selbstorganisation ist hervorzuheben, dass sie für aus Komponenten bestehende Systeme gelten, deren Verhalten in Richtung eines erkennbaren, globalen Ziels wirkt, das jedoch mittels lokalem Wissen erreicht wird.

2.4.3 Grundlegende Techniken der Selbstorganisation

Um zu verstehen, wie Selbstorganisation umgesetzt wird, werden im Folgenden einige grundsätzliche Techniken und Methoden skizziert, wie sie für selbstorganisierte Systeme Anwendung finden [53]. Dazu gehören positive und negative Rückkopplung, Interaktionen zwischen Elementen eines Systems untereinander und ihrer Umwelt, sowie probabilistische Techniken, welche einzeln, aber auch in Kooperation eingesetzt werden können.

Positive und negative Rückkopplung. Unter dem Begriff Rückkopplung bzw. *Feedback* wird im allgemeinen Sprachgebrauch die Rückmeldung über ein beobachtetes Verhalten beschrieben, wobei das beobachtete Verhalten für gut oder schlecht befunden wird. Durch die Rückmeldung wird Verhalten entweder gefördert (und damit verstärkt) oder abgeschwächt bzw. unterbunden. Nach diesem Grundprinzip der Verstärkung oder Abschwächung durch Rückmeldung wird bei technischen Systemen mit Regelung bzw. *Rückkopplung* übersetzt und ist ein grundlegender und weit verbreiteter Steuerungs- bzw. Regelmechanismus. Ausgangspunkt ist dabei ein System, das selbstständig seinen Zustand und den seiner Umgebung erkennt und in der Lage ist, seinen Zustand durch eigene Handlungen verändern zu können.

Eine Möglichkeit der Anpassung eines Systems an den Umgebungszustand ist die positive Rückkopplung (auch unter dem Begriff *Mitkopplung* bekannt). Ein System besitzt dann Möglichkeiten, durch Aktionen seinen Zustand selbst beeinflussen zu können, wobei der neu erreichte Zustand wiederum als Ausgangspunkt für weitere Aktionen dient. Dieser Effekt wird auch als Selbstverstärkung bezeichnet. Die wiederholte Ausführung selbstverstärkter Aktionen kann bereits

als Verhaltensmuster bezeichnet werden. Andauernde Selbstverstärkung kann jedoch auch zu Problemen führen. Zwar kann sich ein System in der Theorie unbegrenzt verstärken, in der Praxis ist dies jedoch aufgrund von Ressourcenrestriktionen nicht möglich, sodass am Ende Schneeballeffekte und Implosionen möglich sind [53]. Der Schneeballeffekt entsteht, wenn ein System mehr als ein anderes System beeinflusst und dieses ebenfalls auf mehr als ein weiteres System positiv einwirkt. Die Änderung eines Systems bedingt somit die Änderung einer Vielzahl weiterer Systeme. Zu einer Implosion kommt es dagegen immer dann, wenn viele Systeme eine Änderung durchführen, diese wiederum von anderen Systemen aufgenommen wird, diese ebenfalls reagieren und es so zu einer Aufsummierung und Verstärkung der ursprünglichen Änderung kommt. Sobald ein System mit den aufsummierten Änderungen überfordert ist, kommt es zur Implosion. Als Gegenstück zur positiven Rückkopplung agiert die *negative Rückkopplung*, die auch als *negatives Feedback* oder *Gegenkopplung* bezeichnet wird. Sie dient dazu, die Auswirkungen der positiven Rückkopplung im Rahmen zu halten, d.h. den Zustand eines Systems innerhalb eines zulässigen Fensters möglicher Parameterwerte zu halten [53]. Vereinfacht gesagt, bewirkt die Mitkopplung eine Verstärkung, während sich die Gegenkopplung hemmend auf die Verstärkung auswirkt.

Bei einer sensitiven Steuerung und Koordination beider Mechanismen bewegt sich ein System in einem zulässigen Zustandsraum und kann trotzdem auf Änderungen reagieren, im Idealfall schwingt das System in einem gewissen, zulässigen Rahmen. Erfolgt die Koordination von Mit- und Gegenkopplung allerdings nicht feinfühlig genug, kann es entweder zu Überreaktionen (bei starker Mitkopplung und schwacher Gegenkopplung) oder einem starren System (bei schwacher Mitkopplung und starker Gegenkopplung) bzw. einer Oszillation zwischen beiden Extremen kommen.

Direkte und indirekte Interaktionen. Jede Interaktion zwischen Systemen, egal ob gleichberechtigt, eingebettet oder in der Beziehung zwischen Ober- und Subsystem, erfordert Kommunikation. Grundsätzlich existieren zwei unterschiedliche Prinzipien mit denen Systeme mit anderen Systemen, mit anderen Instanzen oder auch mit sich selbst kommunizieren können. Die Grundprinzipien lauten direkte und indirekte Kommunikation [29]. Direkte Kommunikation erfolgt mittels Austausch von Informationen zwischen (in Zeit und Raum) benachbarten Systemen; im einfachsten Fall über Signale, Nachrichten, etc. Wie Informationen dabei verteilt werden, entscheidet die genutzte Verteilungsstrategie, die sich grob in gerichtete und streuende Verteilung von Informationen unterteilen lässt.

Eine andere Möglichkeit des Informationsaustauschs ist die indirekte Interaktion über die Umgebung. Inspiriert wird dieses Verfahren u.a. von der Kommunikation zwischen Ameisen, die bspw. den Weg zu einer Futterquelle nicht direkt miteinander austauschen, sondern den Weg zu dieser anhand einer auf dem Boden ausgelegten Pheromonspur hinterlegen und erkennen. Pheromone werden von Ameisen immer dann angelegt, wenn der Weg zu einer Futterquel-

le führt. Je mehr Ameisen den Weg zur Futterstelle beschreiten, je stärker ist die Pheromonspur. Davon inspiriert nutzen auch technische Systeme die indirekte Kommunikation, wenn sie Informationen in einer Umgebung ablegen, zu der auch andere Systeme Zugriff haben. Dies ist bspw. durch die Veränderung eines Systems möglich, auf das beide Kommunikationspartner Zugriff haben. So lassen sich in gemeinsam genutzten Speichern Token hinterlegen oder gemeinsam genutzte Variablen gezielt verändern. Dies entspricht dem Auslegen einer Pheromonspur, das Auslesen und Interpretieren der Daten entspricht dabei dem Aufnehmen der Pheromonspur. Im dritten Schritt erfolgt die Anpassung des Verhaltens des Systems. Durch die indirekte Art der Kommunikation können Systeme mindestens in zeitlicher Ebene voneinander entkoppelt werden.

Probabilistische Techniken. Neben Rückkopplung sowie direkter und indirekter Interaktion bilden Wahrscheinlichkeiten einen weiteren Grundbestandteil selbstorganisierten Verhaltens. Mittels probabilistischer Techniken lässt sich das Verhalten einzelner (Sub-) Systeme steuern, bzw. zur Parametrisierung deterministischer Algorithmen nutzen [53]. Nehmen wir als Beispiel eine Anwendung zur Überwachung und Steuerung der Klimatisierung eines Raums. Die Anwendung greift auf Sensordaten für Temperatur, Luftfeuchtigkeit und Sonneneinstrahlung zurück und steuert die Raumtemperatur über die Regelung der Heizung und Klimaanlage sowie die Beleuchtung und Verdunkelung über Leuchteinheiten und Jalousien. Theoretisch lässt sich jeder Ausgabewert eines Temperatursensors mit jedem möglichen Wert für Sonneneinstrahlung und Luftfeuchtigkeit kombinieren. Dies ergibt zwar einen (in diesem Beispiel) technisch beherrschbaren, trotzdem aber großen Ereignisraum. Kommen noch weitere Attribute und Abhängigkeiten, wie die Anzahl der Personen im Raum, hinzu, wird die Entscheidungsfindung auch für eine Klima- und Belichtungssteuerung sehr komplex. Aufgrund probabilistischer Techniken lässt sich der Ereignisraum jedoch beschränken. So liegt das Gebäude bspw. in Mitteleuropa, mit Außentemperaturen typischerweise zwischen -30 °C und $+50$ °C, wobei die Innentemperatur eines Raums innerhalb eines Gebäudes theoretisch auch nur zwischen -10 °C und $+40$ °C schwankt. Aus der Ausrichtung des Raums zu einer Himmelsrichtung und der Nachbarbebauung ist zudem die mögliche äußere Lichteinstrahlung bestimmbar. Durch dieses Wissen lässt sich zunächst der Ereignisraum minimieren, sodass bestimmte Merkmalsausprägungen nie oder nur sehr selten vorkommen. So sinkt bspw. die Tiefsttemperatur in Mitteleuropa selbst im Winter nie unter -40 °C und nur selten unter -30 °C. In der Folge würde die mögliche Tiefsttemperatur einer Messskala für den mitteleuropäischen Raum bis -35 °C definiert werden.

Ein übliches Verfahren zur Lösungsfindung bei komplexen Problemen ist die Zerlegung in Teilprobleme und die Anwendung einer Lösungsstrategie auf ein übersichtlicheres Teilproblem. Dieses Verfahren ist auch unter der Bezeichnung „Divide et impera“, auf deutsch als „teile und herrsche“ und auf englisch „divide and conquer“, bekannt. Wird für die Lösungssuche auf lokales Wissen zugegriffen, werden lediglich lokale Optima erreicht. Wird in einem Geräteensemble nach einer Konfiguration gesucht, die kostenoptimal Geräte miteinander verbindet, so

ist die Suche nach einer Lösung des Gesamtproblems möglicherweise zu umfangreich, um sie in angemessener Zeit zu finden. Stattdessen werden nur Teilbereiche des Netzwerks betrachtet. Die Geräte sind dabei über unterschiedliche Kanäle direkt oder indirekt über benachbarte Geräte miteinander verbunden. Wird nun anhand lokalen Wissens entschieden, wie Geräte miteinander verbunden werden, wird jeweils lokal nach den günstigsten Verbindungen gesucht, eine möglicherweise vorhandene existente Pfadkombination, die jedoch nur mit Hilfe globalen Wissens identifizierbar ist, wird nicht gefunden. Eine Möglichkeit ohne globales Wissen eine bessere Konfiguration als das lokale Optimum zu erreichen, ist die Ausnutzung probabilistischer Verfahren. Ein solches Verfahren ist bspw. das simulierte Abkühlen. Dabei werden lokale Optima zufällig um neue Optionen erweitert. Wie weit diese Sprünge reichen, hängt von einer imaginären Energie ab, die mit jedem Sprung reduziert wird, sodass die zusätzlich induzierten Variationen betragsmäßig kleiner werden.

2.4.4 Entwurf selbstorganisierender Systeme

Neben den vorgestellten Techniken existieren selbstverständlich auch Paradigmen für den Entwurf selbstorganisierender Systeme [53, 183].

So beginnt der Designprozess mit einer Beschreibung des gewünschten Verhaltens des Gesamtsystems. Dieses globale Verhalten wird anschließend auf lokale Eigenschaften reduziert. Gleichsam werden Restriktionen formuliert, inwieweit Abweichungen gegenüber des zu erreichenden Gesamtziels noch tolerierbar sind. Auf dieser Grundlage werden Verhaltensregeln für die Komponenten aufgestellt und implizite Koordinationsmechanismen entworfen. Daraufhin werden Mechanismen aufgestellt, die auch mit eingeschränkten Informationen über Systemzustände Entscheidungen treffen können. Die Art und Weise, wie diese Entscheidungen zustande kommen, wird dabei adaptiv gestaltet. Nach der Definition möglicherweise auftauchender kritischer Änderungen und der darauf folgenden Reaktionen werden auch Überwachungs- und Kontrollmechanismen entworfen.

Die folgende Auflistung gibt einen Eindruck davon, nach welchem Vorgehen ein selbstorganisierendes System entworfen werden kann. Die einzelnen Paradigmen werden dazu kurz erläutert.

- Entwerfe Regeln für lokales Verhalten, um damit globale Eigenschaften zu erreichen.

  Die Entwurfsentscheidung auf lokale und begrenzte Einheiten mit lokalen Sichten und Interaktionen mit Nachbareinheiten zu setzen, ist ein grundlegendes Prinzip für selbstorganisierte Systeme. Lokalität bedeutet allerdings auch, dass Änderungen des globalen Systems auch zu lokalen Konsequenzen führen, was sich wiederum auf die Stabilität und die Leistungsfähigkeit des Gesamtsystems auswirkt. So kann ein System durch die Aufteilung in Subsysteme insgesamt robuster werden, wenn sich Fehler nur auf einzelne

Komponenten auswirken und Auswirkungen auf andere Systemelemente begrenzt werden können. Durch die Aufteilung eines Systems in Teilsysteme können aber auch Nachteile, bspw. durch Inkonsistenzen, entstehen.

- Nutze implizite anstatt perfekter Koordinierung.

  Kernpunkt impliziter Koordinierung ist die Verlagerung von Entscheidungsprozessen von der globalen auf eine untergeordnete Ebene und damit auf kleinere (Teil-) Systeme. Explizite Koordinierung mit den damit verbundenen Nachteilen wie geringe Skalierbarkeit und der Gefahr von Engpässen wird so umgangen. Allerdings geht damit einher, dass Probleme und Konflikte bis zu einem gewissen Grad toleriert und akzeptiert werden, bis zu einem „Überspringen“ auf benachbarte Subsysteme. Dies geschieht, solange die Konflikte lokal und/oder zeitlich begrenzt sowie mit lokalen Mitteln erkenn- und auflösbar sind. Die Zusammenarbeit von Konflikterkennung und Konfliktlösung und dem Designparadigma der impliziten Koordination ermöglicht eine ressourcen- und zeiteffiziente Koordinierung in selbstorganisierten Systemen.

- Minimiere soweit wie möglich langlebige Statusinformationen.

  Die beteiligten Systemkomponenten besitzen zwei grundlegende Arten von Zustandsinformationen: vorkonfigurierte und zur Laufzeit dynamisch zugewiesene Zustandsinformationen sowie einen synchronisierten Zustand. Der Austausch der synchronisierten Systemzustände mit den anderen Systemen ist aufwendig, daher sind umso weniger zu synchronisierende Zustände zu halten, je dynamischer das System ist. Stattdessen sind die gewünschten Informationen bei Bedarf bereitzustellen.

- Entwerfe solche Protokolle, die in der Lage sind, sich unterschiedlichen Bedingungen anzupassen.

  Fest konfigurierte Systeme und Protokolle einzelner oder auch zusammengesetzter Systeme sind immer dann problematisch, sobald dynamisch Änderungen, wie mobile Komponenten, sich ändernde Anwendungsanforderungen und veränderte verfügbare Ressourcen, auftreten. Gemäß der ersten drei Paradigmen sind die Anpassungen nur lokal und auf der Grundlage lokalen Wissens durchzuführen. Die Anpassungen sind zudem stufenweise abzuarbeiten (Fehlererkennung und Reaktion, Adaption und Optimierung, Reorganisation und Struktur der Protokolle), sie sind jedoch möglichst in einem gemeinsamen Protokoll abzubilden.

  Anstatt nun für jede Stufe entsprechende Protokolle zu erarbeiten, sind Protokolle zu entwickeln, die alle drei Stufen miteinander kombinieren und so Selbstorganisation als Eigenschaft über die gesamte Bandbreite möglicher Änderungen zu integrieren.

2.4.5 Anwendungsbeispiele

Im Abschnitt 2.4.1 wurden selbstorganisierende Netze, Mustererkennung und das TCP/IP-Kommunikationsprotokoll als Beispiele für selbstorgansierende Systeme genannt. Im Folgenden wird insbesondere für die Bereiche Sensornetze und Publish/Subscribe-basierte Kommunikation beschrieben, welche Notwendigkeiten für selbstorganisiertes Verhalten existieren und mit welchen Mitteln dies umgesetzt wird.

Mobile Ad hoc-Netze und Sensornetze. Im Gegensatz zu Netzwerken mit fest vorgegebenen Netzwerkstrukturen ermöglichen Ad hoc-Netzwerke den bedarfsgerechten Aufbau von Kommunikationsverbindungen. Der Aufbau von Netzwerkverbindungen ohne zu Grunde liegende, feste Infrastruktur ist bspw. immer dann sinnvoll, wenn nach einer Naturkatastrophe keine Infrastruktur (mehr) vorhanden ist, eine Infrastruktur nicht dauerhaft betrieben und/oder spontan eine direkte Verbindung zwischen zwei oder mehreren Kommunikationspartnern aufgebaut wird.

Eine Möglichkeit in solchen Situationen zu reagieren, ist die direkte Vernetzung von Geräten. Allerdings ist grundsätzlich die Reichweite von mobiler Kommunikation durch physische Hindernisse, physikalische Einflüsse und nicht zuletzt durch die Sende- und Empfangsleistung der Geräte begrenzt. Können zudem nur direkt verbundene Geräte miteinander kommunizieren, sind entfernte Geräte nur durch aufwendigere Kommunikationsmechanismen erreichbar. Ist keine zentralisierte Kommunikation gewünscht oder möglich, müssen sich die teilnehmenden Geräte selbst organisieren, sind also ein prominentes Beispiel für den Einsatz von Selbstorganisation. Diese kommt bspw. beim Zugriff auf das Kommunikationsmedium zum Einsatz, bei dem zur Laufzeit Fehler erkannt und ausgebessert sowie Routinginformationen und ggf. Routingalgorithmen zur Laufzeit ausgetauscht werden.

Verfügen Geräte in festgelegten Netzwerken typischerweise über ausreichend Kapazitäten bezüglich ihrer Ausstattung (wie bspw. Energiespeicher oder Verarbeitungskapazität), leiden Geräte in drahtlosen Sensornetzwerken (engl. *wireless sensor networks*, kurz *WSNs*) typischerweise unter dem Mangel von Ressourcen. In der historischen Rückschau bestanden WSNs aus Geräten, welche klein sind, Umweltzustände detektieren und diese Informationen zu einer weiterverarbeitenden Stelle leiten. Im Laufe der Zeit haben sich die Fähigkeiten der beteiligten Geräte jedoch dahingehend erweitert, dass sie bereits eine Reihe von Vorverarbeitungsschritten eigenständig ausführen. Die beteiligten Geräte sind zudem heterogener bezüglich ihrer Ausstattung und Leistungsfähigkeit geworden, sie sind zunehmend mobil und agieren autonom. Das hat zur Folge, dass die Durchführung von Aufgaben und das Management der WSNs eine zunehmende Herausforderung darstellen [53]. In diesem Punkt besitzen die WSNs starke Ähnlichkeiten zu intelligenten Ensembles, welche ebenfalls heterogen und mobil sind sowie autonom agieren.

Selbstorganisierte Netze. Wie auch bei drahtlosen Sensornetzen, sind Änderungen in intelligenten Umgebungen vielfältig. So werden heterogene Geräte miteinander vernetzt, die sich hinsichtlich ihrer Architektur, ihrer Ausstattung und der darauf ausgeführten Software unterscheiden. Darüber hinaus können sich Geräte dem Netzwerk anschließen und aus ihm ausscheiden. Auch die ausgeführten Anwendungen sind dynamisch, da sie ihren Ausführungsort im Netzwerk verändern können und ihr Ressourcenbedarf sich während der Laufzeit ebenfalls verändern kann. So ist eine feste Konfiguration von Geräten, Kommunikationsverbindungen und Anwendungsinteraktion nicht praktikabel, da Anpassungen in mehrfacher Hinsicht nötig werden. Ohne eine zentrale Koordinierung, die aufgrund der Auslastung von Kommunikationskanälen nicht sinnvoll und im Sinne eines verteilt agierenden Netzwerks auch nicht gewollt ist, müssen Verfahren angewandt werden, die eine dezentrale Koordinierung und Kommunikation zwischen den Knoten ermöglichen. Das Verfahren der Wahl ist an dieser Stelle der Einsatz der Selbstorganisation.

Selbstorganisation ist in vielen Bereichen selbstorganisierter Netze einsetzbar, wobei die Bandbreite der Einsatzmöglichkeiten vom Zugriff auf das Transportmedium, über Routing, datenzentrierte Zusammenarbeit, Clustering, Kommunikation, Kooperation und Zusammenarbeit sowie Rollenzuweisung reicht [53].

Selbstorganisiertes Routing in Publish/Subscribe. Ein weiterer Anwendungsfall ist das selbstorganisierte Routing in Publish/Subscribe-Netzwerken. Wie in den vorherigen Beispielen zu sehen war, zieht die Mobilität bzw. die unterschiedliche Zusammensetzung von Geräteensembles zur Laufzeit die Notwendigkeit nach sich, die Kommunikation zu reorganisieren. Dabei wurde jedoch vernachlässigt, dass sich auch die Kommunikationsströme selbst zur Laufzeit verändern, bspw. stärker bzw. schwächer werden können. Dies kann dazu führen, dass der ausgeführte Routingalgorithmus weniger geeignet ist, als dies ein anderer an seiner Stelle wäre. Sind bspw. bei wenigen Sendern und Empfängern von Nachrichten einfache Routingmechanismen wie *Simple Routing* oder *Flooding* im Bereich von Publish/Subscribe sinnvoll, sind diese Verfahren bei zunehmender Diversifizierung von Nachrichten nicht mehr geeignet. Statt fest zugeordneter Routingalgorithmen ist die Auswahl des genutzten Routingalgorithmus vom Zustand des Systems abhängig. Der genutzte Routingalgorithmus hat sich adaptiv dem Systemzustand anzupassen. Wie dies selbstorganisiert durchgeführt werden kann, wird bspw. in [111] sowie [190] erläutert, bei denen die Auswahl der Routingalgorithmen in Abhängigkeit von der Netzwerkauslastung realisiert wird.

2.4.6 Gütebeschreibung

Selbstorganisierte Systeme haben bestimmte Eigenschaften wie dezentrale Kontrolle, verteilte Ausführung und das Runterbrechen von globalen auf lokal umsetzbare Ziele. Kann jedoch abgeschätzt werden, ob eine Umsetzung der Selbstorganisation gut, gut genug oder besser als eine andere ist? Subjektive Einschät-

zungen wie die schnelle Anpassung an eine sich ändernde Umgebung oder die Einsparung von Ressourcen fallen dem geneigten Betrachter ein. Dabei sind diese Einschätzungen natürlich weder vollständig noch wird die Frage beantwortet, wie die Eigenschaften quantifiziert werden können. Möglichkeiten, Eigenschaften von selbstorganisierten Systemen zu beschreiben, sind in Bezug auf Sensornetze bspw. die (1) Skalierbarkeit als Eigenschaft eines Systems, neue Systemkomponenten und Ressourcen ohne Leistungsverlust einzubinden, (2) Energieverwaltung und Energieffizienz und die (3) Netzwerklebensdauer als Maß einer Systemeigenschaft, die Bereitschaft von Komponenten bzw. deren Funktionalität und die Funktionalität des Systems zu sichern [53].

Eine allgemeingültige Gütebeschreibung für selbstorganisierende Systeme ist aufgrund der unterschiedlichen Anwendungsgebiete nicht sinnvoll. Die Güte eines einzelnen selbstorganisierten Systems oder einer Gruppe von Systemen wird häufig den jeweiligen Anforderungen entsprechend bewertet, wobei die angesprochenen Aspekte wie Energieeffizienz, Funktionalität und Skalierbarkeit in der Regel in einer spezifischen Kostenfunktion gewichtet werden. Die Ergebnisse einer solchen Kostenfunktion erleichtern die spezifische Vergleichbarkeit.

2.5 Diskussion

Wie die Übersicht der grundlegenden Techniken zeigt, sind im Bereich der ereignisbasierten Kommunikation mit Publish/Subscribe viele Ansätze vorhanden, die darstellen, wie unterschiedlich sich die an der Kommunikation beteiligten Komponenten organisieren können. Besonders hervorzuheben ist die Verbindung der unterschiedlichen Vorgehensweise bei der Filterung und beim Routing von Ereignissen im Zusammenhang mit der Selbstorganisation. Hier wird deutlich, dass die gewählten Filter und Routingalgorithmen anhand einer veränderten Situation ausgetauscht werden können.

Im Umfeld von Publish/Subscribe-basierten Kommunikationsumgebungen existieren bisher keine Ansätze, welche sowohl die automatische Anwendungsplatzierung und -deployment, als auch die Adaption der Platzierung von Anwendung und Anwendungskomponenten umfassen. Die Ansätze, einzelne Anwendungen und Anwendungskomponenten zu verschieben, finden sich außerhalb der Middleware und somit auf anderer Ebene als die der Kommunikation. Dazu zählen bspw. Ansätze, die im Umfeld von Java-basierten Umgebungen integriert sind und die Anwendungen oder Prozesse migrieren können [24, 49, 145]. Die Ansätze beschreiben dabei vor allem den Aufbau der zugrunde liegenden virtuellen Maschinen, die durch ihre Architektur dazu beitragen, die in der virtuellen Maschine bisher nicht separierbaren Threads und Variablen auf eine andere virtuelle Maschine übertragen zu können.

Aus den Unzulänglichkeiten der bisherigen Ansätze, Anwendungen und Anwendungskomponenten in einem dynamischen System adaptiv nach Abgleich von Anforderungen und verfügbaren Ressourcen platzieren zu können, wird in dieser

Arbeit ein Verfahren vorgestellt, dass diesen Unzulänglichkeiten begegnet. Dies beruht in seinen Grundsätzen auf der Idee, einzelne Komponenten innerhalb eines Netzwerks gleicher Umgebungen variabel verschieben und ausführen zu können. Dies wurde am Beispiel von der Verschiebung von Diensten in [97] vorgestellt. Eine naheliegende Idee ist es, Anwendungen als (mobile) Klienten aufzufassen, wie in [213] dargestellt wird. Die Anpassung der Ausführungsumgebung findet zwar hier in gewissem Umfang statt, das (initiale) Deployment selbst wird jedoch nicht betrachtet. Die Hauptaufgabe der Arbeit ist die Entwicklung eines integrierten Vorgehens, welches das Treffen einer (initialen) Platzierungsentscheidung, das Monitoring der ausgeführten Anwendungen und die ggf. nötige Adaption der Platzierung sowie die Ausführung des Deployments ermöglicht.

Kapitel 3

Automatisches Deployment und adaptive Platzierung

Inhalt

3.1 Motivation

In einer idealen, intelligenten Welt liefern verteilte, unsichtbare Sensoren zu jeder Zeit umfassende Informationen über alle verfügbaren Umweltzustände. Geräte erkennen eigenständig ihren Umgebungskontext und geben dem Nutzer die Informationen, die er gerade benötigt und verändern die Umgebung nach seinen Vorlieben und der ermittelten Intention. Geräte und Anwendungen unterstützen den Nutzer in jeder Lebenslage, am besten dezent im Hintergrund und ohne sein eigenes Zutun.

Tatsächlich sind für den Nutzer immer mehr Assistenzsysteme unterschiedlichen Zwecks am Markt erhältlich oder stehen zumindest vor der Serienreife. Die Systeme sollen dem Nutzer vor allem unangenehme Aufgaben abnehmen, sei es im Stop-and-Go-Verkehr im Auto mittels Abstandsassistent und automatischer Scheibenwischerbetätigung, Saugroboter für den Haushalt, Mähroboter für den Garten oder der eigenständig nachbestellende Kühlschrank. Bei näherer Betrachtung fällt auf, dass solche Geräte bisher Insellösungen für klar definierte Probleme sind und, anders als erhofft, trotzdem in nicht zu unterschätzendem Maße Installations- und Einstellungsarbeiten notwendig sind. So erkennt der Abstandsassistent im Auto nicht den „bevorzugten“ Sicherheitsabstand des Fahrers, wischt der durch die Regenerkennung ausgelöste Assistent auch dann, wenn nur ein einzelner Regentropfen den Bereich des Sensors trifft, erkennt der Saugroboter nicht, dass sich doch jemand in der Wohnung aufhält und stellt die Reinigungstätigkeit daraufhin ein und auch für den Betrieb eines Mähroboters müssen meist zusätzliche Kabel am Rand der zu mähenden Grünfläche verlegt werden. Der intelligente Kühlschrank wird wohl auch in naher Zukunft ein Produkt bleiben, welches sich am Markt kaum durchsetzen wird, sei es den Interessen des Einzelhandels oder des befürchteten, unkontrollierten Zugangs zum eigenen Wohnbereich durch Lieferanten geschuldet.

Soviel zu Vision und Wirklichkeit einiger, subjektiv ausgewählter Aufgabenfelder und Entwicklungsrichtungen. Was sich aber letztlich zeigt, ist eine Diskrepanz zwischen Idee und Umsetzung. Ähnlich ist die Lage auch bei Aufgabenfeldern auf unterer Ebene, bei den kleinen und mobilen Geräten. Hier ist noch immer das Vorgehen verbreitet, dass Anwendungen innerhalb intelligenter Umgebungen per Hand einem Gerät zugeordnet und dort solange ausgeführt werden, bis sie ihr Lebensende erreicht haben. Selbst bei ausreichenden Kapazitäten in Bezug auf die Rechenleistung und Speicher der ausführenden, mobilen Geräte werden Datenübertragungskapazitäten vermehrt genutzt, was wiederum mit begrenzten Energiespeichern zu Problemen führt. Sind zudem noch Geräte mit unterschiedlich ausgestatteten Ressourcen im Geräteensemble aktiv, führt dies schnell zu Ungleichgewichten im Netzwerk. Damit ist absehbar mit Engpässen zu rechnen und die manuelle Platzierung von Anwendungen ist daher nicht länger praktikabel. Um stark unterschiedliche Auslastungen und Überlastung einzelner Komponenten zu vermeiden, sind die beteiligten Komponenten gleichmäßig auszulasten. Allerdings kann ein Nutzer die Anforderungen der zu platzierenden Anwendung und die verfügbaren Ressourcen von ihm sicht- und unsichtbarer Geräte

und Verbindungen kaum selbst überblicken, noch ist er in der Lage, mögliche Engpässe oder gemeinsame Informationsmuster zu erkennen und ggf. auszunutzen, daher wird ein automatischer Deploymentprozess benötigt. Im bisherigen Fall bleibt eine einmal getätigte Platzierung bis zum Ende der Laufzeit bestehen. Dabei wird jedoch nicht beachtet, dass sich die Randbedingungen der Anwendungsausführung während der Laufzeit ändern können, bspw. durch äußere (z.B. den Eintritt/Austritt von Geräten) oder innere Einflüsse (Veränderung von Kommunikationsverbindungen, verändertes Anwendungsverhalten). Wechselnde Randbedingungen und Anforderungen in Verbindung mit einer festen Anwendungsverteilung sind nicht geeignet, Anwendungen dauerhaft anforderungskonform und ressourcensparend auszuführen. Um beide Ziele zu erreichen, muss die Platzierung während der Laufzeit überprüft und ggf. angepasst werden.

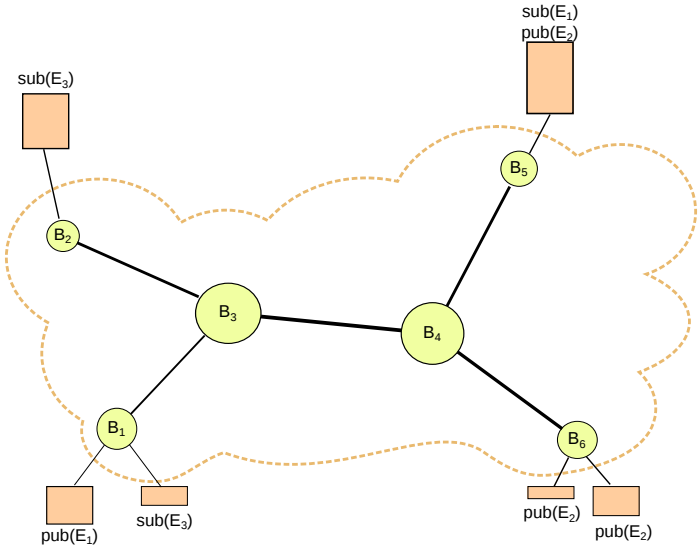


Abbildung 3.1: Publish/Subscribe-Netzwerk mit unterschiedlich leistungsfähigen Klienten, Brokern und Verbindungen.

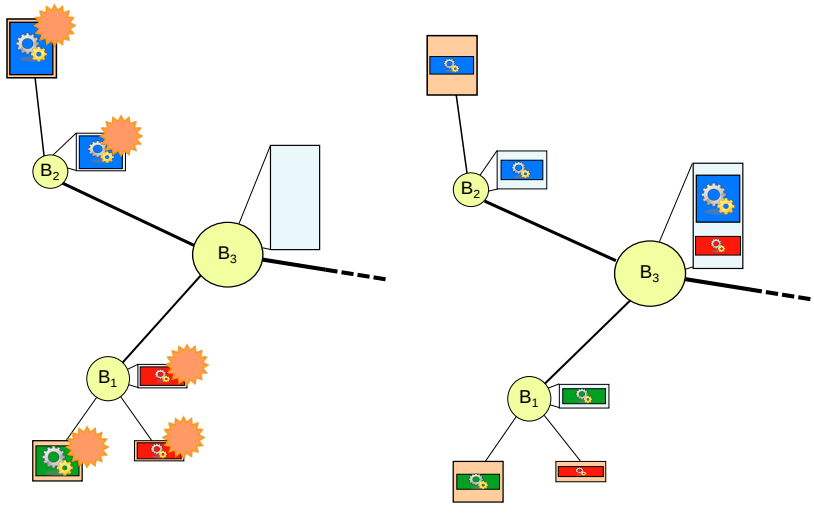
Mit dem Szenario vor Augen, dass Geräte und Anwendungen mobil sind und jederzeit aus dem Konglomerat aus Anwendungen und Geräten ein- und austreten können, zeigt die Notwendigkeit, dass die Anwendungsplatzierung in einer dynamischen Umgebung umzusetzen ist. Die Anwendungsplatzierung agiert nicht lösgelöst von der Umgebung, sondern ist in die dynamische Umgebung eingebettet. Diese muss, um die dynamischen Aspekte zu unterstützen, ein Kommunikations- und Interaktionskonzept umsetzen, welches die Erweiterung eines Geräte- und Diensteensembles überhaupt ermöglicht. Eine Möglichkeit der Umsetzung ist die Abstraktion von einzelnen Geräten und Diensten. Ein Konzept, welches diese Forderungen umsetzt ist Publish/Subscribe. Sender und Empfänger von Nachrichten kommunizieren entkoppelt voneinander, sie nutzen keine festen Adressen, sondern inhaltliche Übereinstimmungen. Der Notifikationsdienst

als Kern des Informationsaustauschs in Publish/Subscribe übernimmt die Weiterleitung von Subskriptionen, Publikationen und Ankündigungen. Die jeweiligen Instanzen des Notifikationsdienstes sind die Broker, welche typischerweise auf den Geräten des Geräteensembles ausgeführt werden. Die Broker bilden die Middleware der Geräte und bieten den Anwendungen der oberen Schichten einheitliche Schnittstellen für Interaktion und Kooperation.

Die Broker bedienen ihre angeschlossenen Klienten, welche über die Middleware miteinander kommunizieren. Die Klienten sind Anwendungen, welche neben diesem auf einer Ausführungsplattform typischerweise neben dem Broker auf demselben Gerät ausgeführt werden. Die Klienten nutzen die Dienste und Schnittstellen der ihnen zugeordneten Broker. Wird die Ausführungsplattform der Klienten vereinheitlicht, können alle Anwendungen auf jedem Klienten und damit jedem Gerät im Netzwerk ausgeführt werden. Jedoch sind die Ausführungsplattformen tatsächlich in ihrer Leistungsfähigkeit von dem sie ausführenden Gerät abhängig und unterscheiden sich diesbezüglich voneinander.

Das Zusammenspiel von Brokern als Instanzen des Notifikationsdienstes eines Publish/Subscribe-basierten Kommunikationsnetzwerks und Klienten in einer heterogenen Umgebung zeigt Abbildung 3.1. Die konkreten Instanzen der Ausführungsplattform sind unterschiedlich leistungsfähig aufgrund der sie ausführenden Geräte und die Kommunikation erfolgt über unterschiedlich leistungsfähige Kommunikationsverbindungen. Die Abbildung präsentiert ausführende Geräte (Kästchen), Anwendungen als Klienten (ebenfalls als Kästchen), Broker (Kreise) und Kommunikationsverbindungen als Kanten (Linien). In diesem Modell führt ein Gerät entweder einen Broker sowie jeweils keinen bzw. mindestens einen Klienten oder lediglich keinen oder mindestens einen Klienten aus. Klienten produzieren und/oder konsumieren Ereignisse, während die Gesamtheit der Broker den Notifikationsdienst repräsentiert und in Zusammenarbeit die Ereignisse zwischen den Klienten austauscht. Die unterschiedliche Größe der die Klienten repräsentierenden Kästchen spiegelt die Leistungsanforderungen der Klienten gegenüber den sie ausführenden Geräten wider. Nach dem gleichen Prinzip wird die Leistungsfähigkeit der Geräte durch unterschiedlich große Kästchen dargestellt. Auch die Broker sind unterschiedlich leistungsfähig, dies wird ebenfalls durch ihre Größe in der Abbildung repräsentiert. Neben den Geräten und Brokern sind die Kommunikationsverbindungen ebenfalls unterschiedlich belastbar. So schlägt sich eine hohe Leistungsfähigkeit in einer größeren Dicke der Linien nieder. Wie bereits beschrieben, müssen die Anwendungen so auf die Geräte verteilt werden, dass sie in der gewünschten Qualität ausgeführt werden, die Geräte und Ressourcen gleichmäßig ausgelastet und auch zukünftige Anpassungen zur Laufzeit möglich sind.

Werden Anwendungen manuell durch den Nutzer mit unvollständigem Wissen deployt, werden sie zumeist in der Nähe der Nutzer positioniert und damit auf Geräten, auf die die Nutzer direkt Zugriff haben. Das führt jedoch dazu, dass gerade diese Geräte mit Anwendungen stark ausgelastet sind bzw. alle benötigten Informationen zu diesen Geräten geleitet werden müssen. Dies führt u.U. dazu,



(a) Ungleichmäßige Anwendungsverteilung.

(b) Gleichmäßige Anwendungsverteilung.

Abbildung 3.2: In der linken Abbildung (a) wurden Anwendungen manuell deployt. Die die Klienten ausführenden Geräte sowie die Broker in deren unmittelbarer Nähe (B_1, B_2) sind überlastet, während Broker B_3 unterausgelastet ist. Durch das verteilte Deployment von Anwendungen im Netzwerk können Geräte und Broker gleichmäßig ausgelastet werden. Diesen Zustand zeigt die rechte Abbildung (b).

dass einige Komponenten des Netzwerks (Geräte und Kommunikationsverbindungen) überlastet, andere Komponenten jedoch unterausgelastet sind. Um einer Überlastung einzelner Geräte und Ressourcen durch manuelles Deployment mit unvollständigem Wissen und begrenzten Möglichkeiten der Einflussnahme vorzubeugen und gleichzeitig den Nutzer vom manuellen Deployment zu entlasten, wird das Anwendungsdeployment als Dienst der Middleware angeboten und sowohl die initiale Platzierung, als auch die laufende Anpassung ohne Zutun des Nutzers realisiert.

Abbildung 3.2 zeigt eine mögliche Verteilung von Anwendungen innerhalb eines dynamischen Netzwerks. Die unterschiedlichen Kapazitäten der Elemente des Netzwerks sind anhand der Durchmesser (Broker) sowie der Breiten der Linien (Kommunikationsverbindungen) zu erkennen. Die Leistungsfähigkeit der Klienten bezüglich der Ausführung von Anwendungen ist an der Größe der sie darstellenden Rechtecke erkennbar. In Abbildung 3.2(a) sind die im Netzwerk vorhandenen Geräte nicht in der Lage, neben dem Notifikationsdienst auch Anwendungen auszuführen. In dem in Abbildung 3.2(b) gezeigten Fall ändert sich dies jedoch. Die Geräte führen den Notifikationsdienst aus und stellen zusätzlich eine Ausfüh-

rungsumgebung für Anwendungen zur Verfügung. Die Leistungsfähigkeit dieser den Geräten auf Middlewareebene zugewiesenen Ausführungsumgebungen ist, analog zu den Klienten, anhand der Größe der sie darstellenden Rechtecke zu erkennen. Die ausgeführten Anwendungen werden als farbige Kästchen innerhalb der Ausführungsumgebungen dargestellt. Die Größe der Kästchen spiegelt den Ressourcenbedarf der Anwendungen wider.

Wie in Teilabbildung 3.2(a) zu sehen ist, wurden Anwendungen an verschiedenen Stellen innerhalb des Netzwerks (Broker), aber auch außerhalb dessen deployt. Dabei sind sowohl die die Anwendungen ausführenden Geräte, als auch die Broker B_1 und B_2 überlastet, während Broker B_3 freie Bearbeitungskapazitäten aufweist. Wenn eine gleichmäßige Auslastung aller Komponenten das Platzierungsziel ist, müssen die bereits deployten Anwendungen so umverteilt werden, dass sie die zur Verfügung stehenden Ressourcen gleichmäßig auslasten. Die rechte Teilabbildung 3.2(b) zeigt diesen Zustand.

Das Ziel ist, eine möglichst gute Verteilung von Anwendungen in einem Netzwerk zu finden. Das Adjektiv „gut“ steht hierbei für vorteilhaft im Sinne eines definierten Platzierungsziels, wie bspw. minimaler Energieverbrauch oder die Nähe von Verarbeitern zur Quelle von Ereignissen. Dabei kann die Platzierung von Anwendungen und Anwendungskomponenten vor oder nach dem Beginn der Arbeit der Anwendungskomponenten oder durch eine Kombination beider Vorgehensweisen erfolgen. Für eine vorteilhafte Platzierung werden von Anwendungen Informationen über die Eigenschaften der Anwendungen selbst sowie Informationen über die Beschaffenheit der möglichen Ausführungsorte benötigt. Sind solche Informationen vor der Ausführung der Anwendungen bekannt, lässt sich eine Platzierungsoptimierung im Vorfeld der Anwendungsausführung realisieren (Deployment). Lassen sich die gewünschten Informationen erst zur Laufzeit der Anwendung ermitteln, sind Optimierungen erst in diesem Zeitraum möglich (laufende Optimierung).

3.2 Verwandte Arbeiten

Sowohl für die initiale Platzierung als auch für die laufende Platzierungsoptimierung von Anwendungen und Anwendungskomponenten existieren bereits Ansätze. Einige davon werden im Folgenden vorgestellt, wobei nicht die konkreten Techniken, sondern grundsätzliche Konzepte im Vordergrund stehen. Das in dieser Arbeit vorgestellte Konzept beinhaltet auch die Anpassung und Migration sowie die Replikation von Anwendungen und Anwendungskomponenten. Dabei sei an dieser Stelle darauf hingewiesen, dass das in dieser Arbeit vorgestellte Verfahren nicht zur Erhöhung der Fehlertoleranz bzw. der Zusicherung der Ausfallsicherheit dient. Am Beispiel komplexer Ereignismuster werden für die Sicherung bzw. Erhöhung der Ausfallsicherheit bspw. Techniken zur vorbeugenden Replikation von Anwendungen eingesetzt. Einen Überblick über solche Techniken findet sich bspw. in Grüneberger et al. [85]. Die Autoren optimieren

die Gesamtauslastung des Systems unter gegebenen Ressourcenbeschränkungen mittels Erhöhung der Verfügbarkeit von Anfrageverarbeitung durch die Replikation ausgewählter Komponenten und achten bereits bei der Platzierung der Komponenten auf Ausfallsicherheit mittels unterschiedlicher Platzierungsstrategien, wobei insgesamt ein bestimmter Grad an Ausfallsicherheit nicht unterschritten werden darf.

Doch zunächst gilt die Aufmerksamkeit der folgenden Betrachtung der Detektion komplexer Ereignismuster, ein Anwendungsgebiet auf das in späteren Kapiteln (insbesondere 5.2) noch genauer eingegangen wird. Die Entwicklung von Methoden zur Erkennung komplexer Ereignismuster lässt sich auf Parallelen zweier allgemeiner Entwicklungsrichtungen in der Informatik zurückführen. Dazu gehören zum einen verteilte Ereignisströme, zum anderen die Entwicklung von Datenbanken. Wird die Detektion komplexer Ereignismuster im Kontext dieser Entwicklungen betrachtet, lassen sich die für die Teilgebiete entwickelten Techniken auch zur Detektion der Ereignismuster nutzen.

Anfragen ermöglichen in Datenbanken die Suche nach komplexen Ereignismustern, so ist eine Anfrage in einer relationalen Datenbank immer die Suche nach einer Menge von Einträgen, welche die definierten Bedingungen erfüllt. Aus der Entwicklung von Optimierungstechniken für Anfragen lassen sich Techniken ableiten, mit deren Hilfe die Suche nach komplexen Ereignismustern vereinfacht bzw. verbessert werden kann. Optimierungstechniken für Anfragen sind, vorsichtig ausgedrückt, seit längerer Zeit beschrieben und erprobt. So bieten bspw. Jarke et al. [112] bereits im Jahr 1984 einen Überblick über Anfrageoptimierungen, hauptsächlich basierend auf logischen Transformationen. Dabei bezieht sich die Betrachtung vor allem auf statische Anfragen und Arbeiten nach dem Prinzip von Vereinfachung, Bewertung und Umsortierung von Teilanfragen. Neben Optimierungen statischer Anfragen existieren umfangreiche Optimierungsansätze auch für verteilte Datenbanken [16, 59, 128], welche von ihrem Grundaufbau dem dieser Arbeit zu Grunde liegenden Anwendungsumfeld verteilter Systeme näher sind als zentrale Datenbankstrukturen. Die Optimierungsansätze arbeiten auf der Grundlage der Umformung von Anfragen. Hier ist, neben den eigentlichen Optimierungsschritten, vor allem das Grundprinzip Vorbild für das in dieser Arbeit vorgestellte Verfahren. Dies arbeitet derart, dass für jede Anfrageoperation anhand eines statistischen Modells die zu minimierenden Kosten ermittelt werden. Durch Umformung der Anfrage mittels Operationen lassen sich die Kosten variieren, so dass eine Folge von Operationen gesucht wird, welche die ursprüngliche Anfrage so verändert, dass die Kosten, die durch ihre Ausführung entstehen, minimiert werden. Ein entsprechendes Vorgehen findet sich bspw. in [16]. Die Bewertung, Umformulierung und Verteilung von Anfragen erfolgt bei Datenbanken jedoch i.d.R. vor der Ausführung der eigentlichen Anfrage, auch sicher deshalb, weil in der Betrachtung Änderungen der Infrastruktur ursprünglich keine Rolle spielen.

Eine weitere Entwicklungsrichtung aus der komplexe Ereignismuster hervorgegangen sind, ist die Verarbeitung von Datenströmen, welche auch als Datenstromverarbeitung bzw. Datenstromanalyse bezeichnet wird. Im Gegensatz zu Datenbanken, bei denen prinzipiell auf vorhandenen und hinterlegten Daten gearbeitet wird, müssen die Daten bei Datenströmen zunächst aus den Strömen herausgelöst werden. Sollen bspw. Operatoren, bekannt aus dem Datenbankstandard *Structured Query Language*, kurz *SQL*, auf strombasierte Daten angewandt werden, müssen bei Aggregatfunktionen daher erst einzelne Daten zu Bereichen zusammengefasst werden. Ein Ansatz dazu sind Fenster, welche Daten bspw. nach einer gewissen Anzahl oder ihren Eigenschaften hin zusammenfassen. Eine mögliche Ausdrucksweise ist *CQL*, die *Continuous Query Language* [6] als Erweiterung des SQL-Standards für strombasierte Daten. Die Spezifikation von CQL erlaubt bspw. die Definition sowohl von Mengen als auch von Zeitfenstern, auf denen die gewohnten SQL-Operatoren arbeiten können. So unterscheiden sich, auch bei gleicher Benennung, die Semantiken der Operatoren. Mögliche Optimierungstechniken beziehen sich dabei auf Änderungen des Operatorgraph, die Platzierung von Operatoren und die Abschätzung von Kostensituationen.

Sind Anfragen und die Herkunft der Daten bekannt, reichen statische Optimierungen aus, welche jedoch bereits NP-schwer sein können. Solch eine Optimierung ist mit der initialen Platzierung von Anwendungskomponenten vergleichbar. Kommen dynamische Änderungen hinzu, wird eine laufende Optimierung benötigt, welche im einfachsten Fall eine statistische, aber ausreichend schnelle Optimierung ist. Stehen jedoch, ob gewollt oder ungewollt, nur lokale Informationen zur Verfügung, ist eine Optimierung der gesamten Abarbeitungsfolge nicht mehr effizient möglich.

Wie bereits bei den Anfrageoptimierungstechniken für Datenbanken ist auch bei der Optimierung von komplexen Ereignismustern bei der Datenstromverarbeitung ein Grundmuster erkennbar. So existiert ein Optimierungsziel mit einem Kostenmodell, auf dessen Grundlage eine Platzierungsstrategie abgeleitet wird. Die Platzierungsstrategie beschreibt, mit welchen Operationen das Platzierungsziel zu erreichen ist. Dies geschieht durch Zuweisung von Anwendungen zu einem Ausführungsort und entweder initial oder in einem laufenden Prozess, bei dem Anforderungen und momentane Situation miteinander verglichen und Veränderungen der Platzierung durchgeführt werden.

3.2.1 Initiale Platzierung

Die initiale bzw. erstmalige Platzierung einer Anwendung erfolgt vor der erstmaligen Ausführung. Bei der initialen Platzierung erfolgt die Zuordnung der Anwendung zu einem Ausführungsort. Die Herausforderung dabei ist es, eine Zuordnung zu finden, die sicherstellt, dass die Ausführungsorte den Anforderungen der Anwendungen gerecht werden, zudem können weitere Anforderungen (bspw. abgeleitet aus einer Kostenfunktion) die Platzierung beeinflussen. Ein Synonym für das Problem der Suche nach einer passenden Platzierung für Anwendungen

ist das sog. *Component Placement Problem (CPP)*, welches selbst wenigstens im *PSPACE-schweren* Bereich liegt [125]. Es beschreibt das Problem, Anwendungskomponenten auf Rechenressourcen, Datenstrukturen und Netzwerkressourcen in einem verteilten, großräumigen System zu verteilen und dabei qualitative und quantitative Constraints einhalten zu müssen. Durch die Komplexität des Problems beschränken sich Ansätze zur Problemlösung selbst auf nur wenige, spezielle Probleme [126]. Im klassischen Anwendungsfall sind dabei die benötigten Ressourcen bekannt bzw. die Ausführungskosten im Vorfeld ableitbar. Eine Optimierung kann bereits anhand der bekannten und ermittelten Information stattfinden. Bei den bisher in dieser Arbeit vorgestellten Ansätzen wurden zudem keine Ressourcenbeschränkungen betrachtet und Optimierungen nur in Bezug auf Kosten vorgenommen. Gilt diese Maßgabe auch für die Platzierung von Anwendungen in ubiquitären Umgebungen, lässt sich eine Anwendung prinzipiell zunächst in Gänze platzieren und eine oder mehrere Optimierungsschritte erst im Nachgang ausführen. Auch bei Geräten mit beschränkten und ausgelasteten Ressourcen können Anwendungen in einem Zug platziert werden, solange eine kurzfristige Überlast toleriert wird.

Die initiale Platzierung dient zunächst dazu, überhaupt eine gültige Platzierung zu finden. Ist bereits initial eine gute oder sogar optimale Platzierung gefordert, ist dies prinzipiell zwar erst der nächste Schritt, dieser kann allerdings mit dem Auffinden einer gültigen Platzierung kombiniert werden. Damit erfolgt die Sammlung von Informationen über Anwendungsanforderungen und Ressourcen unabhängig vom späteren Platzierungsziel (Ziel der Optimierung). Bei der Sammlung von Informationen stellt sich die Frage, ob Anwendungen und Komponenten über eigenes Wissen bezüglich ihrer Anforderungen und Ressourcen verfügen oder ob diese von außerhalb heraus zunächst bewertet werden müssen. Aus dem Spannungsfeld von vorhandenem und beobachtetem Wissen einer vorhandenen Zielfunktion und der Platzierungssteuerung haben sich unterschiedliche Ansätze für die automatische initiale Platzierung von Aufgaben bzw. Anwendungen entwickelt.

Im Umfeld verteilter und komponentenbasierter Anwendungen existieren bereits Ansätze, welche die initiale Platzierung zum Inhalt haben. Einige davon werden im Folgenden genannt und ihre Grundzüge beschrieben. Die Arbeit von Kichkaylo und Karamcheti [126] beschreibt die Planung eines ressourcenabhängigen, optimalen Deployments für komponentenbasierte Anwendungen. Dazu gehören adaptive und verteilte Systeme, Web-Services und Grid-Computing. Nach Aussage der Autoren sind Methoden künstlicher Intelligenz im Umfeld von Ressourcenbeschränkungen nicht unproblematisch. Stattdessen verwenden sie Expertenwissen, mit dessen Hilfe Komponentenverhalten in diskrete Klassen eingeteilt wird. Auf diese Klassen lassen sich anschließend Planungsmethoden der künstlichen Intelligenz anwenden und so eine bessere Abbildung von Anwendungskomponenten auf verfügbare Ressourcen realisieren. Dabei arbeitet der Algorithmus phasenweise, wobei in der ersten Phase eine Kostenabschätzung auf die minimalen zu erwartenden Kosten erfolgt, zu diesem Zeitpunkt werden Ressourcenbeschränkungen und Interaktionen noch nicht berücksichtigt. In der zweiten Phase wird die Be-

rechnung der minimalen Kosten der Interaktionen zwischen den Komponenten durchgeführt und erst in der letzten Phase wird unter Berücksichtigung der geltenden Restriktionen ein Platzierungsplan erstellt.

Weitere Möglichkeiten [126] zur Lösung eines CPP bieten Algorithmen wie der *Composable Adaptive Network Services (CANS)*-Planer [73], bei dem optimale Ketten von Anwendungskomponenten entlang von Kommunikationspfaden im Netzwerk verteilt werden, unter der Beachtung von Anforderungsanfragen und dynamischer Ressourcenverfügbarkeit. Die Kommunikationspfade verbinden dabei partionierte Dienste und Komponenten mit den Ausführungsumgebungen. Dabei werden Anforderungen und verfügbare Ressourcen intervallartig miteinander verglichen und zugeordnet sowie auf maximalen Durchfluss hin optimiert.

Ein weiterer Ausgangspunkt zur Lösung des CPPs, auch aus dem Bereich des Grid-Computings und auf der Basis künstlicher Intelligenz, stammt von Blythe et al. [19]. Hier wird eine Architektur für die Konstruktion von Anwendungen für das Grid-Computing präsentiert, genauer gesagt beschäftigen sich die Autoren mit der automatischen Generierung von Aufgabenworkflows und deren Platzierung, wobei die Nutzer die Möglichkeit haben, die gewünschten Ausgabeergebnisse zu spezifizieren. Der Planer greift dabei auf Heuristiken zurück, um eine qualitativ hochwertige Lösung zu finden und generiert alternative Pläne, die anhand der Laufzeit, der Wahrscheinlichkeit einer erfolgreichen Ausführung und der Beanspruchung von Speicher- und Verarbeitungsressourcen bewertet werden. Auch dieses Vorgehen ist mehrstufig, beachtet jedoch die zur Verfügung stehenden Ressourcen nicht ausreichend und greift zudem auf externe Dienste zur Auswahl und Platzierung der Workflowkomponenten zurück [126]. Auch in Raman et al. [184] wird auf die Einbindung externer Dienste zurückgegriffen. Die Autoren entwickeln ein Platzierungsframework für ein flexibles und verteiltes Ressourcenmanagement. Dabei nutzen sie ein semi-strukturiertes Datenmodell in dem Schemainformationen, Nutzdaten und Anfragen in einer Spezifikationssprache miteinander verbunden werden. Auf dieser Grundlage erfolgt das Matching und die flexible Zuweisung der Anwendungen auf die Ressourcen. In Gribble et al. [84] wird eine Plattform namens NINJA vorgestellt, welche Anwendungskomponenten auf Cluster-basierten Plattformen ausführt. Die Anwendungskomponenten werden dabei zu einem Pfad zusammengefügt und aus verschiedenen, alternativen Pfaden wird ein Pfad ausgewählt und damit in Richtung Datenfluss und Kosten optimiert.

Einige Lösungen für das CPP auf der Grundlage von künstlicher Intelligenz, bspw. durch Vorwärtssuche oder der Verknüpfung schneller Algorithmen, haben jedoch den Nachteil, auf lineare Probleme beschränkt zu sein und nicht adäquat zu skalieren [126]. Eine Möglichkeit, die Beschränkung auf lineare Probleme zu überwinden, ist es, die Teilprobleme der Ressourcenauswahl und der Planung voneinander zu trennen [196, 205]. Dies ist vor allem dann sinnvoll, wenn Ressourcenauswahl und Planung nur lose gekoppelt sind [126].

Die Lösung des CPPs ist keinesfalls trivial, der eben vorgestellte Überblick zeigt die Herangehensweise von Algorithmen zur Lösung des Problems. Das Problem

wird zunächst in Teilproblembereiche zerlegt und anschließend mittels einer Heuristik angegangen. Die Lösungsmethoden bedienen sich der Planung und Zuweisung von Anwendungskomponenten zu Ausführungsplattformen. Planungen werden dabei zentral ausgeführt, die Menge und Güte der zur Verfügung stehenden Informationen sind mitentscheidend für die Qualität der Platzierung. In der in der vorliegenden Arbeit beschriebenen Vorgehensweise wird anlassbezogen die Platzierung einer Anwendung vorgenommen. Dazu ist eine (zentrale) Komponente vonnöten, die die Platzierung steuert. Die Planung, wie welche Anwendung zerlegt werden kann, übernimmt der Anwendungsentwickler, sodass die eigentliche Platzierung und damit die Zuordnung im Vordergrund stehen. Die betrachteten Algorithmen bemühen sich, bereits früh eine möglichst gute Platzierung zu finden und betreiben dazu einen verhältnismäßig hohen Aufwand. Da in einer dynamischen Umgebung mit Änderungen gerechnet werden muss, wird dieser Aufwand für die initiale Platzierung in der vorliegenden Arbeit nicht betrieben. Stattdessen wird initial lediglich eine gültige Platzierung gefunden. Zudem sind die vorgestellten Methoden statisch, eine laufende Anpassung ist dabei nicht vorgesehen. Die für die Durchführung der Abbildung benötigten Informationen stammen dabei von den Anwendungen selbst, indem der Anwendungsentwickler den Anwendungen entsprechende Informationen hinzugefügt hat.

3.2.2 Laufende Platzierungsanpassung

Die laufende Anpassung der Anwendungsplatzierung erfolgt als Reaktion auf eine sich ändernde, dynamische Umgebung, um die geltenden Platzierungsziele einzuhalten. Dazu können, neben den durch den Anwendungsentwickler beigefügten Informationen, auch Beobachtungen über das Verhalten der bereits ausgeführten Anwendungen genutzt werden. In der Literatur findet sich zur laufenden Platzierungsanpassung synonym der Begriff der dynamischen Platzierung.

Allgemeine Möglichkeiten der Einflussnahme. Bereits aus der historischen Entwicklung, wie bei den zuvor angesprochenen strombasierten Datenbanken, existieren dynamische Platzierungsansätze. Mögliche Verfahren finden sich in den Bereichen der Änderungen von Operatorgraphen, der Operatorplatzierung oder der Kostenabschätzung [180].

Änderungen von Operatorgraphen zielen darauf ab, dass die Reihenfolge, in der die jeweiligen Daten (hier Tupel) die Operatoren besuchen, dynamisch angepasst wird [180]. Von den dort genannten Verfahren zur Änderung der Operatorgraphen werden zwei Ansätze herausgestellt. Das erste Verfahren stammt von Nehme et al. [163], bei dem zunächst unterschiedliche Routingpläne erstellt und diese dann je nach Situation für eine bestimmte Menge von Tupeln ausgewählt werden. Statt eine Vielzahl von Routingplänen auszuarbeiten, wovon einige vielleicht nie zum Einsatz kommen werden, aber trotzdem erstellt und abgelegt werden müssen, ist die Änderung der Routingpläne bzw. Anfragepläne eine alternative Vorgehensweise dazu. Die Umordnung bei Hueske et al. [109] erfolgt

nach festgelegten Regeln, welche sicherstellen, dass die Ergebnisse der Anfrage stets gleich sind, egal ob sie mit oder ohne Umsortierung der Operatoren entstanden sind. Die Autoren gehen bei ihrer Lösung von einer beherrschbaren Dynamik aus, der mittels einer im voraus definierten Anzahl von Plänen entgegengewirkt werden kann. Auf gravierende, bisher nicht erwartete Änderungen wird dagegen nicht reagiert. Eine weitere Möglichkeit zur adaptiven Anpassung ist das Multi-Routing [206]. Hier werden keine Pläne im Vorhinein erzeugt und je nach Situation ausgewählt, sondern Tupel stapelweise zu ausgewählten Detektoren geleitet, wobei die Auswahl adaptiv und vom aktuellen Zustand des Netzwerks und der Detektoren abhängig ist. Welche Statistiken dazu besonders geeignet sind, ist ein eigenes Forschungsfeld, eine Möglichkeit zur Umsetzung ist die Verwendung zeitlich partitionierter Indizes [206].

Nach der Änderung des Operatorgraphen stehen im Folgenden Verfahren zur Anpassung der Operatorplatzierung im Fokus. So bezeichnen Pollner et al. [180] die Arbeiten von Khandekar et al. [124], Ke et al. [122] und Karnagel et al. [121] als beispielhaft. Khandekar et al. [124] beschreiben eine Lösungsidee auf der Grundlage der Überführung von Operatoren, welche zur Kompilierungszeit geschaffen werden, zu solchen, die in Softwareeinheiten (Verarbeitungselemente) überschaubarer Größe umgewandelt und platziert werden. Kernstück ist die Abbildung eines logischen Operatorgraphen auf ein physikalisches Netzwerk, wobei die dadurch entstehende Platzierung auf die Minimierung von Kommunikationskosten hin optimiert wird. Mittels Techniken zur Graphenpartitionierung werden Komponenten schrittweise in kleinere Einheiten zerlegt und anschließend einzeln einer Ausführungseinheit zugeordnet. Werden in einer einfachen Variante nur die Anwendungsanforderungen und verfügbaren Ressourcen beachtet, kommen bei einer umfangreicheren noch der Lastausgleich und benutzerdefinierte Constraints hinzu. Zur Lösung des Platzierungsproblems wird dabei eine Heuristik eingesetzt. Auch Karnagel et al. [121] bewegen sich im Umfeld heterogener Geräte, sie beschäftigen sich konkret mit der Verarbeitung von Daten in hauptspeicherbasierten Datenbanken. Es werden zunächst ein Kosten- und Operatormodell vorgestellt und genutzt, mit deren Hilfe Ausführungskosten von Operatoren ermittelt werden. Auf dieser Grundlage erfolgt die Platzierung von Operatoren, welche zwischen dem noch immer genutzten klassischen Anfrageoptimierer und der eigentlichen Anfrageausführung agieren. Zur Kostenoptimierung werden, unter Beachtung der Datentransferrate, Latenz und Ausführungsmöglichkeit je nach Situation unterschiedliche Heuristiken genutzt.

Die Definition und Nutzung eines Kostenmodells sind ein zentraler Baustein bei der Bewertung von Platzierungsalternativen und damit ein Grundbaustein für viele der bisher vorgestellten Verfahren. Anhand des Kostenmodells erfolgt die Berechnung und Gewichtung von Kosten. Das Ergebnis von Berechnungen auf der Grundlage eines Kostenmodells führt im Endeffekt zu einer Platzierungsentscheidung. Die Abschätzung oder Ermittlung von Kosten auf der Grundlage eines Kostenmodells können auf unterschiedlicher Grundlage erfolgen. Häufig werden die möglichen Einsparungen eines Operators dadurch bestimmt, wie viele Nachrichten herausgefiltert bzw. verarbeitet und nicht weitergeleitet werden.

Daneben können auch andere Messgrößen, wie bspw. Tupelraten [203], Verwendung finden. Dabei beschränken sich Kostenmodelle nicht allein auf die Anzahl von weitergeleiteten bzw. nicht weitergeleiteten Nachrichten, sondern bewerten den Durchsatz oder die Verarbeitungsgeschwindigkeit mit Kosten [180].

Verwandte Arbeiten im Umfeld ubiquitärer Systeme. Neben den zunächst allgemeinen Möglichkeiten, Anwendungen laufend neu zu platzieren, dient der nachfolgende Abschnitt dazu, Optimierungstechniken im Umfeld ubiquitärer Systeme zu betrachten.

Wird ausgehend von der Platzierung von Operatoren bzw. Anwendungskomponenten bis zu Anwendungen im Allgemeinen abstrahiert, lässt sich die Auffassung darüber, was eine Komponente ist oder darstellt, mühelos bis hin zu virtuellen Maschinen (VM) hinauf weiterführen. Wie auch bei der bisherigen Anwendungsplatzierung ist ein wichtiges Optimierungsziel die Einsparung von Kosten. Und auch die Platzierung virtueller Maschinen unterscheidet zwischen dem eigentlichen Migrationsmechanismus (siehe Kapitel 4) und dem eingesetzten bzw. zu Grunde liegenden Konzept, welches in dieser Arbeit in diesem Kapitel (3) vorgestellt wird und in der Literatur auch unter dem Namen *Policy* bekannt ist [216]. Die Kostenbestimmung für eine optimale Platzierung im Umfeld virtueller Maschinen erfolgt auf Grundlage von Energieverbrauch oder Ressourcenauslastung, womit die Optimierungsziele denen des hier beschriebenen Ansatzes sehr ähnlich sind. Und auch bei den vorhandenen Ansätzen kommt eine Heuristik zum Einsatz, welche die Anwendungsplatzierung und damit ein Problem der Klasse CPP löst. Beim Blick auf das Optimierungsziel „Kostensenkung“ durch Senkung des Energieverbrauchs lässt sich feststellen, dass der Energieverbrauch eines Gesamtsystems zwar auf der Grundlage eines bestimmten Zeitpunkts optimiert, diese Optimierung jedoch nur solange Gültigkeit hat, bis keine bedeutenden Änderungen der Umgebung oder der Anwendung eintreten, weil zukünftiges Anwendungsverhalten sowie Ressourcenverbräuche, und damit Anwendungsverhalten im Allgemeinen, nicht vorhergesagt werden können [216]. Dem Nachteil, dass Heuristiken Lösungen für ein begrenztes Anwendungsfeld und dabei häufig lokale Lösungen bieten, kann bspw. durch den Einsatz von Techniken des simulierten Abkühlens, und damit nach einer Anpassung des Suchraums entgegen gewirkt werden. So schlagen Zhao et al. [216] eine Heuristik vor, die durch Migration von virtuellen Maschinen Energieverbräuche minimiert und dabei sowohl Komponenten der Partikelschwarmoptimierung, als auch Ansätze des simulierten Abkühlens [17] nutzt. Damit können die Nachteile lokaler Optima teilweise durchbrochen und längerfristige Optimierungsprozesse ausgeführt werden. Der Rückgriff auf Methoden des simulierten Abkühlens wird auch bei anderen adaptiven Verfahren gebraucht, bspw. für Heuristiken in der Nachbarschaftssuche kontinuierlicher Variablen [141]. Als Ergänzung sei an dieser Stelle angemerkt, dass neben der simulierten Abkühlung auch genetische Algorithmen [106] zur Lösung von Optimierungsproblemen genutzt wurden und werden. Die Anwendung bewährter Methoden ist auch in der Informatik ein oft eingesetztes Prinzip, so abstrahieren Srikanataiah et al. [195] die dynamische Konsolidierung

von Anwendungen im Umfeld vom Cloud Computing auf ein mehrdimensionales Behälter- (engl. *bin packaging*-) Problem, welches mit einer Heuristik auf der Grundlage der Minimierung Euklidischer Distanzen zwischen dem derzeitigen und dem optimalen Ausführungsort angegangen wird. Eine Abstraktion des Behälterproblems zur VM-Platzierung findet sich auch in Beloglazov und Buyya [15] wieder, wobei die Autoren zur Lösung des Optimierungsproblems einen modifizierten „Best Fit Decreasing"-Algorithmus („Power Aware Best Fit Decreasing" [14]) nutzen, d.h. die zu platzierende VM dort positionieren, wo gerade noch ausreichend geforderte Ressourcen verfügbar sind. Eine weitere Besonderheit der Arbeit ist der Einsatz adaptiver Detektion von überlasteten Ausführungsorten, neu zu platzierenden VMs und der Auswahl der zukünftigen Ausführungsorte. Die Idee der Partikelschwarmoptimierung in Zusammenarbeit mit adaptiven und selbstorganisierten Verfahren findet sich ebenfalls in dem vorgestellten Potpurri möglicher Heuristiken in Jeyarani et al. [118], bei dem durch Platzierung von VM-Instanzen auf eine Serverfarm der globale Energieverbrauch minimiert wird, was jedoch zu Lasten der Ausführungsqualität geht. Die Arbeit von [207] nutzt ebenfalls eine Heuristik und verwendet die Idee eines künstlichen Bienenschwarms für die erste Verbesserung sowie das Bayes-Theorem zur weiteren dauerhaften Optimierung. Dabei ist noch zu erwähnen, dass neben den genannten Methoden auch die uniforme und randomisierte initiale Verteilung, die binäre Suche und Boltzmann-Auswahlstrategie genutzt werden.

Wichtig bei der Bewertung der hier vorgestellten Ansätze ist, dass Optimierungen gerade im Umfeld wechselnder Umgebungsparameter dauerhaft und adaptiv auf die Umgebung durchzuführen sind. Zudem ist es ratsam, nicht nur lokale Optimierungsmaßnahmen durchzuführen, sondern auch einschneidende, globale Änderungen in die Optimierung einzubeziehen.

Neben mathematischen Modellen kommen mit den Techniken der künstlichen Intelligenz auch Methoden zum Einsatz, deren Vorbild natürliche Prozesse sind, dazu gehören das simulierte Abkühlen und Schwarmoptimierung. Eine andere Blickrichtung ist die Umsetzung technischer Lösungen anderer Disziplinen mit Abbildung auf die Informatik. Ein Beispiel dafür ist die dynamische, dezentrale Lastbalancierung in Multi-Computer-Systemen mittels Partikeloptimierung [91]. Dabei werden Anwendungen als nicht-mischbare Flüssigkeiten unterschiedlicher Dichte modelliert, welche in einem begrenzten Gefäß mit ebenem Boden, als Synonym für die ausführenden Geräte, durch ihre unterschiedliche Dichte andere Flüssigkeiten verdrängen oder selbst verdrängt werden. Beziehungen zwischen den Anwendungen werden zusätzlich als Kohäsionskräfte zwischen den Flüssigkeiten dargestellt. Die Gravitation zieht die Flüssigkeit dabei in Richtung des Gefäßbodens, ob sie diesen erreicht, hängt jedoch von der Viskosität der Flüssigkeit, der Kohäsion zu anderen Flüssigkeiten und der Reibung, die bei einer Migration entstehen würde, ab. Im Laufe der Zeit bildet sich ein stabiles Gleichgewicht, sodass sich Flüssigkeiten eher am Boden oder eher in höheren Schichten sammeln und ggf. in andere benachbarte Gefäße „überlaufen".

Eine andere Analogie aus der Mechanik sind Körper im Raum, die mittels Federn an einer bestimmten Position gehalten werden. Ist eine Feder stärker gespannt als ihr Gegenspieler, bewegt sich der Körper in Richtung Aufhängungspunkt der stärker gespannten Feder. Mit der Bewegung des Körpers gleichen sich die unterschiedlichen Federspannungen aus, die Federn an der die Anwendungen aufgehängt sind, sind gleichmäßiger gespannt und es wird ein stabiler Zustand erreicht. Die Kräfte, welche die Federn spannen, werden bspw. durch Kosten modelliert, sodass ein Ausgleich von Federspannungen zu ausgeglichenen, insgesamt geringeren Kosten führt. Dies gilt zumindest immer dann, solange in einem Kostenmodell mit steigenden Grenzkosten umzugehen ist, eine zusätzliche Einheit auf einem ausgelasteten Gerät teurer ist als die zusätzlichen Einheiten auf nicht so stark belasteten Geräten. Dieses Kräftemodell ist in der Literatur auch als *Spring Relaxation* bekannt und stellt das Grundprinzip der Anwendungsverteilung des in dieser Arbeit vorgestellten Konzepts dar. Diese Herangehensweise ermöglicht die Modellierung aller Kosten, welche auf die Platzierung von Anwendungen als Kräfte einwirken. Sind einzeln einwirkende Kräfte ausreichend groß, ändern sich die Platzierung der Anwendungen und damit auch die einwirkenden Kräfte sowie die Kosten. Dabei werden für jede Positionierung innerhalb eines Raumes individuell die einwirkenden Kräfte ermittelt. Innerhalb des Verbunds von Anwendungen und Ausführungsumgebungen wird so eine dezentrale, flexible Anpassung von Ausführungsorten ermöglicht. Auf welche Änderungen dabei reagiert wird, entscheidet das zu Grunde liegende Kräfte- und Kostenmodell.

Es existieren bereits Arbeiten, die das Prinzip der einwirkenden und sich entspannenden Kräfte nutzen. Das Einsatzgebiet umfasst dabei grundsätzlich alle Bereiche, bei denen Anwendungen innerhalb einer Umgebung bzw. Umwelt beweglich sind. So wird es für drahtlose Sensornetze [56, 214] genauso genutzt, wie für die Platzierung von Diensten in intelligenten Umgebungen [96] und die Anwendungsplatzierung in Publish/Subscribe-basierten Systemen [165, 175].

Eckert et al. [56] nutzen die Spring Relaxation für die autonome Lokalisierung von Robotern innerhalb drahtloser Sensornetze auf der Grundlage der Überlegungen von Howard et al. [107]. Ohne globales Wissen und ohne Synchronisationsmechanismen berechnen die Knoten eigenständig und selbstorganisiert ihre Position in einem virtuellen Koordinatensystem auf Grundlage von Distanzabschätzungen zu den Nachbarn. Eine weitere Möglichkeit der Selbstlokalisierung in Sensornetzen mittels Spring Relaxation kommt auch bei Zhang et al. [214] zum Einsatz, wobei diese, wie auch die vorhergehende Lokalisierungsmethode, gleichzeitig möglichst genau, aber auch energiesparend agiert. Bei der Arbeit von Zhang et al. erfolgt die Abschätzung einer Position in einem lokalen Bezugsgebiet auf Basis der Messung von Signalstärken empfangener Nachrichten. Die daraus gewonnene erste Platzierungsinformation wird anschließend, in einem zweiten Schritt verfeinert, indem Signalstärkeinformationen direkter Nachbarn mit dem aufgestellten Kräftemodell verglichen werden.

Die Positionierung von Objekten in einem echten oder virtuellen Raum kann auch für die Platzierung von Anwendungen genutzt werden. Als Beispiel dienen

hier die Arbeiten von Pietzuch et al. [175] und besonders O'Keeffe [165]. Bei beiden Ansätzen werden Anwendungen in einem Netzwerk platziert, wobei die Platzierung in einem Latenzraum stattfindet und die relative Entfernung der Geräte von der jeweiligen Latenz der Datenübertragung zwischen den Komponenten abhängt. Der Ansatz von Pietzuch zeichnet sich ebenfalls durch eine dezentrale, iterative Arbeitsweise aus, die zu platzierenden Komponenten sind Operatoren der strombasierten Datenverarbeitung. Die von O'Keeffe beschriebene Vorgehensweise [165] entwickelt die Idee von Pietzuch et al. weiter, greift allerdings auch auf die Positionierung von Komponenten in einem virtuellen Koordinatensystem und einem physikalischen Raum zurück. In diesem Raum sind Geräte positioniert, die miteinander kommunizieren und Anwendungen ausführen. Zweitens existiert ein virtueller Raum, der auf den physischen Raum abgebildet wird. Die Anordnung der Komponenten erfolgt auf der Grundlage von Latenzen. Es wird zunächst initial nach einer Platzierung von Komponenten im virtuellen Latenzraum gesucht. Ist diese gefunden, erfolgt die Abbildung auf das physische Netzwerk. Mit Hilfe relaxierender Kräfte erfolgt anschließend die Anpassung der Anwendungsplatzierung. Erfolgt die erste Platzierung noch zentral, sind weitere Verbesserungen auch mit lokalem Wissen möglich. Die unterschiedlichen Latenzen zwischen aktueller und einer möglichen veränderten Platzierung erzeugen Kräfte, die auf die Anwendungsplatzierung einwirken. Eine genaue Beschreibung der initialen und laufenden Platzierungsanpassung nach O'Keeffe findet sich innerhalb des Abschnitts 5.2.2 inklusive der Beschreibung zum virtuellen Latenzraum als Grundlage der Platzierungsbestimmung im Rahmen der Betrachtung über die verteilte Platzierung von Anwendungskomponenten in einem Publish/Subscribe-basierten System in dieser Arbeit.

Im Umfeld intelligenter Umgebungen stechen insbesondere die Arbeiten von Herrmann et al. [98, 99] und deren Vorarbeiten zu Ad hoc-Service Grids [100] in [96, 97] heraus, bei denen sich Dienste selbstorganisiert, mit lokalem Wissen auf der Grundlage der Spring Relaxation-Heuristik verteilen. Die in Herrmann et al. vorgestellte Idee bewegt sich im Bereich intelligenter Umgebungen, bei denen Nutzern über mobile Geräte stets lokale Dienste in der Nähe mit ausreichender Güte zur Verfügung gestellt werden. Beispiele für solche Dienste finden sich in der Informationsbereitstellung in Einkaufszentren oder auch in Baustellenbereichen. Die Idee hinter der Arbeit ist, solche Dienste mittels der Ad hoc-Service Grids bedarfsgerecht bereitzustellen, die Dienste also nur dann verfügbar zu machen, wenn ein Bedarf dafür vorhanden ist. Dazu werden die Dienste an den Orten, an denen Bedarf besteht, repliziert bzw. migriert und aufgelöst, sobald sie nicht mehr benötigt werden. Für die Steuerung der Aktionen werden lokal Regeln genutzt, welche global gesehen Kosteneffizienz durch die Einsparung von Kommunikationsaufwand erreichen. Dienste werden in der Nähe der Konsumenten ausgeführt und Kommunikationsaufwand eingespart. Dazu wird regelmäßig überprüft, welcher Bedarf besteht, sowie ob und welche Veränderungen nötig sind. Liegt die Anzahl der Dienstekonsumenten lokal unter einem Schwellwert, wird der Dienst beendet. Erhält ein vorhandener Dienst Anfragen aus unterschiedlicher Richtung, erfolgt dessen Replikation. Empfängt ein Dienst signifi-

kant viele Anfragen aus einer Richtung, die diese Anfragerichtung dominiert, wird der Dienst in Richtung der dominierenden Anfrage verschoben. Eine weitere, feingliedrige Anpassung der Diensteplatzierung findet allerdings nicht statt, die Dienste werden als atomare Einheit betrachtet.

3.3 Anwendungsmodell

Anwendungen in intelligenten Umgebungen sind vielfältig. Sie reichen von der einfachen Darstellung von Informationen über kontextbezogene Informationssammlungen und Darstellungen bis hin zu kontextabhängigen Steuerungsanwendungen. Im folgenden Abschnitt wird beschrieben, welchen Anforderungen ein Anwendungsmodell genügen muss. Weiterhin wird das Anwendungsmodell als Grundlage für das in dieser Arbeit vorgestellte Platzierungskonzept entwickelt.

3.3.1 Anforderungen an Anwendungen und Ausführungsorte

Um Anwendungen in intelligenten Umgebungen sinnvoll im Sinne eines Platzierungsziels zu platzieren, werden sowohl Informationen über die Anwendungen, als auch über die möglichen ausführenden Geräte benötigt. Mit Hilfe dieser Informationen werden die Anwendungen zu unterschiedlichen Ausführungsorten mit bestimmten Ressourcen zugeordnet. Sind die Eigenschaften von Geräten als mögliche Ausführungsorte bekannt, müssen auch Anwendungen und ausführende Geräte bestimmte Anforderungen erfüllen, soll die Verteilung von Anwendungen durch einen Dienst der Middleware möglich sein. Der nun folgende Absatz dient als Vorarbeit zur Definition eines eigenen Anwendungsmodells zunächst der Klärung, was ein Anwendungsmodell leisten muss.

1. Anforderung: Selbstbewusstsein, Selbsterkennung und Beschreibung
Die Eigenschaften bereits ausgeführter Anwendungen bezüglich ihrer Anforderungen an Ressourcen müssen vom bisher beobachteten Verhalten abgeleitet werden können. Dazu muss ein Mechanismus vorhanden sein, der eine Beurteilung des Ressourcenverbrauchs ermöglicht und die Eigenschaften der Anwendung bekannt macht. Um die Anwendungsanforderungen bereits vor ihrer (erstmaligen) Ausführung beurteilen zu können, werden Notationen benötigt, welche im Vorfeld der Anwendungsausführung die Anforderungen und Eigenschaften der Anwendungen bezüglich der Platzierung beschreiben. Es obliegt an dieser Stelle dem Entwickler, für entsprechende Annotationen bzw. Deploymentdeskriptoren zu sorgen.
2. Anforderung: Serialisierung und Deserialisierung von Anwendungen
Die Anpassung der Anwendungsverteilung basiert hauptsächlich auf der Migration von Anwendungen hin zu einem besser geeigneten Ausführungsort. Damit Anwendungen jedoch überhaupt verschoben werden können, muss ihr Zustand

serialisierbar sein. Anschließend müssen sie in serialisiertem Zustand zum Zielausführungsort gebracht werden, um dort deserialisiert werden zu können. Erst danach ist eine weitere Ausführung möglich.

3. Anforderung: Einsatz einer einheitlichen Ausführungsplattform

Um eine nahtlose Weiterführung einer Anwendung zu ermöglichen, muss die Ausführungsplattform am Zielort dazu geeignet sein, also prinzipiell der Ausführungsplattform des Ursprungsorts entsprechen, sodass keine tiefgreifenden, sondern lediglich kleinere Anpassungen notwendig sind. Unter solch kleinere Anpassungen fällt u.a. das Nachladen einzelner Programmpakete.

4. Anforderung: Komponentenbasiertes Anwendungsdesign

Durch die Erfüllung der Anforderung 1 ist das Verhalten einer Anwendung bezüglich der geforderten und der tatsächlich genutzten Ressourcen ausreichend beschrieben, um sie vor ihrer erstmaligen Ausführung und auch während ihrer Laufzeit ausreichend gut einschätzen und platzieren zu können. Allerdings können Anwendungen in ihren Anforderungen so umfangreich sein, dass ein kompaktes Deployment als Ganzes oder eine notwendige Migration nicht möglich sind. In solchen Fällen ist es sinnvoll, Anwendungen in Teilkomponenten zu zerlegen (oder einzelne Teilkomponenten zusammenzuführen) und als solche separat zu deployen und auszuführen. Zur Realisierung dieser Anforderung ist der komponentenbasierte Anwendungsaufbau das Mittel der Wahl. Alle Anwendungskomponenten müssen die in diesem Abschnitt genannten Anforderungen erfüllen, vor allem die nach Selbstbewusstsein und Selbsterkennung des Verhaltens und der Anforderungen. Eine Zerlegung von Anwendungen in Teilkomponenten ist zudem durch die Anwendungen selbst unter Einhaltung vorgegebener Schnittstellen selbstständig durchzuführen. Für die Aufspaltung, die Zusammenführung oder das Klonen von Anwendungen bzw. der Anwendungskomponenten werden entsprechende Methoden benötigt, welche zwar anhand von (einheitlichen) Schnittstellen spezifiziert, aber durch den Anwendungsentwickler für die Anwendungskomponente bereitgestellt werden müssen.

5. Anforderung: Methoden zur Erstellung eines sicheren Zustands und zur Weiterführung an verändertem Ausführungsort auf Anwendungsebene

Die Aufspaltung, Replikation und Migration von Anwendungen sind gemäß der o.g. Anforderungen nur möglich, wenn spezifizierte Schnittstellen ausprogrammiert, eine Serialisierung und Deserialisierung vorgesehen und eine einheitliche Ausführungsumgebung vorhanden sind. Weiterhin muss jedoch gegeben sein, dass Anwendungen, bevor sie serialisiert werden, in einen sicheren Zustand gebracht werden. Das bedeutet, dass Anwendungen laufende Prozesse erst beenden. Wie die Umsetzung der Serialisierung und Deserialisierung ist es auch hier Aufgabe des Anwendungsentwicklers, vorher spezifizierte Schnittstellen auszuprogrammieren.

6. Anforderung: Synchronisierung von Uhren

Während der Weiterleitung von Nachrichten in einem Netzwerk kommen diese möglicherweise nicht in der ursprünglichen Reihenfolge beim Empfänger an. Damit die Reihenfolge von Nachrichten trotzdem gewährleistet werden kann, kommen i.d.R. zusätzliche Techniken zum Einsatz, um zumindest eine relati-

ve Ordnung von Nachrichten sicherzustellen. Bekannte Techniken nutzen dafür eine zusätzliche Nummerierung oder Zeitstempel. Werden Nachrichten nur von einer Instanz emittiert, kann die Reihenfolge der Nachrichten beim Empfänger wieder geordnet zusammengestellt werden. Werden jedoch möglicherweise Konflikte bei Nachrichtenempfängern durch unterschiedliche Nachrichtenquellen bei einem Empfänger ausgelöst, reichen die Nummerierung oder die Beimischung eines Zeitstempels des nachrichtenemittierenden Geräts bzw. der Anwendung nicht aus. Stattdessen wird ein synchronisierter Zeitstempel genutzt, der jede erzeugte Nachricht mit einem Zeitstempel versieht. In dieser Arbeit wird davon ausgegangen, dass die Uhren der Geräte regelmäßig synchronisiert werden und eventuell entstandene Abweichungen kurzfristig ausgeglichen werden.
7. Anforderung: Quantifizierungsmöglichkeit von Anforderungen und Ressourcen
Die Bewertung von Anforderungen und verfügbaren Ressourcen ermöglicht einen Abgleich beider Aspekte und eine Bewertung, ob und welche Platzierung günstiger ist als alternative Varianten. Dazu wird ein einheitlicher Bewertungsmaßstab für einzelne Ressourcen bzw. Gruppen von Ressourcen benötigt. Dieser ist verbindlich für das gesamte Netzwerk und gilt sowohl für die Anwendungsanforderungsbeschreibung als auch für die Beschreibung von Ressourcen der Geräte.

3.3.2 Anwendungen und Anwendungskomponenten

Aus den Anforderungen 4 und 5 (wobei auf 4 ein besonderes Gewicht liegt) geht hervor, dass Anwendungen bestimmte Schnittstellen implementieren müssen, um überhaupt flexibel in einem Gerätenetzwerk positioniert werden zu können. Doch Flexibilität auf Anwendungsebene wird, nach Anforderung 4, vor allem durch ein komponentenbasiertes Anwendungsdesign erreicht. In Fortführung von Anforderung 1 (nach Selbstbewusstsein und Selbsterkenntnis) und analog zu den Anforderungen 2 und 5 müssen die Anwendungen selbst gemäß spezifizierter Schnittstellen und nach dem Ausprogrammieren durch den Entwickler entscheiden, wie sie in Komponenten zerlegt werden können, wie und welche Variablen aufgeteilt werden und über welche Kanäle/Datentypen kommuniziert wird.

Grundsätzlich kann eine Komponente aus beliebig vielen Teilkomponenten bestehen, und die Teilkomponenten selbst wiederum aus beliebig vielen weiteren Komponenten. Sind Komponenten jedoch logisch miteinander verknüpft, geschieht dies durch Operatoren, welche in ihrem Verhalten grundsätzlich von der ihnen zu Grunde liegenden Algebra abhängig sind. Auf solche miteinander verknüpften Komponenten wird noch einmal gesondert im Abschnitt 5.2.1 eingegangen.

Wichtig an dieser Stelle ist vor allem, dass Anwendungen sich nach Aufruf einer definierten und einheitlichen Schnittstelle eigenständig zerlegen. Dies ermöglicht einen einfachen Aufruf der Funktionalität durch die Middleware, ohne dass diese in die eigentliche Ausführung der Anwendung eingreift. Eine Anwendung wird von der Middleware lediglich aufgefordert, bestimmte Aktionen auszuführen. Mit Hilfe eines Rückgabewertes wird festgestellt, ob die Aktion erfolgreich war

und welche Teilkomponenten künftig im Netzwerk ausgeführt werden. Die weitere Verteilung der Teilkomponenten übernehmen die im Folgenden (insbesondere in Abschnitt 3.5) genannten Basisoperationen der laufenden Platzierung. Die Verschiebung von Anwendungen wird durch die Middleware und damit von außen aufgerufen, während die Umsetzung durch die Anwendung selbst durchgeführt wird. Diese Umsetzung unterscheidet sich von dem Konzept, welches bspw. von mobilen Agenten genutzt wird. Diese sind sofort und damit nach jeder Instruktion verschiebbar, auch unter Abbruch sämtlicher aktiver Kontrollflüsse. Sie können aus jedem laufenden Prozess heraus serialisiert und an anderer Stelle deserialisiert werden. Bei solch einem Konzept wird von starker Mobilität [187] gesprochen. Bei den Anwendungen, die in dem in dieser Arbeit vorgestellten Konzept betrachtet werden, steht die Verschiebung des Zustands der Anwendung im Vordergrund. Die Anwendung selbst steuert ihre eigene Serialisierung und hat dabei ausreichend Zeit, sich in einen serialisierbaren Zustand ohne aktive Aufrufe zu begeben. Nach erfolgter Verschiebung wird die Anwendung von dem gesicherten Zustand heraus weitergeführt, in diesem Zusammenhang wird von schwacher Mobilität gesprochen [187].

3.4 Automatisches Anwendungsdeployment

Das Deployment von Anwendungen im engeren Sinne umfasst alle Prozesse die nötig sind, um Anwendungen auf einem Gerät ausführbar zu machen und bereitzuhalten. Es beginnt mit der Installationsvorbereitung und verläuft über die Ausführung bis hin zur Aktualisierung und Verwaltung. Diese Betrachtung ist nicht Bestandteil dieser Arbeit, da davon ausgegangen wird, dass eine einheitliche Ausführungsumgebung vorhanden ist, auf der die Anwendungen ausgeführt werden können. Allerdings besitzen die Geräte, trotz einheitlicher Ausführungsumgebung, Unterschiede bezüglich ihrer Ausstattung. Diese befähigen sie in unterschiedlicher Weise die Anwendung auszuführen. Der eigentliche Vorgang der Installation inkl. der Installationsvorbereitung, der Ausführbarmachung von Anwendungscode, der Registrierung und Versorgung mit Aktualisierungen steht zwar nicht im Fokus des Deployments im Sinne des in dieser Arbeit vorgestellten Konzepts, wird aber zur Einordnung in das Gesamtbild in Abschnitt 3.4.2 erläutert. Das Hauptaugenmerk des Deployments in dieser Arbeit liegt stattdessen auf der bedarfsgerechten Bestimmung des Ausführungsortes einer Anwendung bzw. Anwendungskomponente. Diesem Thema widmet sich der Hauptteil dieses Kapitels.

Das Deployment von Anwendungen spielt sowohl bei der initialen, als auch bei der laufenden Platzierung von Anwendungen eine Rolle. Während bei der initialen Platzierung noch keine Anwendungen platziert und ausgeführt wurden, ist bei der laufenden Platzierung die Anwendung bereits ausgeführt worden. Damit ist jedoch das Deployment sowohl im engeren Sinn wie auch im Sinne des hier vorgestellten Konzepts nicht abgearbeitet. Wie aus dem Abschnitt 3.5.2 ersichtlich ist, wird insbesondere im Zusammenhang mit der Dekomposition, der

Migration und der Replikation erneut ein Deployment von Anwendungskomponenten durchgeführt. Dabei werden entweder die gesamte Anwendung (Migration, Replikation) bzw. einzelne Anwendungskomponenten (Dekomposition mit Migration) auf einem Gerät deployt.

3.4.1 Ziel des Anwendungsdeployments

Ziel des Anwendungsdeployments ist die möglichst optimale und bedarfsgerechte Verteilung von Anwendungskomponenten in einem Netzwerk von miteinander verbundenen Geräten. Auch wenn konzeptionell zwischen initialer Platzierung und laufender Platzierungsanpassung unterschieden wird, ist die Zielstellung beider Deploymentaspekte grundsätzlich gleich und wird im Folgenden erläutert.

Aufgrund der bereits in der Einleitung der Arbeit beschriebenen Eigenheiten (NP-schweres Problem selbst in einem statischen Umfeld, dem System inhärente Dynamik) ist eine Optimierung im klassischen Sinne aufgrund des damit verbundenen Aufwands schwerlich möglich. Da heterogene Umgebungen zudem häufig durch eine dezentrale Organisation geprägt sind, existiert auch kein Gerät, welches alle verfügbaren und notwendigen Daten für eine Optimierung sammelt, eine optimierte Verteilung ermittelt und diese anschließend umsetzt. Erschwerend kommt hinzu, dass bei einem zentralisierten Vorgehen alle benötigten Informationen zum entscheidungsbefugten Gerät transportiert werden müssen, was eine Überlastung einzelner Kommunikationskanäle nach sich ziehen könnte. Daher wird mit Hilfe der initialen Platzierung zunächst keine optimale, sondern zunächst einmal eine *gültige*, d.h. bedarfsgerechte Anwendungsplatzierung gesucht. Bedarfsgerecht bedeutet hier, dass eine Platzierung die Anforderungen von Anwendungen ebenso beachtet wie die verfügbaren Ressourcen seitens der ausführenden Geräte. Nach einer gefundenen gültigen, wenn auch nicht zwangsläufig optimalen, Platzierung von Anwendungskomponenten hat die daran anschließende laufende Platzierungsanpassung zwei Aufgaben. Beide Aufgaben dienen der Optimierung der Anwendungsplatzierung, wobei einerseits die weitere Optimierung von einer gültigen Platzierung in Richtung einer optimalen Platzierung, andererseits aber auch die Beibehaltung einer guten bzw. optimalen Platzierung unter sich ändernden Bedingungen zu finden sind. Zur laufenden Platzierung werden die in Abschnitt 3.5.2 genannten Basisoperationen genutzt.

Für die Zuordnung von Anwendungen zu den sie ausführenden Geräten sind sowohl während der initialen, als auch während der laufenden Platzierung einige Voraussetzungen zu erfüllen, welche in der folgenden Übersicht genannt werden.

- Spezifizierbare Anforderungen.
  Die Anforderungen der Anwendungen sind beschreibbar und ausreichend beschrieben.
- Spezifizierte Eigenschaften (möglicher) Ausführungsorte
  Die Geräte, welche Anwendungen ausführen können, sind sich ihrer Eigenschaften und Kapazitäten bewusst und können diese beschreiben.

- Austausch über Anforderungen und Ressourcen
  Es existiert ein Mechanismus, mit dem Informationen über Anwendungsanforderungen und verfügbare Ressourcen ausgetauscht werden können.

Wichtig für die initiale und laufende Platzierung ist also, dass sich Anwendungen wie auch ausführende Geräte ihrer Anforderungen respektive Fähigkeiten bewusst sind. Während Geräte bspw. mit Hilfe von Monitoringdiensten ihre freien und ausgelasteten Kapazitäten erkennen können, ist dies Anwendungen indirekt über das Gerätemonitoring möglich, indem die Kapazitätsauslastung zwischen unterschiedlichen Zuständen verglichen wird. Die Verbrauchsmessung über das Monitoring ist jedoch nur möglich, wenn die Anwendung ausgeführt und der Ressourcenverbrauch am ausführenden Gerät auch einer spezifischen Anwendung zugeordnet werden kann. Ist dies nicht möglich, kann der spezifische Ressourcenverbrauch prinzipiell auch über eine Schätzung ermittelt werden. Dies ist regelmäßig der Fall, wenn noch keine Verbrauchswerte vorliegen, z.B. wenn eine Anwendung bisher noch nicht ausgeführt wurde. In diesem Fall müssen der Anwendung zusätzliche Informationen beigefügt werden, welche eine Ressourcenverbrauchsabschätzung ermöglichen. Die für die Ressourcenverbrauchsabschätzung nötigen Informationen sind dazu zusätzlich vom Entwickler der Anwendung bereitzustellen. Während also bei der laufenden Platzierung auf Ergebnisse von Monitoring, auch in Kombination mit Zusatzinformationen zurückgegriffen werden kann, werden bei der initialen Platzierung Informationen über die Anwendungen zwingend benötigt.

3.4.2 Deployment im engeren Sinn

Nach der Bestimmung der vorhandenen und benötigten Ressourcen erfolgt die Zuweisung der Anwendungen zu den sie ausführenden Geräten. Daran schließt sich direkt die Installation der Anwendungen an. Dazu gehören die im Folgenden genannten Aspekte:

- Verteilung des Anwendungsquelltextes bzw. Zwischencodes
  Hier wird ein Mechanismus benötigt, der den gewünschten Anwendungscode zu einem bestimmten Ausführungsort hin verteilt, sodass dieser dort installiert und ausgeführt werden kann. Bei dem in dieser Arbeit vorgestellten Konzept zur initialen und laufenden Verteilung von Anwendungskomponenten wird allerdings von einer einheitlichen Ausführungsumgebung (Java) ausgegangen. Damit muss nicht der Anwendungsquelltext übergeben und vor Ort kompiliert werden, sondern es reicht, den Anwendungsquelltext einmalig zu kompilieren und im Folgenden als Zwischencode (Java-Bytecode) zu verteilen.
- Verfügbarkeit einer Installationsroutine
  Mit Hilfe einer Installationsroutine wird die Installation im engeren Sinn vor Ort durchgeführt.

- Bereitstellung einer Ausführungsumgebung
  Die Ausführung einer Anwendung auf verschiedenen Geräten ist nur dann möglich, wenn eine Ausführungsplattform existiert, die auf allen in Frage kommenden Geräten existiert und von der zu Grunde liegenden Hardware abstrahiert. Ohne die Gleichheit von Ausführungsumgebungen ist eine Verschiebung von Anwendungen und Anwendungskomponenten nicht ohne weiteres möglich. Ein Beispiel für solche Ausführungsumgebungen ist bspw. die *JavaVirtualMachine* (JavaVM).

Für das Konzept zur bedarfsgerechten Verteilung der Anwendungskomponenten wird auf eine Grundstruktur zurückgegriffen, welche die drei o.g. Aspekte bereits umgesetzt hat. Die Neuinstanziierung von Anwendungen bzw. Anwendungskomponenten sowie ihre Verschiebung werden so technisch ermöglicht.

3.4.3 Initiale Anwendungszuweisung

Beim Suchen und Finden einer gültigen initialen Platzierung gelten für den hier vorgestellten Ansatz einige Einschränkungen. Das Anwendungsumfeld ist ein heterogenes Netzwerk von Knoten. Die Heterogenität ist dabei so zu verstehen, dass auf den angeschlossenen Geräten grundsätzlich die gleiche Ausführungsumgebung vorgehalten wird und Anwendungen prinzipiell auf jedem Gerät ausführbar sind. Trotzdem unterscheiden sich die Geräte untereinander, sie sind heterogen. Bei der Betrachtung von Heterogenität wird zwischen zwei Ebenen unterschieden. Es unterscheiden sich die Geräte hinsichtlich (1) ihrer angeschlossenen Komponenten. So besitzen einige Geräte Sensoren zur Detektion unterschiedlicher (Umwelt-) Zustände oder Schnittstellen zu bestimmten Ressourcen, z.B. zu einer Datenbank. Die gleiche Ausführungsumgebung (hier die JavaVM) ist dennoch gegeben. Von Heterogenität auf der zweiten Ebene ist im Zusammenhang mit dieser Arbeit die Rede, wenn die Geräte zwar ebenfalls über eine einheitliche Ausführungsumgebung, jedoch über (2) unterschiedlich große einzelne Ressourcengruppen verfügen. Das bedeutet, ihre Ausstattung unterscheidet sich nicht in Bezug auf das Vorhandensein von bestimmten Ressourcen wie Arbeitsspeicher, Verarbeitungskapazität und Energiespeicher, sondern bezüglich des Umfangs der jeweiligen Ressourcen.

Grundsätzlich können Anwendungen nur auf solchen Geräten ausgeführt werden, welche den Anwendungsanforderungen entsprechend ausreichende Ressourcen aufweisen. Neu zu platzierende Anwendungen sind unter Beibehaltung dieses Prinzips nicht in der Lage, bereits platzierte und ausgeführte Anwendungen zu verdrängen. Wäre dies möglich, so würde sich ein weiteres Problem aus der Klasse der NP-schweren Probleme auftun, da es auf ein klassisches Verteilungsproblem von Aufgaben auf Ressourcen zurückzuführen ist (siehe *Task Assigning Problem* [136, 137]). Dies gilt insbesondere, wenn es durch die Verdrängung von Anwendungen zu einer kaskadierenden Umverteilung kommt. Dies ist bei der in dieser Arbeit dargestellten Vorgehensweise jedoch nicht gewünscht. Kann keine

gültige Platzierung gefunden werden, wird die Anwendung, unter der Inkaufnahme damit verbundener Nachteile, nicht ausgeführt und der Nutzer informiert.

Das Konzept zur bedarfsgerechten Verwaltung sieht einen im Folgenden beschriebenen Ablauf vor: Die zu verteilenden Anwendungen werden entsprechend ihrer Anforderungen einem Gerät zugeordnet. Ein Broker, der die Rolle einer Verwaltungskomponente ausführt, steuert die Verteilung der Anwendung und ihrer Komponenten. Diese Verwaltungskomponente initiiert die bedarfsgerechte Verteilung, indem sie die beschriebenen Eigenschaften und Anforderungen der Anwendungen ausliest, die verfügbaren Ressourcen ermittelt und die Verteilung anschließend initiiert. Die Rolle des Koordinators kann grundsätzlich von jedem Broker innerhalb des Netzwerks ausgeführt werden, z.B. von dem Gerät, das mit dem Nutzer interagiert, von einem zentralen Koordinator oder einem Koordinator, der fallbasiert ausgewählt wird. Letzteres ließe sich ähnlich der Implementierung von Rendezvous-Brokern erreichen. Um einen allgemein gültigen Ansatz ohne die Abhängigkeit von Rendezvous-Brokern realisieren zu können, wird jedoch davon ausgegangen, dass der Koordinator von dem Gerät ausgeführt wird, welches mit dem Nutzer interagiert.

Voraussetzungen. Für die Verteilung und Ausführung von Anwendungen müssen die in Abschnitt 3.4 genannten Anforderungen an die Spezifizierbarkeit von Anwendungen, verfügbaren Ressourcen auf Geräteebene und Mechanismen zum Austausch von Informationen erfüllt sein. Weiterhin vorausgesetzt werden Mechanismen zur Verteilung des Java-Bytecodes, die Verfügbarkeit einer Installationsroutine und die Bereitstellung einer einheitlichen Ausführungsumgebung, letzteres für das Deployment.

Für eine bedarfsgerechte Platzierung spielt jedoch Kommunikation eine entscheidende Rolle. Wurden die bisherigen Ausführungen schon mit Beispielen aus der Welt der ereignisbasierten Kommunikation angereichert, fußt das Verteilungskonzept sowohl für den initialen Platzierungsfall, aber auch für die laufende Platzierungsanpassung auf einer Publish/Subscribe-basierten Kommunikation und Interaktion. Die Verteilungsmechanismen benötigen eine möglichst präzise Beschreibung von Anforderungen und verfügbaren Ressourcen, welche durch Publish/Subscribe-Primitive ausgetauscht werden.

Anwendungszuweisung. Grundsätzlich obliegt es dem Anwendungsentwickler, für die Bereitstellung der Anwendungsanforderungen in einer Form zu sorgen, die ihre Verbreitung mittels der Primitive von Publish/Subscribe ermöglicht. Wenn die benötigten Informationen nicht in geeigneter Form vorliegen, müssen sie zunächst in ein entsprechendes Format aufgearbeitet werden. Die angesprochenen Primitive sind die Arten von Nachrichten im Publish/Subscribe-Netzwerk, also Ankündigungen, Subskriptionen und Notifikationen, welche vom Notifikationsdienst entsprechend der eingesetzten Strategien verteilt werden. Somit entscheiden die Strategien der Weiterleitung, neben der Anzahl von Geräten,

wesentlich über die Anzahl der im System vorhandenen Nachrichten. Publish/Subscribe bietet die Möglichkeit, mit Hilfe von Ankündigungen und der Publikation von Interessen durch Subskriptionen die Weiterleitung nicht benötigter Nachrichten zu begrenzen. So können durch die Veröffentlichung von Interessen die Anforderungen von Anwendungen ebenso bekannt gemacht werden wie die verfügbaren Ressourcen von Geräten. Sowohl die Anwendungen, als auch die Geräte benötigen jedoch eine eindeutige Bezeichnung für die Zuordnung von Anwendungen und Geräten. Dank einer eindeutigen Zuordnung sind die Koordinatoren in der Lage, die Geräte zu erkennen, welche wiederum dazu im Stande sind, die zu platzierende Anwendung auszuführen. Nun könnten Geräte einerseits ständig ihre verfügbaren Ressourcen publizieren, was bedeuten würde, dass alle Empfänger der Nachrichten das Wissen über die verfügbaren Kapazitäten haben, jeder Empfänger im Netzwerk über globales Wissen verfügt. Aufgrund der Dynamik des Systems und der Vielzahl möglicher Teilnehmer wäre dabei allerdings eine Vielzahl von Nachrichten vonnöten, auch die Frage nach der Aktualität von Informationen würde sich stellen. Müssen nicht laufend neue Anwendungen verteilt werden, ist die Mehrzahl der Nachrichten jedoch überflüssig, würde die Kommunikationsinfrastruktur belasten und damit Kosten verursachen. Stattdessen informieren die Geräte nur anlassbezogen auf Platzierungsanfragen über ihren Zustand, sprich über verfügbare Ressourcen.

3.4.4 Platzierungsmodell

Die Platzierung einer Anwendung, d.h. das Finden einer gültigen Platzierung ist dann möglich, wenn sowohl Anwendungsanforderungen als auch verfügbare Ressourcen in Übereinstimmung gebracht werden. Das Vorgehen zum Suchen und Finden einer gültigen Anwendungsplatzierung erfolgt mehrstufig. So wird grundsätzlich zunächst versucht, die Anwendung in Gänze zu platzieren. Ist dies nicht möglich, wird die Anwendung, soweit möglich, in ihre nächst niedrigere Ebene und die dortigen Teilkomponenten aufgebrochen. Nun wird abermals versucht, für die Anwendung, d.h. für alle Anwendungskomponenten, eine gültige Verteilung zu finden. Ist dies jedoch wieder nicht möglich, erfolgen die Aufspaltung in weitere Teilkomponenten und der Versuch, diese zu verteilen und auszuführen. Dieser Zyklus wird dabei solange ausgeführt, bis entweder alle Teilkomponenten verteilt sind oder keine weitere Aufspaltung mehr möglich ist. Ist auch auf der letzten Komponentenebene keine Platzierung möglich, wird die Anwendungsplatzierung abgebrochen. Es erfolgt eine entsprechende Meldung an den Initiator des Anwendungsdeployments, den Nutzer.

Der Ablauf der initialen, mehrschichtigen Platzierung ist in jeder Ebene grundsätzlich gleich, werden doch auf jeder Ebene die erwarteten bzw. ermittelten Anforderungen einer Anwendung oder einer Anwendungskomponente von einem *Koordinator* ausgelesen und verteilt. In der Publish/Subscribe-Umgebung lassen sich die Anforderungen, wie auch die verfügbaren Ressourcen, mit Hilfe von Name/Wert-Paaren und in ein Ereignis verpackt der Publish/Subscribe-

Kommunikationsinfrastruktur übergeben. Die Quantifizierung der Anforderungen und verfügbaren Ressourcen seitens der Geräte erfolgt durch die Codierung von konkreten Werten und ihren Bezeichnungen als Name/Wert-Paare, sowohl für notwendige, als auch für hinreichende Bedingungen sowie verfügbare Ressourcen. Eine eindeutige Zuordnung zwischen den geforderten Anforderungen und verfügbaren Ressourcen benötigt eine eindeutige sowie allgemein gültige und akzeptierte Bezeichnung, welche als Namensteil der Name/Wert-Paare im Netzwerk bekannt sein muss. Dies wird durch die Broker sichergestellt, indem sie die Anforderungen und die dazugehörigen Antworten mit eindeutigen *IDs* versehen und versenden. Jedes Gerät im Netzwerk wird durch eine eindeutige, globale ID im Netzwerk repräsentiert und ordnet den durch sie verwalteten Anwendungen und Anwendungskomponenten sowie der zu verteilenden Anwendung bzw. Anwendungskomponenten lokal eindeutige IDs zu. Ebenfalls eindeutig anhand der IDs sind die Anforderungen der jeweiligen Anwendungen unterscheidbar, sodass sich schrittweise eine eindeutige Gesamt-ID aus Gerät bzw. Koordinator, Anwendung und Anforderung ergibt. So senden die Koordinatoren mit ihrer eigenen ID die spezifischen Anwendungs-IDs bzw. zuzüglich der Komponenten-ID mit den veröffentlichten Anforderungen als Name/Wert-Paare mit. Zur Auswahl eines oder mehrerer ausführender Geräte senden auch die Geräte selbst auf die Antwort der Ressourcenanfrage ihre eigene und eindeutige Geräte-ID als Name/Wert-Paar mit.

Grundsätzlicher Ablauf. Der Koordinator erhält die Aufgabe, die ihm zugewiesene Anwendung im Publish/Subscribe-Netzwerk zu platzieren. Er erhält dazu die Anwendung sowie die zur Ausführung dieser notwendigen Ressourcen. Der Koordinator sendet zunächst eine spezifische Ressourcenanforderung in das Netzwerk. Dort werden die geforderten Anforderungen mit den verfügbaren Ressourcen verglichen und bilden einen Vergleichswert („Matchingwert“), dieser wird für die Anwendung an den Koordinator zurück übermittelt. Gleichzeitig blockiert das Gerät die benötigten Ressourcen. Der Koordinator ermittelt aus den eingehenden Nachrichten das Gerät, welches am besten für die Ausführung der Anwendung geeignet ist und überträgt an dieses die Ausführung der Anwendung, welches nun die Rolle des Anwendungsausführenden einnimmt. Die anderen Geräte erhalten einen Widerruf der vormals benötigten Ressourcen, sodass die Geräte die reservierten Ressourcen freigeben können. Ist hingegen die Rolle des Anwendungsausführenden nicht zuweisbar, widerruft der Koordinator die ursprüngliche Ressourcenanfrage, die Geräte mit reservierten Ressourcen geben diese frei, die Verteilung startet von neuem, diesmal jedoch mit den Anwendungsteilkomponenten der ursprünglichen Anwendung. Die Zerlegung wird solange durchgeführt, bis entweder alle Teilkomponenten vom Koordinator verteilt wurden bzw. die letzte Teilkomponentenstufe erfolglos durchlaufen wurde.

Wie aus dieser kurzen Beschreibung zu entnehmen ist, offeriert der Koordinator zunächst die zu platzierende Anwendung inklusive der benötigten Ressourcen. Die Geräte nehmen die Informationen zu den Anforderungen auf, vergleichen sie mit den eigenen vorhandenen Ressourcen und berechnen den individuellen

Matchingwert. Sie erstellen und senden eine individuelle Antwort, inklusive der Angabe ihrer ID. Aus den eingehenden Matchingwerten ermittelt der Koordinator die Geräte, welche den Ausführungszuschlag bekommen. Diese erhalten eine individuelle Nachricht, auf die sie sich im Vorfeld subskribieren. Genauso erhalten alle anderen Geräte eine Nachricht, dass sie nicht ausgewählt wurden. Anschließend bestätigt eine weitere Nachricht vom ausführenden Gerät an den Koordinator, dass das Anwendungsdeployment erfolgreich durchgeführt wurde.

Beim Blick auf die Kommunikationsströme fällt auf, dass sowohl Kommunikation von einem Sender an mehrere Empfänger (one-to-many) als auch direkte Kommunikation zwischen zwei Kommunikationspartnern (one-to-one) genutzt werden. Da die Publish/Subscribe-basierte Kommunikation grundsätzlich entkoppelt, insbesondere von Sendern und Empfängern agiert, wird von Publish/Subscribe keine Kommunikation zwischen zwei Kommunikationspartnern unterstützt. Durch die Gestaltung von Ankündigungen, Subskriptionen und Publikationen lässt sich jedoch auch bidirektionale Kommunikation durch Publish/Subscribe-Primitive abbilden.

Initiale Platzierung mit Publish/Subscribe-Primitiven. Die bei der Ausführung der initialen Platzierung beteiligten Rollen sind der Koordinator, der aus der Menge der verfügbaren Broker den/die ausführende/n Broker (weitere Rolle) auswählt und damit auch die Broker bestimmt, die nicht zum Zuge kommen (dritte Rolle). Einen Überblick über die möglichen Rollen zeigt Abbildung 3.3.

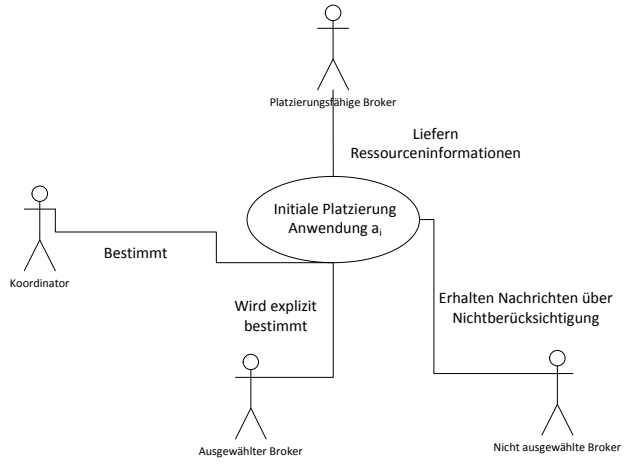


Abbildung 3.3: UML-Anwendungsfalldiagramm zur initialen Platzierung

Diese Rollen beziehen sich spezifisch nur auf eine zu platzierende Anwendung. In Bezug auf die Platzierung und Ausführung weiterer Anwendungen finden sich sowohl ausführende, als auch nicht ausführende Broker des Netzwerks in der Rolle der möglichen Ausführungsorte wieder.

Die Rollen selbst sind hierarchisch gegliedert, alle Rollen werden durch Broker des Netzwerks (selbst eine Rolle) ausgeführt und stammen von diesen ab. Die Rolle des Koordinators unterscheidet sich dabei von den Brokern, welche Anwendungen grundsätzlich ausführen können (Koordination vs. Ausführung). Bei diesen, hier als „Platzierungsfähige Broker“ bezeichnet, wird weiterhin zwischen den Brokern unterschieden, welche die Anwendung ausführen und denen, die dazu nicht ausgewählt wurden. Der Unterschied zwischen den Netzwerkbrokern und den platzierungsfähigen Brokern besteht darin, dass zweitere überhaupt in der Lage sind, eine bestimmte Anwendung auszuführen und dies durch Abgabe dieser Informationen auch im Rahmen der initialen Platzierung kundtun. Die Gliederung der Rollen (Akteure) und deren Zusammenhänge finden sich in der Abbildung 3.4.

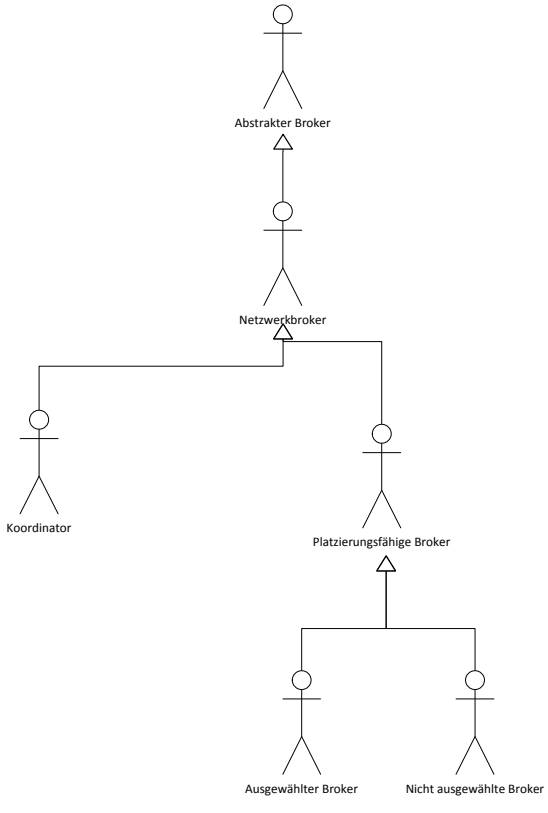


Abbildung 3.4: Akteure und Rollen der initialen Platzierung

Der Ablauf der initialen Platzierung findet sich in der Abbildung 3.5 in Form eines UML-Ablaufdiagramms. In diesem werden die Entscheidungen und Tätigkeiten genannt, welche im Zuge der initialen Platzierung nötig sind.

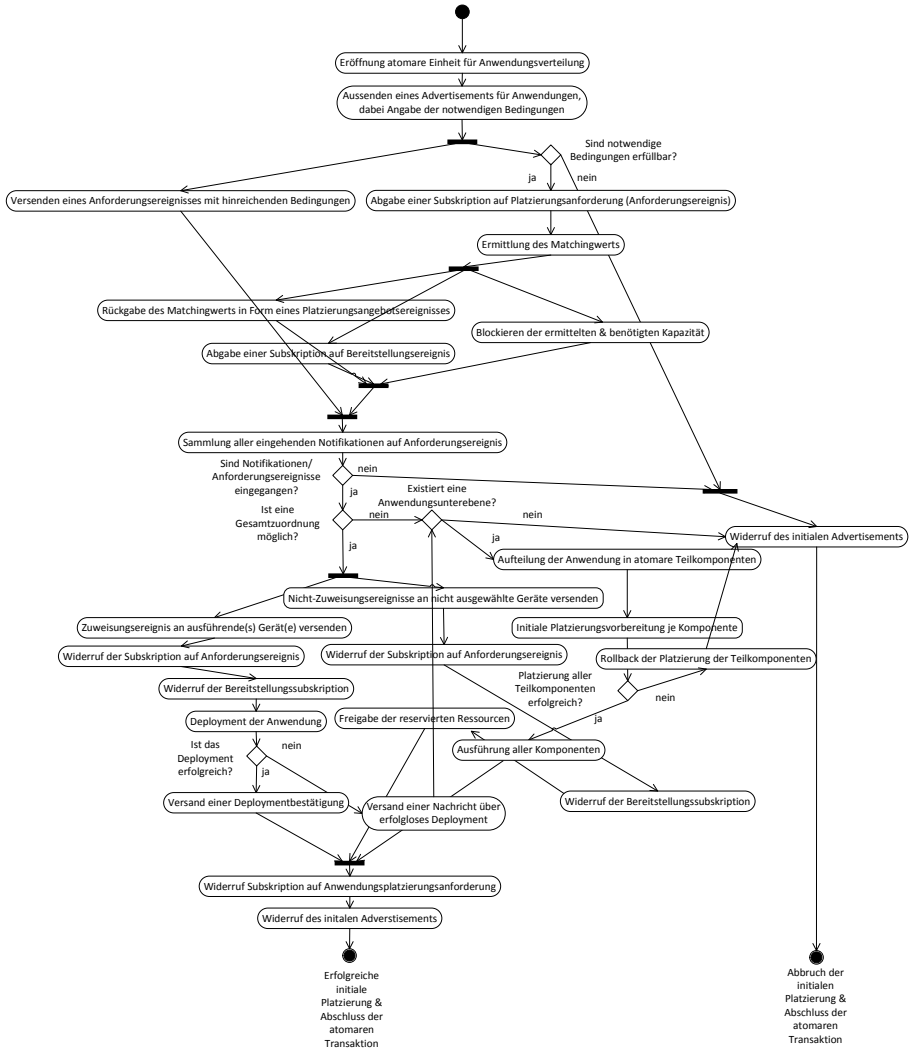


Abbildung 3.5: UML-Ablaufdiagramm zur initialen Platzierung

Falls eine Aufteilung der zu platzierenden Anwendung in Teilkomponenten notwendig wird und möglich ist, sind weitere Tätigkeiten und Entscheidungen nötig („Initiale Platzierungsvorbereitung je Komponente“). Bei dieser Tätigkeit wird eine vollständige Liste aller Teilkomponenten erzeugt. Der Ablauf der Aufspaltung und der Platzierungsversuche orientiert sich am bisherigen Vorgehen zur initialen Platzierung der Gesamtkomponente. Dies ist im Folgenden als tabellarische Darstellung der initialen Platzierung beschrieben. Die Darstellung zeigt außerdem die Abbildung der initialen Platzierung auf Primitive von Publish/Subscribe. Dabei wird auf Seiten der ausführenden bzw. ausgewählten und nichtausgewählten Broker vereinheitlicht von den platzierungsfähigen Brokern gesprochen, die weitere Einteilung ergibt sich aus der textuellen Beschreibung.

	Koordinator	**Platzierungsfähige Broker**
1	eröffnet atomare Einheit für Anwendung auf oberster Ebene	
2	Aussenden eines Advertisements für Anwendung unter Angabe der notwendigen Bedingungen	
3		Wenn notwendige Bedingungen erfüllt: Erstellen einer Subskription auf (2)
4	Versenden eines Anforderungsereignisses mit den hinreichenden Bedingungen	
5	gleichzeitig mit (4) Abgabe einer Subskription auf Platzierungsanforderungsangebot	
6		Ermittlung des Matchingwerts auf der Grundlage der formulierten Anforderungen aus (2) und der verfügbaren Ressourcen
7		Rückgabe des Matchingwertes
8		Abgabe einer Subskription für Bereitstellungsereignis
9		Blockieren der ermittelten Kapazität
10	Sammlung der eingehenden Notifikationen auf (4)	

Wenn eine Gesamtzuordnung möglich ist:

11	Zuweisungsereignis an ausführende(s) Gerät(e) versenden	
12	Nicht-Zuweisungsereignisse an nicht ausgewählte Gerät(e) versenden	
13		Empfang des Ausführungsereignisses
14		Widerruf der Bereitstellungsereignissubskription
15	Widerruf Subskription auf Anwendungsplatzierungsanforderung (5)	
16	Widerruf des Advertisements (2)	
17	Abschluss der atomaren Transaktion	

Wenn keine Zuordnung möglich ist und die Anwendung nicht in Anwendungskomponenten aufgebrochen werden kann, ist die Verteilung gescheitert. (Fall a)

11a	Meldung der gescheiterten Platzierung an Anwender versenden	
12a	Nicht-Zuweisungsereignisse an alle nicht ausgewählten Gerät(e) versenden	
13a	Widerruf der Bereitstellungsereignissubskription	
14a	Widerruf Subskription auf Anwendungsplatzierungsanforderung (5)	
15a	Widerruf des Advertisements (2)	
16a	Abschluss der atomaren Transaktion	

Ist hingegen (zunächst) keine Zuordnung möglich und die Anwendung in mindestens eine weitere Unterebene mit mehreren Teilanwendungen zerlegbar (Fall b), wird mindestens ein weiterer Versuch unternommen, die Anwendung zu platzieren. Die laufende Platzierung wird wie im vorherigen Fall zunächst abgebrochen und anschließend werden die Schritte 1 bis 17 erneut durchlaufen, allerdings angereichert um die Aufspaltung der Anwendung in ihre Teilkomponenten.

11b	–	–
12b	Nicht-Zuweisungsereignisse an alle nicht ausgewählten Gerät(e) versenden	
13b	Widerruf der Bereitstellungsereignissubskription	
14b	Widerruf Subskription auf Anwendungsplatzierungsanforderung (5)	
15b	Widerruf des Advertisements (2)	
16b	Abschluss der atomaren Transaktion	

Anschließend erfolgt ein erneuter Versuch der Anwendungsplatzierung, wie beschrieben jedoch auf der Ebene der Teilanwendungen (ta). Es wird rekursiv versucht, alle Teilanwendungen im Netzwerk zu verteilen.

1.1ta	Aufteilung der Anwendung in atomare Teilanwendungen	
1.2ta	eröffnet atomare Einheit für Teilanwendungen auf nächstniedrigerer Ebene	
1.1ta	Aufteilung der Anwendung in atomare Teilanwendungen	
1.2ta	eröffnet atomare Einheit für Teilanwendungen auf nächstniedrigerer Ebene	

Falls die Verteilung erfolgreich durchgeführt werden konnte, folgen die Schritte

17ta	Abschluss der atomaren Transaktion	

bzw. bei nicht erfolgreicher Durchführung

17taa	Abschluss der atomaren Transaktion	
18ta	Meldung der nicht erfolgreichen Verteilung der Anwendung an Nutzer.	

Wie in dem vorangegangenen Absatz bereits erwähnt, werden zur Platzierung von Anwendungen prinzipiell drei unterschiedliche Ereignistypen (inklusive der dazugehörigen Advertisements und Subskriptionen) genutzt: *Anforderungsereignisse*, *Bereizstellungsereignisse* und *Zuweisungsereignisse*. Die Grundstruktur

aller Ereignisse besteht aus Name/Wert-Paaren. Dies gilt auch für die Advertisements (Anforderungsereignisse) und Subskriptionen (Anforderungsereignisse, Bereitstellungsereignisse, Zuweisungsereignisse). Dabei bestehen insbesondere die Anforderungsereignisse aus einer Menge von Name/Wert-Paaren unterschiedlicher Kategorien, während die Bereitstellungs- und Zuweisungsereignisse zwar auch Name/Wert-Paare enthalten, inhaltlich jedoch vorwiegend digitale Anweisungen (Bereitstellung ja/nein; Zuweisung ja/nein) umfassen. Es bietet sich an, die Name/Wert-Paare insbesondere für die Anforderungsereignisse in Kategorien einzuteilen. Eine mögliche Einteilung dazu bietet exemplarisch die folgende Aufzählung.

- Sensor.
  Diese Gruppe beschreibt Anforderungen an verfügbare Sensoren im Anwendungskontext. Je nach Anwendungskontext existieren unterschiedliche Anforderungen, bspw. Anforderungen nach Sensoren für Licht, Temperatur und Luftfeuchtigkeit bei Raum- bzw. Klimasensoren; im Bereich Automobilentwicklung fallen darunter bspw. Sensoren für Geschwindigkeit, Abstand zum vorausfahrenden Fahrzeug, Fahrbahnbeschaffenheit etc.

- Rechenleistung.
  Unter diese Gruppe fallen Leistungsbeschreibungen der Recheneinheit eines Gerätes wie Taktgeschwindigkeit des Prozessors/der Prozessoren, Prozessorgeschwindigkeit, die Anzahl der verfügbaren Prozessorkerne etc.

- Speicher.
  Unter diese Gruppe fällt die Beschreibung des verfügbaren Speichers sowohl im Hinblick auf Primär- als auch Sekundärspeicher. Dazu gehören Angaben zum Speichertyp, der Zugriffsgeschwindigkeit auf Daten und die verfügbare Kapazität.

- Umgebung.
  In dieser Gruppe werden die Umgebungseigenschaften von Geräten zusammengefasst. Dazu gehören z.B. ein lokal verfügbares Datenbanksystem oder eine latenzarme Kommunikationsverbindung.

- Energie.
  In diese Kategorie gehören die Eigenschaften der Energieversorgung der Geräte, welche insbesondere bei mobilen Geräten eine wichtige Rolle spielen. Zu dieser Gruppe gehören bspw. die Gesamtkapazität des Akkus und die verfügbare Kapazität.

Bei der Einteilung von Anforderungen in Gruppen ist jedoch zu beachten, dass Teile bzw. komplette Klassifikationen sowohl als notwendige, als auch als hinreichende Bedingungen für die Anwendungsplatzierung genutzt werden. Die entsprechenden Anforderungen werden dann entweder zu den notwendigen Anforderungen (Advertisement auf Anforderungsereignisse) bzw. den hinreichenden Bedingungen (veröffentlicht im Rahmen der Anforderungsereignisse) gegliedert.

Prinzipiell ließen sich die Anforderungen auch ohne die Nutzung von Advertisements formulieren, nämlich (1) indem Anforderungen ins Netzwerk geflutet werden. Dies führt jedoch zu einer erhöhten Nachrichtenlast im Netzwerk. Sind differenzierte Anforderungen und Ausführungsgeräte vorhanden, ist eine selektive Weiterleitung der Anforderungsereinisse durch Routing sinnvoll und im Sinne von Effizienz auch notwendig. Daher ließen sich die Anforderungen als Name/Wert-Paare in Notifikationen verpackt auch als einziges Ereignis versenden, wobei sich dann im Vorfeld die Geräte für entsprechende Notifikationen subskribiert haben müssen. Schritt 2 des Ablaufs entfällt dann.

Zusätzlich ist noch zu klären, wann eine Verteilung von Anwendungskomponenten gescheitert ist und der Koordinator die Verteilung abbrechen muss (Schritte 14 bis 17). Einerseits ist ein Abbruch durchzuführen, wenn explizit keine Verteilung möglich ist, der Koordinator also eine entsprechende Nachricht erhält. Prinzipiell müssten alle Geräte bestätigen, dass sie keine Anwendungen ausführen können. Auf zusätzliche Advertisements und Routing von Subskriptionen und Notifikationen kann dabei verzichtet werden, müssten doch alle Anforderungsereignisse an alle Geräte ausgeliefert werden, damit diese antworten können. Alle Geräte müssten alle Anfragen bearbeiten und beantworten, diese Herangehensweise wäre jedoch im Hinblick auf die ressourceneffiziente Kommunikation im Publish/Subscribe-Netzwerk wenig sinnvoll. Durch den brokerseitigen Einsatz von dezentralen Synchronisationsmechanismen, z.B. aus der verteilten, ereignisbasierten Simulation [143], könnte zwar das zu verbreitende Nachrichtenvolumen eingrenzt werden, allerdings ließen sich gleichzeitig nicht die Möglichkeiten einer Publish/Subscribe-basierten Kommunikation ausnutzen.

Sinnvoller ist hier vielmehr, auf eine implizite Strategie zu setzen. Implizit bedeutet, dass der Koordinator die Platzierung abbricht, wenn nicht innerhalb einer bestimmten Zeit Platzierungsangebote eingehen. Diese Zeit lässt sich bspw. durch netzwerkinterne Testnachrichten ermitteln, die von einem Punkt aus Nachrichten aussenden, die bis an die Peripherie des Netzwerks geleitet und von dort wieder zurück zum Ausgangspunkt gesendet werden. Aus der Laufzeit der Nachricht ließe sich ableiten, wie lange nun der direkte Nachrichtenaustausch zwischen dem Koordinator und dem Rand des Netzwerks dauert. Wird das Messverfahren mehrfach, möglichst unter unterschiedlichen Lastsituationen, durchgeführt, kann der Koordinator zuzüglich der benötigten Rechenzeit zur Ermittlung des Matchingwerts am potentiell ausführenden Gerät, eine sinnvolle Zeitspanne ermitteln. Eine weitere Variante ist, dass der Koordinator Anwendungsplatzierungsereignisse veröffentlicht und zunächst großzügig lange auf eine Antwort wartet. Durch Zeitmessung bei erfolgreichen Platzierungen lässt sich anschließend sukzessive der Zeitraum eingrenzen. Da dieses Verfahren ohne zusätzliche Nachrichten auskommt, wird es verwendet. Es wird unter der Bezeichnung „sukzessive Anpassung“ in das Konzept der initialen Platzierung eingeführt.

Die durch Name/Wert-Paare gegliederten Platzierungsanforderungsnotifikationen werden an den potentiellen Zielplattformen, bei entsprechend im Vorfeld abgegebenen Subskriptionen, empfangen. Anhand eines lokalen Monitorings und

einer verfügbaren Beschreibung wird ein *Matchingwert* für die geforderten Anforderungen ermittelt. Ebenfalls als Name/Wert-Paar wird dieser Matchingwert an den Koordinator (nach Abgabe der entsprechenden Subskription) zurück übermittelt. An dieser Stelle sei der Hinweis erlaubt, dass notwendige Platzierungsbedingungen bereits bei der Abgabe von Ankündigungen und der Subskription auf die Anforderungsereignisse abgegeben und verarbeitet werden. Der Matchingwert beschreibt einen Zahlenwert, welcher die Eignung eines Gerätes bezüglich einer Anforderung beschreibt. Die so ausgestatteten Antwortereignisse werden in das Publish/Subscribe-Kommunikationssystem eingespeist. Dies geschieht durch die Abgabe einer Subskription des Koordinators auf eine einheitliche Anwendungs-ID und dem Typ „Bereitstellungsereignis“, unter welcher die Antwortereignisse ebenfalls veröffentlicht werden. Gleichzeitig mit der Abgabe des Bereitstellungsereignisses veröffentlicht das ausführende Gerät eine Subskription unter der Anwendungs-ID und dem Typ „Zuweisungsereignis“. Derweil sammelt der Koordinator die Bereitstellungsereignisse und wertet die erhaltenen Antworten bezüglich ihrer Matchingwerte aus und bestimmt das ausführende Gerät. Geht im definierten Antwortzeitraum genau ein Ereignis ein, so wird der Absender dieser Notifikation als Ausführungsort der Anwendung ausgewählt. Sobald mehrere Antworten eingehen, obliegt dem Koordinator die Auswahl des am besten geeigneten Aufführungsstandortes. Regelmäßig ist daher der Ausführungsort mit den höchsten Matchingwerten auszuwählen. Eine weitere Untergliederung bzw. Auswertung kann anhand weiterer Aspekte erfolgen und durch die individuelle Gruppierung von Name/Wert-Paaren durch den Koordinator durchgeführt werden. Die Klassifizierungen lassen sich wiederum gewichten, sodass der Koordinator individuelle Platzierungsziele erreichen kann.

Für die Spezifikation der Platzierungsziele durch den Koordinator ist es nötig, eine geeignete Funktion zu definieren, mit der für die initiale, aber auch für die laufende Platzierung, die bestmögliche Platzierung der Anwendung, jedoch zumindest der Vergleich zwischen zwei oder mehreren Alternativen möglich ist. Dies wird mit der Definition einer zur Anwendung gehörigen Zielfunktion erreicht. Zur Definition einer Zielfunktion gehört auch die Definition einer Platzierungsstrategie in der bspw. festgelegt wird, ob und welche Aspekte für eine Platzierung mindestens erreicht werden müssen und wie bei der Gleichheit von Matchingwerten umzugehen ist. So lässt sich bei mehreren gleichen bzw. annähernd gleichen Matchingwerten nur eine Anwendung (zufällig gewählt zwischen den möglichen Standorten) deployen oder alternativ die Anwendung an mehreren Stellen, mit einem aufgeteilten Quellereignisraum, verteilen und ausführen. Die Berechnungsvorschrift für die Matchingwerte, die Gruppierung und Bewertung, müssen der Beschreibung der Anwendung beiliegen. Sie werden vom Koordinator (Gruppierung und Bewertung der Matchingwerte) sowie von den potentiellen Ausführungsorten (Berechnung der Matchingwerte) bestimmt. Die Verteilung der Berechnungsvorschriften der Matchingwerte erfolgt analog mit den Advertisements (Schritt 2 der initialen Platzierung) und den Anforderungsereignissen (Schritt 4 der initialen Platzierung).

Im skizzierten Ablauf der initialen Platzierung werden die nicht zum Zuge gekommenen Geräte mit benachrichtigt (Schritt 12ff) und aus dem internen Speicher des Koordinators gelöscht. Eine Speicherung der zunächst nicht zum Zuge gekommenen Geräte und ein Wiedereinsetzen sind zunächst nicht vorgesehen. Allerdings könnte dies sinnvoll sein, falls die Anwendungsinitialisierung nicht erfolgreich ist (Ausbleiben des Ausführungsereignisses von Schritt 13). In diesem Fall könnte das nächstbeste, zunächst unterlegene Gerät, den Ausführungszuschlag erhalten (Rücksprung auf Schritt 11 mit neuer Zielplattform). Der Ablauf würde wie geplant weitergeführt bzw. solange wiederholt werden, bis eine Platzierung zustande gekommen ist. In diesem Fall wird zwischen einer vorübergehend gescheiterten und einer endgültig gescheiterten Platzierung unterschieden. Ein Deployment ist dann gescheitert, wenn entweder eine explizite Nachricht über die gescheiterte Platzierung abgegeben wird oder durch Zeitablauf indirekt detektiert wird.

Zur Erinnerung: Zunächst werden vom Koordinator das Zuweisungsereignis bzw. die Zuweisungsereignisse (Schritt 11) an den ausgewählten Ausführungsort bzw. die ausgewählten Ausführungsorte versandt. Anschließend werden alle anderen interessierten, aber nicht zum Zuge gekommenen Geräte, Nachrichten über die Nichtberücksichtigung in Form von Nicht-Zuweisungsereignissen erhalten (Schritt 12). Diese widerrufen anschließend ihre abgegebenen Bereitstellungsereignissubskriptionen (Schritt 14), während sie auch aus der internen Liste des Koordinators gelöscht werden. Ist das Deployment erfolgreich, emittiert das ausführende Gerät ein Ausführungsereignis, welches vom Koordinator empfangen wird. Ist eine Platzierung jedoch nicht möglich, wird dies mit einem Ausführungsereignis in Form einer Null-Nachricht quittiert. Die Platzierung wird daraufhin abgebrochen (Schritte 11a bis 16a bzw. Schritte ab 11b). Ein neuer Versuch ist im Ablauf bisher nicht vorgesehen, da beim Koordinator bereits die internen Ausführungskandidaten gelöscht wurden und diese bereits die Subskriptionen auf Zuweisungsereignisse widerrufen haben. In der Folge müsste die Platzierung entweder neu initialisiert, oder stattdessen mit der Platzierung von Teilanwendungen fortgeführt werden. Anstatt nun die Platzierung der Anwendung komplett oder ihrer Teilanwendungen neu zu starten, kann stattdessen mit dem Abbruch des Platzierungsversuches solange gewartet werden, bis die Anwendung erfolgreich deployt wurde. Erst dann werden die Nicht-Zuweisungsereignisse durch den Koordinator verschickt und mögliche Platzierungsalternativen bleiben in „Bereitschaft“. Auf diese Weise werden die möglichen Ausführungsorte, die im ersten Anlauf nicht zum Zuge gekommen sind, noch nicht aus ihrer Bereitschaft freigegeben, sondern erst, wenn die Platzierung endgültig erfolgreich war. Diese Vorgehensweise hat jedoch den Nachteil, dass die Ressourcen auf Seite der potentiell ausführenden Geräte längere Zeit blockiert bleiben. Vorteilhaft ist jedoch, dass der Platzierungsprozess nicht neu initialisiert werden muss und daher die Publish/Subscribe-Primitive nicht neu aufgesetzt werden müssen.

Als Kompromiss zwischen beiden Aspekten ist es empfehlenswert, den Platzierungsprozess nicht nach dem ersten erfolglosen Versuch abzubrechen, sondern

einige weitere Versuche durchzuführen. Somit müssen die Publish/Subscribe-Primitive für die initiale Verteilung nicht neu aufgesetzt werden, zweitens müssen die für die Anwendungsausführung notwendigen Ressourcen nicht länger als notwendig geblockt werden. Außerdem kann in einem dynamischen System nicht davon ausgegangen werden, dass die ermittelte, bestmögliche Platzierung noch aktuell ist.

Nach gescheitertem ersten Platzierungsversuch unternimmt der Koordinator daher weitere Platzierungsversuche mit nachgeordneten Geräten. Welcher Trade-Off dabei zwischen der geforderten Aktualität im dynamischen Ensemble und der Einsparung der Publish/Subscribe-Kommunikation besteht, kann empirisch untersucht werden. Vorerst beschränkt sich die Anzahl der alternativen Versuche auf drei. Selbstverständlich ist dies nur möglich, falls überhaupt ausreichend Platzierungsoptionen vorhanden sind. Damit sind insgesamt vier Versuche für die initiale Platzierung der gesamten Anwendung vorgesehen.

	Koordinator	Geräte im Netzwerk
1	eröffnet atomare Einheit für Anwendung auf oberster Ebene	
2	Aussenden eines Advertisements für Anwendung unter Angabe der notwendigen Bedingungen	
3		Wenn notwendige Bedingungen erfüllt: Erstellen einer Subskription auf (2)
4	Versenden eines Anforderungsereignisses mit den hinreichenden Bedingungen	
5	gleichzeitig mit (4) Abgabe einer Subskription auf Platzierungsanforderungsangebot	
6		Ermittlung des Matchingwerts auf der Grundlage der formulierten Anforderungen aus (2) und verfügbaren Ressourcen
7		Rückgabe des Matchingwertes
8		Abgabe einer Subskription für Bereitstellungsereignis
9		Blockieren der ermittelten Kapazität
10	Sammlung der eingehenden Notifikationen auf (4)	

Zunächst erfolgt der Versuch einer Gesamtzuordnung:

11	Zuweisungsereignis an ausgewähltes ausführende(s) Gerät(e) versenden	
12		Empfang des Ausführungsereignisses

An dieser Stelle erfolgt eine Fallunterscheidung: a) Ist eine Gesamtzuweisung möglich (Positives Ausführungsereignis):

13	Nicht-Zuweisungsereignisse an nicht ausgewählte Gerät(e) versenden	
14		Widerruf der Bereitstellungsereignissubskription
15	Widerruf Subskription auf Anwendungsplatzierungsanforderung (5)	
16	Widerruf des Advertisements (2)	
17	Abschluss der atomaren Transaktion	

b) Es ist keine Gesamtzuweisung möglich, d.h. Timeout bzw. Nichteintreffen eines Ausführungsereignisses oder Eintreffen eines negativen Ausführungsereignisses. Dabei werden die folgenden Fälle unterschieden:
b1) Es sind weitere Ausführungsoptionen vorhanden: Es folgen bis zu drei weitere Platzierungsversuche gemäß der Schritte 11 bis 17.
b2) Es sind keine weiteren Versuche möglich, die Anwendung kann jedoch in Teilanwendungen aufgebrochen werden, dann erfolgt die Platzierung gemäß der Schritte zur Initialisierung von Teilanwendungen (siehe Abschnitt 3.4.4).

Wenn keine Zuordnung möglich ist und die Anwendung in keine Unterebene aufgebrochen werden kann, ist die Verteilung endgültig gescheitert. Der weitere Ablauf ändert sich nicht.

11a	Meldung der gescheiterten Platzierung an Anwender versenden	
12a	Nicht-Zuweisungsereignisse an alle bisher nicht ausgewählten Gerät(e) versenden	
13a	Widerruf der Bereitstellungsereignissubskription	
14a	Widerruf Subskription auf Anwendungsplatzierungsanforderung (5)	
15a	Widerruf des Advertisements (2)	
16a	Abschluss der atomaren Transaktion	

Aufgrund der vielfältigeren Abhängigkeiten beim initialen Deployment von Teilkomponenten wird auf eine alternative Platzierung von Anwendungskomponenten verzichtet, hier bleibt es bei nur einem Versuch (siehe Abschnitt 3.4.4).

Ermittlung des Matchingwertes

Der Matchingwert beschreibt, wie gut ein Gerät für die Ausführung einer Anwendung geeignet ist. Dieser wird lokal durch das jeweilige Gerät auf das Anforderungsereignis hin erstellt. Grundlage sind die in Anforderungsereignissen in Name/Wert-Paaren gekapselten Anforderungen. Dabei müssen die Anforderungen zum einen gruppierbar (siehe Abschnitt 3.4.4), zum anderen quantifizierbar sein (siehe Abschnitt 3.3.1, insbesondere Anforderung 7).

Die Ermittlung des Matchingwerts erfolgt anhand der in den Anwendungsanforderungsereignissen gegliederten und quantifizierten Anforderungen und den ermittelten freien Ressourcen (siehe Abschnitt 3.3.1, Anforderung 1). Die Ermittlung des Matchingwerts in einer Umgebung mit unterschiedlich ausgeprägten Ressourcen muss vergleichbar und in Bezug auf zukünftige Teilnehmer skalierbar sein. Eine Möglichkeit, welche in diesem Konzept umgesetzt wird, ist die der relativen Messung. Dabei wird in einem einfachen Bewertungsverfahren sowohl die Ressourcenanforderung als auch die Ressourcenausnutzung ausgedrückt. Es wird demnach für jede Ressourcenanforderung bzw. Ressourcenanforderungsgruppe RR_x einer zu verteilenden Anwendung A_y bezüglich der vorhandenen Ressourcen eines Gerätes DR_x ein Matchingwert $M_{R_x A_y}$ bestimmt. Die Berechnung erfolgt als Ratio zwischen den verfügbaren und geforderten Ressourcen. Daher ergibt sich ein hoher Matchingwert bei einer geringen zu erwartenden Ressourcenauslastung. In Formelschreibweise bedeutet dies $M_{A_y R_x} = DR_x / RR_x$. Intuitiv und auch in dieser Platzierungsoptimierung wird damit eine gute bzw. bessere Platzierung beschrieben.

Vermeidung bzw. Lösung von Konflikten

Konflikte bei der initialen Platzierung können entstehen, wenn mehrere Koordinatoren gleichzeitig Anwendungen platzieren. Die Koordinatoren versenden Ankündigungen für die notwendigen Bedingungen, die Geräte erstellen bei Erfüllung eine Subskription auf die folgenden, vom Koordinator veröffentlichten, Anforderungsereignisse mit den hinreichenden Ausführungsbedingungen. Sind die Anforderungen erfüllbar, antworten die Geräte mit einem ermittelten Matchingwert und reservieren gleichzeitig die benötigten Ressourcen. Werden die Ressourcen nicht ausdrücklich für die Anwendungsausführung durch den Koordinator vorgesehen, werden sie ebenso ausdrücklich durch den Koordinator freigegeben bzw. verfallen nach einer bestimmten Zeitspanne und werden durch das Gerät wieder freigegeben.

Es kann zu Konflikten kommen, wenn ein Gerät von einem Koordinator K_1 ein Anforderungsereignis E_1 mit dem Zeitstempel t_3 für Anwendung A_1 erhält. Sind

die geforderten Ressourcen vorhanden, werden diese blockiert, das Vorhandensein dem Koordinator gemeldet und abgewartet, ob der Zuschlag für die Anwendungsausführung erteilt, die Ausführungsanfrage zurückgezogen wird oder abläuft. Erreicht in dieser Zeit eine weitere Anfrage mit dem Zeitstempel $t_4 > t_3$ das Gerät und die geforderten Ressourcen sind für eine weitere Variante nicht ausreichend, werden keine Ressourcen reserviert und keine Bereitstellungsnachricht an den Koordinator gesendet. Bis dahin ist der Ablauf korrekt. Erreicht das potentiell ausführende Gerät allerdings K_1 nach dem Anforderungsereignis E_1 ein weiteres Anforderungsereignis E_2 für Anwendung A_2, jedoch mit dem Zeitstempel $t_2 < t_3$, kommt es zum Konflikt, da nicht der Zeitpunkt des Eingangs der Anwendungsanforderung entscheidend ist, sondern der Zeitstempel bei Abgabe des Anwendungsanforderungsereignisses. Erreicht also ein Anforderungsereignis E_2 mit dem Zeitstempel t_2 den Koordinator, muss zunächst die bereits durchgeführte Reservierung der Ressourcen für Anwendung A_1 zurückgezogen werden. Daraufhin werden die Anforderungen für Anwendung A_2 geprüft. Ist eine Platzierung möglich, wird der standartisierte Ablauf weitergeführt. Was an dieser Stelle noch fehlt, ist der Rückruf des abgegebenen Matchingwerts. Dies erfolgt durch Abgabe einer Null-Nachricht in Form eines Antwortereignisses (Schritt 7). Die Anwendungsplatzierungsanforderung wird jedoch nicht widerrufen. Grundsätzlich ist der Widerruf eines abgegebenen Matchingwertes solange möglich, wie die Anwendung noch nicht an dem Gerät ausgeführt wird und nicht bereits eine Bestätigung der Ausführung der zu verteilenden Anwendung an den Koordinator in Form eines Ausführungsereignisses gesendet wurde (Schritte 11 und 12). Statt einer Ausführungsbestätigung wird in Schritt 12 eine Fehlernachricht (Null-Nachricht) an den Koordinator gesendet. Hält der Koordinator noch weitere Ausführungsalternativen (sukzessive Anwendungsverteilung) vor, kann die Anwendungsausführung an die nächstbeste Alternative delegiert werden, andererseits bricht die Anwendungszuweisung an dieser Stelle ab (Schritte 13 bis 17). Dies ist allerdings der ideale Ablauf. Denkbar ist jedoch auch, dass während der Platzierung von A_1 und A_2 mit ihren Zeitstempeln t_2 bzw. t_3 das potentiell ausführende Gerät die Reservierung der Ressourcen für Anwendung A_1 nicht mehr zurückziehen kann, da die Null-Nachricht den Koordinator nicht rechtzeitig erreicht und die Platzierung von A_1 fortgesetzt wird, obwohl die Ressourcen lokal bereits für A_2 reserviert wurden. In diesem Fall können weder A_1 noch A_2 ausgeführt werden, es ist ein Deadlock eingetreten. Damit ist das hier beobachtete Verhalten ähnlich den Problemen durch zyklische Abhängigkeiten, die sich aus der Einhaltung des Zweiphasen-Sperrprotokolls [83, 130] ergeben. Grundsätzlich lassen sich Deadlocks mittels konservativer oder optimistischer Verfahren vermeiden bzw. auflösen. Für beide Gruppen von Algorithmen existieren Umsetzungen, die bspw. bei verteilten Datenbanken oder verteilten Simulationen eingesetzt werden. Konservative Verfahren versuchen, Deadlocks zu vermeiden. In der verteilten Simulation werden dazu u.a. zusätzliche Null-Nachrichten versendet, dies ist allerdings zur Vermeidung weiterer Netzwerklast konzeptionell nicht erwünscht. Zudem kennt das die Anwendung potentiell ausführende Gerät nicht die anderen Geräte im Netzwerk, da Publish/Subscribe dies nicht zulässt. Auch

das konservative Barriere-Verfahren von Mattern [142] ist für die ereignisbasierte Kommunikation mit Publish/Subscribe nicht geeignet. Überschneidungen von Anwendungsanforderungsereignissen können minimiert werden, wenn der Zeitstempel beim Eingang der Anwendungsanforderung maßgeblich ist und nicht länger der Zeitstempel bei Abgabe des Anwendungsanforderungsereignisses. Allerdings könnte es bei internen Fehlern einer Warteschlange trotzdem noch zu Vertauschungen kommen. Die optimistische Sicht auf Deadlocks besteht dagegen darin, diese zuzulassen und dafür im Fehlerfall mögliche *Rollbacks* auszuführen. Beispiele für solche Verfahren aus der Welt der Simulation sind der *Time Warp*-Algorithmus [115], die *Lazy Cancellation* [74, 204] oder die *Lazy Reevaluation* [204]. Aufgrund des Charakters von Publish/Subscribe mit der entkoppelten Kommunikation ist der Einsatz von optimistischen Verfahren den konservativen Verfahren vorzuziehen. Dies ist vor allem geboten, weil davon auszugehen ist, dass der Großteil der Ereignisnachrichten durch die Middleware und die integrierten Schlangen und Weiterleitungsalgorithmen in der richtigen Reihenfolge ausgeliefert und empfangen wird und bei konservativen Verfahren mit höheren Latenzen zu rechnen ist.

3.4.5 Alternativer Verteilungsalgorithmus

Im bisherigen initialen Verteilungsalgorithmus wird davon ausgegangen, dass eine Anwendung digital, d.h. als Ganzes bzw. die Gesamtheit aller Anwendungskomponenten ausgeführt wird oder alternativ die Anwendungsausführung komplett scheitert, selbst wenn nur eine Platzierung scheitert. Scheitert bspw. die Verteilung der letzten Anwendungskomponente, ist das Rollback der gesamten Anwendungsplatzierung notwendig. Möglicherweise wurden während der vorherigen Platzierungsrunden Ressourcen blockiert, und andere Platzierungsanfragen konnten nicht bedient werden. Damit wären unnötigerweise Anwendungen nicht ausgeführt worden.

Anstatt den Bytecode von Anwendungen erst komplett zu verteilen und solange mit der Ausführung der Gesamtanwendung zu warten, bis alle Komponenten verteilt und ausführbar gemacht wurden, sind alternativ dazu weitere Varianten möglich. Allein durch die Möglichkeiten von Kombinationen mit der Anwendungsaufsplittung und der Strategie beim Abwarten auf positive Platzierungsangebote und der tatsächlichen Anwendungsausführung ergibt sich eine Vielzahl möglicher Strategien. Doch anstatt diese Möglichkeiten hier durchzudeklinieren, sei nur ein alternativer Vorschlag genannt, nämlich die möglichst schnelle Verteilung vieler Anwendungskomponenten. Dieses Verfahren ist in der Literatur unter dem Namen greedy, zu deutsch gierig, bekannt. Dabei wird die gesamte Anwendung in ihre Komponenten aufgeteilt, diese werden anschließend parallel verteilt, d.h. der oben beschriebene Ablauf in einfacher Form durchgeführt, also erfolgt für jede Komponente eine atomare Platzierung.

	Koordinator	**Platzierungsfähige Broker**
1alt	eröffnet atomare Einheit für die jeweilige Anwendungskomponente	
2alt	Aussenden eines Advertisements für Anwendungskomponente unter Angabe der notwendigen Bedingungen	
3alt		Wenn notwendige Bedingungen erfüllt: Erstellen einer Subskription auf (2alt)
4alt	Versenden eines Anforderungsereignisses mit den hinreichenden Bedingungen	
5alt	gleichzeitig mit (4) Abgabe einer Subskription auf Platzierungsanforderungsangebot	
6alt		Ermittlung des Matchingwerts auf der Grundlage der formulierten Anforderungen aus (2alt) und der verfügbaren Ressourcen
7alt		Rückgabe des Matchingwertes
8alt		Abgabe einer Subskription für Bereitstellungsereignis
9alt		Blockieren der ermittelten Kapazität
10alt	Sammlung der eingehenden Notifikationen auf (4) und Entscheidung über Ausführungszuschlag	
11alt	Zuweisungsereignis an ausgewähltes ausführende(s) Gerät(e) versenden bzw. ohne Zuweisungsereignis Sprung zu Schritt (13alt)	
12alt		Empfang des Ausführungsereignisses
13alt	Nicht-Zuweisungsereignis an alle nicht ausgewähltes/n Gerät(e) versenden	
14alt		Widerruf der Bereitstellungsereignissubskription
15alt	Widerruf Subskription auf Anwendungsplatzierungsanforderung (5)	
16alt	Widerruf des Advertisements (2)	
17alt	Abschluss der atomaren Transaktion und Meldung der nicht erfolgreichen Transaktion	

Bevor die einzelnen Anwendungskomponenten jeweils verteilt werden, muss allerdings die Gesamtanwendung in ihre atomaren Teilanwendungen aufgespaltet werden. Dazu werden vor dem Schritt 1alt die folgenden Schritte durchgeführt:

1.1alt	Aufteilung der Anwendung in atomare Teilanwendungen	
1.2alt	eröffnet atomare Einheit für Teilanwendungen auf nächstniedrigerer Ebene	

Nach dem Schritt 17alt ist zwar die Verteilung der atomaren Teilanwendungen abgeschlossen. Erfolgte jedoch kein Empfang eines Ausführungsereignisses (Schritt 12alt) bzw. konnte keine Zuweisungsentscheidung getroffen werden, so wird die Platzierungstransaktion ohne positives Ergebnis abgeschlossen (siehe Schritt 11alt und 17alt). Diese Komponenten müssen jedoch für eine erfolgreiche Ausführung der Gesamtanwendung dennoch ausgeführt werden, anderenfalls wäre die Platzierung von Teilanwendungen nicht sinnvoll. Die Möglichkeit der Wahl wäre es daher, zunächst die Gesamtanwendung an einer Stelle im Netzwerk zu platzieren, welche jedoch selbstverständlich über ausreichende Ausführungsressourcen verfügen müsste, und davon ausgehend zu versuchen, diese Platzierung im nächsten Schritt zu verbessern.

Daher ergibt sich grundlegend der im Folgenden dargestellte Ablauf.
Möglichkeit 1:

- Platzierung der Anwendung als Ganzes

Ist dies nicht möglich, dann

- Schrittweise Aufspaltung der Anwendung in Komponenten
- Einzelne Verteilung der Komponenten
- Platzierung nur erfolgreich, wenn alle Komponenten verteilt wurden, ansonsten Abbruch

Anschließend erfolgt die adaptive, schrittweise Optimierung.

Möglichkeit 2:

- Zerlegung der Anwendung in atomare Komponenten
- Platzierung aller Anwendungskomponenten auf einem Gerät

Ist dies nicht möglich, erfolgt bereits an dieser Stelle der Platzierungsabbruch. Ansonsten werden für alle Teilkomponenten global bessere Ausführungsorte gesucht. Die nicht verteilten Anwendungen verbleiben am ersten ausgewählten Ausführungsort.

Diskussion der alternativen initialen Verteilungsoptionen

An dieser Stelle sei angemerkt, dass eine solche „gierige“ Platzierung nur ausführbar ist, wenn ein ausreichend kraftvolles Gerät vorhanden ist. Dabei wird globales Wissen genutzt, welches jedoch u.U. mit einer Mischung von „klumpenhafter“ Häufung von Anwendungskomponenten und einem größeren Verteilungsgebiet der anderen Anwendungskomponenten einhergeht. Bei ersterer Variante, also der Platzierungssuche mit dem schrittweisen Aufbrechen der Anwendung in ihre Komponenten, wird stattdessen nach einer möglichst zusammenhängenden Platzierung von Komponenten gesucht. Die anschließende Optimierung bzw. adaptive Anpassung erfolgt im Umfeld der jeweils durchgeführten Platzierung.

Im Umfeld einer Vielzahl unterschiedlicher Geräte kann nicht immer davon ausgegangen werden, dass es zu jeder Zeit möglich ist, eine Anwendung als Ganzes auszuführen. Ziel ist es stattdessen, zunächst eine gültige Platzierung zu finden und damit die Grundlage für adaptive Veränderungen und Optimierungen zu schaffen. Um nicht beim letzten Schritt der Anwendungszuweisung erneut den gesamten Zyklus durchlaufen zu müssen, ist die Variante der Komplettverteilung unter Berücksichtigung der nächstbesten Alternativen den anderen Varianten vorzuziehen und wird in diesem Konzept verwendet.

3.5 Laufende Anwendungsplatzierung

In einem heterogenen Geräteumfeld, welches sich zudem zur Laufzeit verändert und sowohl die ausgeführten Anwendungen als auch die Gerätezusammensetzung Änderungen unterliegen, ist eine feste Zuordnung von Anwendungen zu den sie ausführenden Geräten nicht sinnvoll, zumindest nicht, solange die Anwendungsverteilung bestimmten Kriterien genügen muss. Die automatische Anwendungsplatzierung muss also dafür sorgen, die Verteilung von Anwendungen, bedarfsgerecht und den aktuellen Gegebenheiten entsprechend durchzuführen.

3.5.1 Voraussetzungen

Die Geräte, die in einer intelligenten und dynamischen Umgebung miteinander vernetzt sind, werden in unserem Anwendungsfall durch eine Middleware verbunden, welche für die Weiterleitung der gewünschten Informationen zuständig ist. Es gelten die in Abschnitt 3.3.1 gestellten Anforderungen.

Die miteinander verbundenen Geräte sind gleichberechtigt und werden durch die Middleware in Form eines azyklischen Netzwerks verbunden, d.h. die Middleware gewährleistet einen Informationsaustausch ohne zirkulierende Informationen, die somit nicht mehrfach und damit fälschlicherweise verarbeitet werden. Die miteinander verbundenen Geräte selbst unterscheiden sich zwar in ihrer Ausstattung, nutzen jedoch die gleiche Ausführungsplattform. Für das (initiale) Deployment

von Anwendungskomponenten auf den jeweiligen Ausführungsplattformen werden durch automatisierte Verfahren die benötigten Bibliotheken und Zusatzkomponenten installiert.

Die Anwendungen selbst bestehen aus definierten Komponenten, welche an Sollbruchstellen miteinander verbunden sind und an diesen auch wieder zusammengefügt werden können.

Neben der Dekomposition und Rekombination sind sie innerhalb des Gerätenetzwerks mit gleichen Ausführungsumgebungen migrierbar und replizierbar, verfügen sie doch über Schnittstellen und entsprechende Implementierungen zur Ausführung dieser Operationen. Darüber hinaus sind sich die einzelnen Komponenten ihrer Anforderungen bezüglich ihrer Ausführungsumgebung und ihres Verhaltens gegenüber der Ressourcenbeanspruchung bewusst.

Zur Migration von Anwendungen und Anwendungskomponenten wird zudem vorausgesetzt, dass die Anwendungen Methoden vorhalten, die neben der Dekomposition, Rekombination, Replikation und Migration auch Schnittstellen bieten und Funktionalitäten implementieren, welche die Serialisierung und Deserialisierung der Komponenten erlauben.

Neben den benötigten Eigenschaften und Funktionalitäten der Ausführungsumgebungen und der Anwendungen selbst, muss für das Deployment von Anwendungen der entsprechende Anwendungscode vorhanden und austauschbar sein. Darüber hinaus ist es wichtig, dass für eine mögliche Wiederverwendung von Anwendungskomponenten, diese eindeutig zugeordnet werden können. Auf die Notwendigkeit, Geräte eindeutig identifizieren zu können, wurde bereits bei der initialen Platzierung eingegangen.

3.5.2 Basisoperationen

Um Anwendungen und Anwendungskomponenten im Netzwerk gemäß eines definierten Ziels auf Grundlage einer gemeinsamen Ausführungsumgebung verteilt ausführen zu können, werden verschiedene Operationen zur Verschiebung und Verteilung verwendet. Zur Anpassung der Anwendungsverteilung werden vier Basisoperationen genutzt, dazu gehören Dekomposition, Migration, Replikation und Rekombination. Die Operationen dienen zur Aufspaltung und Zusammenführung des Ereignisraums sowie der Dekomposition und der Verschiebung von Komponenten.

Ausgelöst werden die Operationen nach Beobachtungen und der Beurteilung, ob und vor allem welche Operationen auf den Anwendungen ausgeführt werden. Die Anwendung reagiert auf den Aufruf einer entsprechenden Methode mit ihrer jeweiligen Implementierung der Operation. Da die konkrete Umsetzung der Methoden jedoch anwendungsspezifisch ist, obliegt sie dem Entwickler. Der Anwendungsentwickler ist auch dafür zuständig, die Anwendungen vor einer Veränderung in einen konsistenten Zustand zu bringen. Die Anwendung stellt

die Operationen als Schnittstelle nach außen bereit, sodass die Grundzüge der Basisoperationen für alle Anwendungen übereinstimmen.

Um die Arbeitsweise der Basisoperationen zu demonstrieren, wird im Folgenden eine komponentenbasierte Anwendung zur Gebäudeüberwachung und Gebäudesteuerung ausgewählt. Gebäude, die für unterschiedliche gewerbliche und/oder Wohnzwecke genutzt werden, sind üblicherweise mit einer Verkabelung ausgestattet. Neben den standardmäßig verlegten Telefonkabeln, ist besonders in gewerblich genutzten Immobilien regelmäßig auch eine Verkabelung für Datennetze zu finden. Von dieser Verkabelung ausgehend werden dann die Vernetzungen der jeweiligen Etagen vorgenommen und anschließend die Etagen untereinander vernetzt. Bei einer Belegung mit unterschiedlichen Mietern bzw. Abteilungen können auch andere Verkabelungsmuster vorliegen. Wurden in den zurückliegenden Jahren vor allem PCs, Server und Telefone und Telefonanlagen kabelgebunden vernetzt, können durch drahtlose Übertragungstechniken Sensoren unterschiedlicher Arten auch ohne (Neu-) Verkabelung integriert werden. Auf diese Weise lassen sich zügig und kostengünstig Sensoren für Raumtemperatur, Lichteinfall oder zur Einbruchserkennung in ein System einbetten. In gleichem Maße wie (mobile) Sensoren Einzug in die Gebäudeüberwachung gehalten haben, lassen sich auch Gebäudesteuerungen in ein Netzwerk integrieren. Benötigten Steuerungen zur Heizungsregulierung und Raumverschattung bis vor ca. einem Jahrzehnt noch den Anschluss an ein Bussystem, lassen sich Thermostate von Heizkörpern und Rollläden heute über einen drahtlosen Anschluss ansteuern. Klassischerweise wird die Steuerung der Geräte von einer Anwendung übernommen, welche in das Szenario eingebunden ist und die angeschlossenen Mechanismen zur Kontrolle von Beschattung, Temperatur etc. steuert. Dazu kommt meist, dass es sich hier um proprietäre Systeme handelt, die wenig bis gar keine Erweiterungsmöglichkeiten für Fremdmodule bieten. Ein Weg, unterschiedliche Systeme zu verbinden, ist die Anbindung an eine gemeinsame Middleware, die als zusätzliche Vermittlungsschicht zwischen den jeweiligen Systemen agiert und einheitliche Schnittstellen anbietet. Eine solche Kommunikationsmöglichkeit bietet eine ereignisorientierte Middleware mit Publish/Subscribe.

Durch die Middleware wird damit eine gemeinsame Kommunikationsplattform eingezogen, allerdings unterscheiden sich die damit im Netzwerk verbundenen Geräte anhand ihrer Fähigkeiten. Die Geräte sind in der Lage, untereinander auf einem gewissen Abstraktionslevel miteinander zu kommunizieren. Dabei wird auf unterschiedliche Verbindungstechniken zurückgegriffen. Diese reichen von kabelgebundenen Verbindungen verschiedener Spezifikationen bis hin zu drahtlosen Verbindungen wie WiFi. Die miteinander verbundenen Geräte selbst haben eine noch weitere Spannweite, reichen sie doch von kleinen Chips mit Sensoren und Kommunikationsschnittstellen, über leistungsfähigere mobile Geräte und Smartphones, bis hin zu ortsgebundenen Steuerungskomponenten sowie Arbeitsplatzrechnern und Servern.

In einem Umfeld unterschiedlicher Geräte und Anwendungen ist auch der folgende Anwendungsfall angesiedelt. Statt eines Raumes soll ein Mensch in me-

dizinischer Hinsicht überwacht werden. Beim betreuten Wohnen ist es üblich, dass Mieter mit unterschiedlichen geistigen und körperlichen Fähigkeiten eine Wohneinheit bewohnen. So benötigen etwa körperlich und geistig fitte Bewohner keine gesonderten Überwachungsgeräte, bei einer fortschreitenden Demenzerkrankung ist die Situation jedoch völlig anders. Ist für die Bewohner zunächst eine Überwachung in dem Sinne angebracht, dass sie zu ihrer eigenen Sicherheit nicht versehentlich elektrische Geräte eingeschaltet lassen und Unterstützung bei Tätigkeiten wie dem Zubereiten von Speisen erhalten, benötigen sie bei fortschreitender Erkrankung möglicherweise ständig Überwachung, um sich und andere nicht zu gefährden und zu erkennen, ob ein Notfall vorliegt. In diesem Fall müssen Ärzte und Pflegepersonal unverzüglich reagieren. Es ist offensichtlich, dass einerseits bei der Ausstattung einer Wohneinheit mit Überwachungs- und Assistenzsystemen unterschiedliche Geräte zum Einsatz kommen, wobei vor allem der Zustand der Bewohner über die Notwendigkeit entscheidet, das eine oder andere Gerät einzusetzen. Sollen die Wohnungen für alle Eventualitäten ausgerüstet sein, müssen alle verfügbaren Geräte in die Wohnung integriert werden. Andererseits kann aus Kostengründen nicht jede Wohneinheit mit allen erdenklichen Geräten ausgestattet werden, sodass eine vollständige Ausstattung der Wohneinheiten mit allen verfügbaren Geräten zur Überwachung von Patienten auch psychologisch für die Bewohner nicht unbedingt von Vorteil ist. Ziel ist es daher, eine Infrastruktur zu schaffen, die eine bedarfsgerechte Ausstattung der Wohneinheiten gemäß den Anforderungen der Bewohner ermöglicht. So werden verschiedene Geräte eingefügt oder bei Bedarf entfernt. Die Kommunikation erfolgt über bereits vorhandene, kabelgebundene Medien wie Telefon- oder Datenleitungen und drahtlose Verbindungen.

Auf diesem Konglomerat unterschiedlicher Geräte werden nun verschiedene Anwendungen ausgeführt. Wird eine Anwendung zentral ausgeführt, müssen alle zur Ausführung benötigten Informationen zur Anwendung und somit zum ausführenden Gerät weitergeleitet werden. Solange ausreichend Kapazitäten für die Weiterleitung und Verarbeitung von Informationen vorhanden sind, ist der Ausführungsort der Anwendung nicht kritisch. Bei begrenzten Kapazitäten jedoch kann es zu Engpässen kommen. So ist die Weiterleitungskapazität mobiler Geräte indirekt durch die zur Verfügung stehenden Energieressourcen begrenzt. Dies gilt bereits bei modernen Smartphones, bei denen durch ständige Datenanbindung und -verarbeitung bereits nach wenigen Stunden die Energiereserven aufgebraucht sind. Bei mobilen Sensoren ist die Situation ähnlich, stehen im Vergleich zum Smartphone jedoch noch weniger Energieressourcen zur Verfügung.

Konkretes Beispiel ist eine Anwendung, die aus Beobachtungsdaten auf den Zustand eines Bewohners schließt und feststellt, ob sich dieser eventuell in einer Notlage befindet. Dazu nutzt die Anwendung unterschiedliche Sensoren, bspw. mobile Beschleunigungssensoren, die an der Kleidung der Bewohner getragen werden und Sensoren im Fußboden der Wohnung, welche die Bewegung von Personen erfassen. Die Sensoren emittieren durchgehend Informationen in Form von Ereignissen, diese werden drahtlos und ungefiltert an die nächste Empfangsstation weitergeleitet und dort an einen Rechner verschickt, der die Informationen

auswertet und Pflegekräfte alarmiert, sobald ein Notfall erkannt wurde. Es ist dann von einem Notfall auszugehen, wenn sich eine Person auf dem Fußboden befindet und sich nicht mehr bewegt und zuvor der Beschleunigungssensor eine fallende Bewegung der Person in Richtung Boden erkannt hat. Die zentrale Auswertung ist außerdem vorteilhaft, wenn potentiell viele Empfänger an den Nachrichten über die gestürzte Person interessiert sind. Doch nicht alle (potentiell) hilfsbedürftigen Menschen wollen in einer Wohneinheit des betreuten Wohnens leben, solange sie geistig und körperlich noch dazu in der Lage sind, ihr Leben ohne Hilfe zu gestalten. Trotzdem möchten sie auf eine schnelle Alarmierung im Notfall nicht verzichten. Außerdem sollen vermeintlich schlechter ausgestattete (im Sinne von fest verdrahteten Kommunikationsmedien) Wohneinheiten ebenfalls schnell und kostengünstig (d.h. ohne aufwendige Baumaßnahmen) nachgerüstet werden können. Aufgrund der weiten Verbreitung und der guten Netzabdeckung könnten zur Weiterleitung von Notfällen auch Mobilfunkgeräte genutzt werden. Die Idee der „Seniorenhandys“ als reines Telefon für den Notfall oder als normales Telefon mit großen, gut sicht- und bedienbaren Tasten ist ein Produkt, welches bereits seit einigen Jahren erhältlich ist. Neuer ist allerdings die Idee, das Mobiltelefon mit Sensoren zu koppeln. So nimmt das Telefon die von den Sensoren erzeugten Ereignisse entgegen und leitet sie dann in Gänze weiter. An einem zentralen Punkt werden die Informationen anschließend ausgewertet. Die Weiterleitung der gesamten Informationen ist jedoch nicht in allen Fällen sinnvoll, so verbraucht die Weiterleitung Energie, weshalb das Telefon öfter nachgeladen werden muss. Weiterhin sind solche Informationen nicht relevant, solange sie nicht zur Erkennung eines Notfalls führen. So können bewusst nicht weitergeleitete und nicht benötigte Informationen zu einer Entlastung der an der Weiterleitung und Verarbeitung beteiligten Komponenten, aber auch zur Vorbeugung missbräuchlicher Datennutzung beitragen. Um die Anwendungsausführung nicht von unterschiedlich ausgestatteten Geräten und Kommunikationsverbindungen ausbremsen zu lassen, sondern die Unterschiede stattdessen als Potential für die Anwendungsausführung ausnutzen zu können, muss die Anwendungsausführung flexibel gestaltet werden können. Dazu gehört die verteilte Ausführung, mit allen dazu notwendigen Kommunikations- und Kooperationsmechanismen. Auch eine komponentenbasierte Struktur der Anwendungen ist nötig, allerdings als Voraussetzung für die den Bedürfnissen und Ressourcen angepasste Verteilung.

Bisher fehlen jedoch noch die Methoden, welche für die bedarfsgerechte und adaptive Platzierung der Anwendungen und Anwendungskomponenten sorgen. Es wird zunächst eine Methode benötigt, welche Anwendungskomponenten verschiebt, d.h. aus ihrer Ausführungsumgebung löst, in konsistentem Zustand verschickt, an ihrem Zielort in die Ausführungsumgebung packt und von dort aus weiter ausführt.

Das in dieser Arbeit vorgestellte Konzept nutzt vier Basismethoden, die Dekomposition, die Migration, die Rekombination und die Replikation. Eine wichtige Basisoperation dient der Verschiebung von Anwendungskomponenten. Die Migration soll dabei nahtlos im Sinne der Übertragung der Zustände der An-

wendungen (und ihrer Komponenten) sowie der Anpassung der Subskriptionen und Ereignisnotifikationen agieren. Eine Anpassung der Ereignisströme ist daher auf der Stufe der Publish/Subscribe-Kommunikation notwendig, wobei auch die konsistente Übertragung und Weiterverarbeitung gepufferter Ereignisse und die konsistente Anwendungsausführung beachtet werden müssen. Die Migration von Anwendungskomponenten wird dabei typischerweise entlang der bestehenden Ereignisströme durchgeführt. Auch wenn dieser als Migration bezeichnete Vorgang zweifellos wichtig ist, sind auch die weiteren Basisoperationen bedeutsam. Um eine Verteilung von Anwendungen überhaupt erst ermöglichen zu können, müssen Anwendungen, wenn ihr Aufbau es gestattet, in Komponenten zerlegt werden, welche dann eigenständig ausgeführt und wiederum verschoben werden können. Die Zerlegung von Anwendungen in feingliedrigere Komponenten wird durch die Dekomposition realisiert. Stammen bspw. gleichartige Informationen aus unterschiedlichen Richtungen, kann es aus Effizienzgründen sinnvoll sein, gleiche Informationsmuster an unterschiedlichen Stellen zu erkennen bzw. zu verarbeiten. Dafür ist es notwendig, Anwendungen zu duplizieren und getrennt voneinander zu behandeln. Die dazu genutzte Basisoperation ist die Replikation. Sollen zu einem späteren Zeitpunkt die replizierten Anwendungen wieder zusammengefügt werden, kommt die Rekombination als Gegenpart zum Einsatz.

Dekomposition

Durch die Dekomposition werden hierarchisch aufgebaute Anwendungen in ihre Teilkomponenten zerlegt. Die separierten Teilkomponenten können dann individuell, auch an unterschiedlichen Umgebungen, ausgeführt werden. Grundsätzlich wird die Dekomposition von der Anwendung selbst durchgeführt, die sich an der Anwendungsstruktur orientiert. Die durch eine Schnittstelle von außen aufgerufene Operation wird durch die Anwendung selbst ausgeführt, die dann bspw. die Zerlegung sowie die Aufteilung und Duplizierung von Variablen übernimmt. Abbildung 3.6 zeigt die Dekomposition einer Anwendung A_0 in unabhängige Teilanwendungen A_{0_1}, A_{0_2} und A_{0_3}.

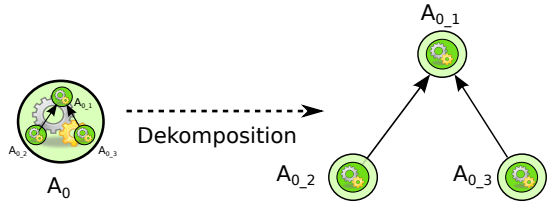


Abbildung 3.6: Dekomposition einer Anwendung in Teilanwendungen

Wird eine bisher umgesetzte Kommunikation zwischen Anwendungen und Geräten auf die ereignisbasierte Kommunikation mittels Publish/Subscribe umge-

stellt, sind möglicherweise Anpassungen notwendig. Dies betrifft bspw. die Ersetzung von bisher intern ausgetauschten Informationen durch Ereignisse, die durch die Anwendungen entsprechend annonciert und abonniert werden müssen. Die Annoncierung wird mit Hilfe der üblichen Publish/Subscribe-Schnittstellen implementiert und die Weiterleitung der Ereignisse obliegt der Publish/Subscribe-Infrastruktur, d.h. Ankündigungen, Subskriptionen und Ereignisnotifikationen sind gemäß dem/den gewählten Routingverfahren zu verteilen. Die dekomponierten Anwendungskomponenten lassen sich individuell von den restlichen Basisoperationen bearbeiten.

Migration

Die Migration verschiebt Anwendungen zwischen unterschiedlichen Ausführungsumgebungen innerhalb des Geräteensembles. Sie dient vor allem der Platzierung von Komponenten gemäß eines Optimierungsziels. Auf der Grundlage gleicher Ausführungsumgebungen sind Komponenten prinzipiell an jedem Ort innerhalb des Netzwerks ausführbar. Aufgabe der Migration ist, die Ausführung von Anwendungen und Anwendungskomponenten auf andere am Netzwerk teilnehmende Geräte zu verschieben. Sie ist die grundlegende Basisoperation für die Anpassung der Anwendungsverteilung an die (sich ändernde) Umgebung.

Informationen innerhalb des Netzwerks fließen in der Regel entlang bestimmter Wege, welche sich spätestens nach einer Phase des Einschwingens, bspw. gemäß bestimmter Routingverfahren, ausbilden und dann bis zu einer Änderung stabil sind. Erfolgt die Bestimmung des Ausführungsortes anhand der Informationsströme, bewegen sich die Anwendungskomponenten entlang dieser Ströme. Generell richtet sich die Entscheidung über die Anwendung bzw. Anwendungskomponenten ausführenden Geräte nach dem verwendeten Entscheidungsverfahren bzw. dem zugrunde liegenden Kostenmodell.

Abbildung 3.7 zeigt die Arbeitsweise der Migration. In der Abbildung sind fünf miteinander verbundene Broker (B_0 bis B_4) dargestellt. Eine Anwendung A_0 befindet sich anfangs auf Broker B_1. Sie erhält Nachrichten vom Broker B_2, welcher allerdings nur als Zwischenstation fungiert, da B_2 die Nachrichten lediglich von B_3 und B_4 an B_1 weiterleitet. Die Anwendung A_0 produziert Nachrichten, die von B_1 an B_0 weitergeleitet werden. Mögliche Nachrichteneinsparungen durch frühzeitigeres Filtern von Nachrichten aus Richtung von B_2 veranlassen A_0 zur Migration nach B_2.

Unabhängig vom Ausführungsziel sowie der Art und Weise wie die Platzierungsentscheidungen zustande kommen, hat die Migration die Aufgabe, Anwendungen aus der bestehenden Ausführungsumgebung herauszulösen, zu transportieren und am vorgesehenen Ausführungsort wieder einzubinden. Zu den Tätigkeiten während der Migration gehören das Anhalten der Anwendung bzw. der Anwendungskomponente in konsistentem Zustand, das Serialisieren, die Weiterleitung an das gewünschte Gerät, das Deserialisieren sowie die Installation der Anwendung/der Anwendungskomponente auf dem Zielgerät. Außerdem müssen,

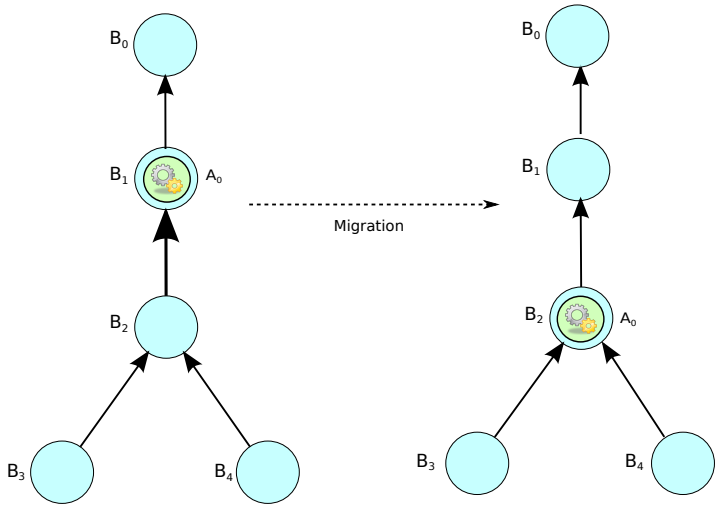


Abbildung 3.7: Migration einer Anwendung in Richtung der Eingangsströme

in Abhängigkeit des genutzten Kommunikationsverfahrens, Informationsströme an die neue Anwendungsplatzierung angepasst werden.

Rekombination

Die Rekombination führt eigenständige Anwendungen bzw. Anwendungskomponenten zusammen, sie ist der Antagonist zur Dekomposition. Die zusammenzuführenden Komponenten wurden zumeist bereits einmal voneinander gelöst und unterlagen eigenständig der Ausführung weiterer Basisoperationen. Es ist ebenfalls möglich, dass unabhängig voneinander deployte Anwendungen zu einer Anwendung zusammengefasst werden. Abbildung 3.8 zeigt die Rekombination von unabhängigen, aber ehemals zusammenhängenden Teilanwendungen A_{0_1}, A_{0_2} und A_{0_3} zu einer Anwendung A_0.

Vornehmliches Ziel der Rekombination ist die Zusammenführung von ehemals zusammenhängenden bzw. abhängigen Komponenten. Eine Zusammenführung ist bspw. sinnvoll, wenn eine separate Ausführung aufgrund der resultierenden Ressourcenbelastung nicht als sinnvoll erscheint. Dies kann in dieser Konstellation immer dann der Fall sein, wenn der Aufwand für die Verwaltung von Ressourcen den Nutzen der verteilten Anwendung übersteigt. Eine solche Situation liegt vor, wenn nur wenige eingehende Informationen verarbeitet werden, aber ungleich mehr Variablen/Zustände redundant gespeichert und entsprechend synchronisiert werden müssen.

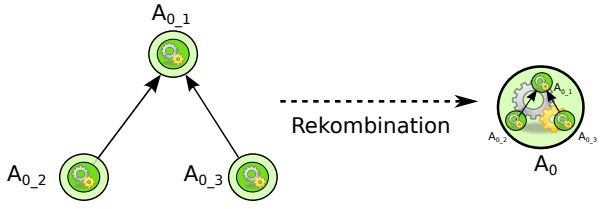


Abbildung 3.8: Rekombination von Teilanwendungen zu einer Anwendung

Die Rekombination von Komponenten zu einer gemeinsamen Komponente kann aus verschiedenen Gründen veranlasst werden, dazu gehören die Neukonfigurationen des Netzwerks, Mechanismen zur Lastbalancierung, ausgeschiedene Informationsquellen etc.

Neben den Veränderungen der Anwendungsstruktur zieht die Rekombination auf Anwendungsebene auch Anpassungen auf der Kommunikationsebene nach sich. Basiert die Kommunikation bspw. auf Ereignissen, so müssen die bestehenden Ereignisströme von und zu den Anwendungen gemäß deren neuen Positionen umgeleitet werden. Das bedeutet, dass Ankündigungen, Subskriptionen und Notifikationen entsprechend der neuen Anwendungsplatzierung verändert werden. Darüber hinaus muss gewährleistet werden, dass Ereignisnotifikationen von Ergebnissen, nach der Umstrukturierung der Notifikationen und Subskriptionen, die aktualisierten Empfänger verlustfrei und in der richtigen Reihenfolge erreichen. Die vormals über die ereignisbasierte Kommunikationsinfrastruktur abgewickelte Kommunikation wird zudem intern, am gleichen Gerät abgearbeitet werden. Für alle mit der ereignisbasierten Kommunikation zusammenhängenden Änderungen existieren entsprechende Schnittstellen, sodass die Änderung der Kommunikationswege letztendlich der Middleware obliegt.

Replikation

Replikation bedeutet zunächst, dass eine Anwendung oder Anwendungskomponente dupliziert und damit mehrfach innerhalb der Umgebung ausführbar gemacht wird. Somit sind unterschiedliche Instanzen im Geräteensemble für separate Bereiche des Netzes zuständig. Im Idealfall sind die einzelnen Anwendungsinstanzen für disjunkte Teilbereiche des Netzwerks verantwortlich. Dies wird dadurch erreicht, dass die replizierte Anwendung zwar die gleichen Typen von Informationen benötigt, selbige aber aus unterschiedlichen, nicht überlappenden Bereichen des Netzwerks stammen. Eine Trennung des Netzwerks in disjunkte Abschnitte wird durch räumlich differente Platzierung, also durch die Migration der replizierten Anwendungen bzw. Anwendungskomponenten erreicht. Abbildung 3.9 verdeutlicht die Arbeitsweise der Replikation im Zusammenhang mit

der Migration. Es ist jeweils ein Brokernetzwerk, bestehend aus sechs Brokern (B_1 bis B_6) dargestellt. Es wird zunächst (linke Teilabbildung) eine Anwendung A_0 auf Broker B_2 ausgeführt, welche einen Ausgangsereignisstrom in Richtung B_1 liefert. Die Anwendung erhält Eingangsinformationsströme über die Broker B_3 und B_4, welche die Informationen von B_5 (über B_3) und B_6 an A_0 weiterleiten. Beide Eingangsflüsse sind gleichen Typs und werden nach definierten Bedingungen durch A_0 gefiltert und weiterverarbeitet. Durch frühzeitiges Filtern lassen sich Weiterleitungskosten einsparen, so dass die Anwendung A_0 in identische Anwendungen A_{0-1} und A_{0-2} repliziert wird. Anschließend migrieren die Anwendungen in Richtung der Eingangsströme. Den Zustand nach der Replikation und der Migration beider Anwendungen zeigt der rechte Teil der Abbildung.

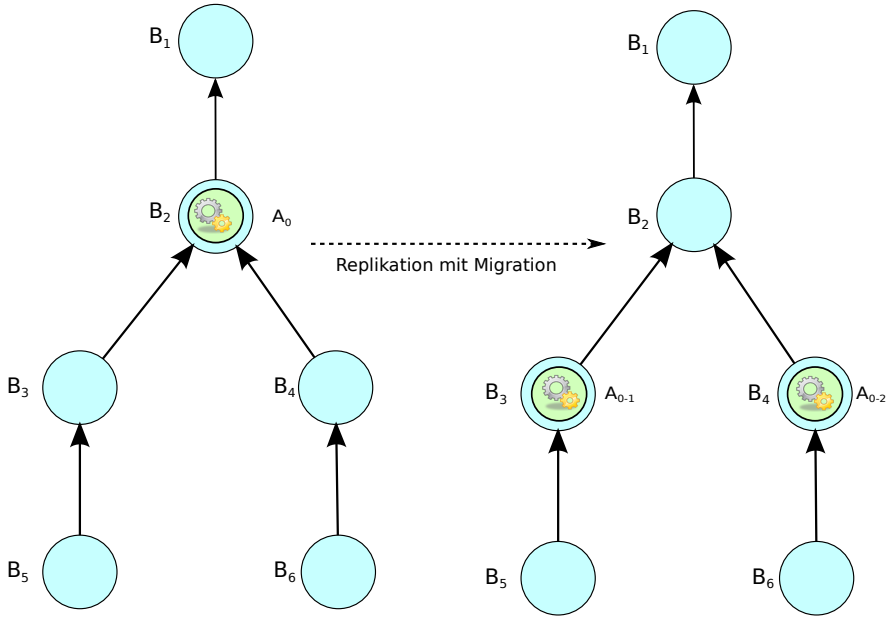


Abbildung 3.9: Replikation und Migration einer Anwendung in Richtung der Eingangsströme

Mit der Replikation einer Anwendung bzw. einer Anwendungskomponente entsteht eine weitere Daten- bzw. Informationssenke. Der Kommunikationsinfrastruktur obliegt es dafür zu sorgen, dass Informationen nicht doppelt, d.h. von beiden Informationssenken empfangen und verarbeitet werden und so disjunkten (Teil-) Mengen des Ensembles entgegenstehen.

3.5.3 Platzierungsziele

Die Platzierung von Anwendungen in einem dynamischen, heterogenen Umfeld kann unterschiedliche Ziele haben. So benötigen manche Anwendungen bspw. die physische Nähe zu bestimmten Sensoren, andere Anwendungen sind wiederum von der Nähe zu Datenbanken oder einer kabelgebundenen Energieversorgung abhängig. In diesem Fall unterscheiden sich die Platzierungsziele nicht von denen der initialen Platzierung (siehe Abschnitt 3.4.1) und werden im Folgenden auch als *statische Ziele* bezeichnet.

Die Erweiterung der laufenden Platzierung besteht an dieser Stelle darin, neben den statisch formulierten Anwendungsanforderungen des (initialen) Deployments aktuelle Beobachtungen und Ziele einfließen zu lassen. Zu diesen Kriterien gehört bspw. die maximale Einsparung von Netzwerkressourcen. Diese Kriterien sind jedoch insoweit abstrakt, dass sie keine konkreten, sondern Maximalforderungen ausdrücken, wobei das Ausmaß der Ergebnisse von dem zu Grunde liegenden Kontext abhängig ist. Daher wird diese Gruppe von Zielen im Folgenden in Abgrenzung von den statischen Zielen als *dynamische Ziele* beschrieben.

Sowohl statische als auch dynamische Ziele finden bei der dynamischen Verteilung der Anwendung Beachtung. So müssen für die Verschiebung von Anwendungen grundlegend die Voraussetzungen für eine Platzierung (statische Ziele) gegeben sein, während die Suche nach einer geeigneteren Ausführungsumgebung auf Grundlage der dynamischen Ziele erfolgt.

Als nächstes stellt sich an dieser Stelle die Frage, wie aus dynamischen Zielen eine Platzierungsentscheidung getroffen werden kann. Die Antwort darauf ist, dass das Optimierungsziel anhand einer Kostenfunktion ermittelt wird. Ziel dabei ist ein möglichst minimaler Kostenverbrauch unter gegebenen Umständen. Dabei fließen, grob gesagt, alle Kosten, die mit der Anwendungsausführung verbunden sind, in ein Kräftemodell. Dieses wird anschließend auf ein Ziel hin optimiert.

Optimierungsziel Ressourceneffizienz

Kostenmodelle dienen dazu, die Art und die Höhe einzelner Kostengruppen zu bestimmen, die auf die Gesamtkosten einwirken. Abhängig von den identifizierten Kosten, deren Höhe und dem Ziel der Platzierung lassen sich unterschiedliche Kostenmodelle entwickeln. Auch in dem in dieser Arbeit vorgestellten Ansatz geht es um die Reduktion von Aufwand und damit um die Reduktion von Kosten, die bei der Ausführung von Anwendungen im Netzwerk entstehen. Das genutzte Kostenmodell basiert auf der Idee der Ressourceneffizienz. Das bedeutet, dass jeglicher Aufwand, der für die Ausführung aller in einem Netzwerk vorhandenen Anwendungen eine Rolle spielt, in ein Modell einfließt. Das Optimierungsziel besteht darin, die Kosten für das gesamte Netzwerk und der dort ausgeführten Anwendungen unter Beachtung der Anwendungsanforderungen zu minimieren.

Um die Gesamtkostensituation beurteilen zu können, müssen mehrere Aspekte betrachtet werden. Zunächst einmal muss aus den Geräten ein Verbund geformt

werden, es existieren somit Aufwendungen für die Kontaktaufnahme und die Aufrechterhaltung des Netzwerks. Die erste Kontaktaufnahme, der Aufbau einer Kommunikationsinfrastruktur (bspw. eines Overlays), ist Teil der Netzwerkinitialisierung. Nach erfolgter Initialisierung muss das Netzwerk auch weiterhin strukturell am Leben erhalten werden, wofür im Umfeld von Publish/Subscribe bspw. *Heartbeat-Events* genutzt werden. Der initiale Aufbau des Kommunikationsverbunds sowie die Kosten zur Aufrechterhaltung, wozu auch die Kosten der Anpassung des Netzwerks an sich ändernde Situationen gehören, sind jedoch unabhängig von den konkret ausgeführten Anwendungen. Analog zu verbrauchsabhängigen und verbrauchsunabhängigen Kosten spiegeln die eben genannten Kosten demnach die *Grundkosten* bzw. *Basiskosten* wider.

Da diese Kosten unabhängig von konkreten Anwendungen entstehen, spielen sie bei dem Ziel der Ressourceneffizienz nur eine Nebenrolle und werden im Folgenden vernachlässigt. Stattdessen stehen die Kosten im Mittelpunkt, welche direkt mit der Anwendungsausführung in Beziehung stehen. Zu der Gruppe dieser *anwendungsbezogenen Kosten* gehören folgende Kostengruppen:

- Platzierungskosten
- Ausführungskosten
- Kommunikationskosten
- Speicherkosten

Platzierungskosten enstehen durch die Bewertung des Aufwands zur Platzierung einer Anwendung. Dies umfasst grundsätzlich den Austausch von Informationen im Vorfeld der eigentlichen Platzierung und deren Koordination. Grundsätzlich fällt für die Organisation eines Netzwerks laufend Overhead an, lässt sich dieser jedoch der Platzierung einer einzelnen Anwendung zuordnen, werden die Kosten dieser Kostengruppe zugerechnet.

Die *Ausführungskosten* resultieren aus der Belegung von Ressourcen mit einer ausgeführten bzw. kurz vor der Ausführung stehenden Anwendung. Während sich zweiteres vor allem auf den Speicherplatz für den Java-Bytecode bezieht, umfasst ersteres neben dem Speicherplatz auch die belegten Ressourcen bezüglich der Verarbeitung von Informationen. Ausführungskosten variieren mit dem Umfang der Rechenoperationen und den verarbeiteten Datenmengen. Bei Geräten mit geringer Ressourcenausstattung wie Sensorknoten und (teilweise älteren) mobilen Geräten kann es bei der Anwendungsausführung durchaus zu Engpässen kommen, was die Ausführung von Anwendungen verteuert.

Die *Kommunikationskosten* sind der bewertete Aufwand, der für die Kommunikation der Anwendungen anfällt und damit alle eingehenden und ausgehenden Nachrichten einer Anwendung umfasst. Geräte in intelligenten Umgebungen sind häufig drahtlos miteinander verbunden. Die drahtlose Kommunikation

ermöglicht im Gegensatz zu kabelgebundenen Verbindungen eine höhere Flexibilität. Mit ihrer Nutzung gehen jedoch höhere Kosten für die Kommunikation einher, sind doch mit dem drahtlosen Versenden und Empfangen von Daten höhere Energiemengen verbunden als mit der Kommunikation mittels drahtgebundener Medien. Doch gerade mobile Geräte haben zudem den Nachteil begrenzter Energieressourcen. Die Batterie ist hier ein wichtiger, limitierender Faktor. Je mehr eine Anwendung Daten konsumiert und produziert, je aktiver die Kommunikation und je teurer ist die Anwendungsausführung. Ließe sich der Kommunikationsaufwand unter Einhaltung der statischen Ziele im gesamten Netzwerk gesehen senken, umso weiter könnte das Gesamtsystem in Richtung Optimalität verschoben werden.

Kosten für die Nutzung von Speicherressourcen fallen an, sobald ausgeführte Anwendungen Speicher auf den sie ausführenden Geräten verbrauchen. So wird regelmäßig Speicher für zwischengespeicherte einzelne Ereignisse, Teilereignismuster oder Variablen verbraucht. Diese Kosten werden allerdings bereits den Ausführungskosten zugerechnet. In Abgrenzung dazu spielen *Speicherkosten* eine Rolle, sobald Ereignisse und Ereignismuster an mehreren Stellen gespeichert werden. Diese kommen dann zum Tragen, wenn aus Gründen der Redundanz Informationen abgelegt werden.

Optimierungsziel Dienstgüte

Analog zu dem beschriebenen Ziel der Ressourceneffizienz können auch andere Ziele in einem Optimierungsmodell abgebildet werden. Neben der grundsätzlichen Ausführungsfähigkeit, also der Erfüllung der statischen Ziele, wird die Dienstgüte als Optimierungsziel genutzt. Wie schon bei der Kostenmodellierung in Bezug auf die Ressourceneffizienz entstehen auch in diesem Fall die bereits beschriebenen *Grundkosten*, welche jedoch nicht Gegenstand der Optimierung sind, da sie auch in diesem Kontext grundsätzlich anfallen und durch die Anwendungsplatzierung nicht beeinflusst werden können.

Die *Ausführungskosten* werden von den wesentlichen Aspekten der Dienstgüte, im vorliegenden Fall von Latenz und Verarbeitungsgeschwindigkeit, beeinflusst. Sowohl Latenz als auch Verarbeitungsgeschwindigkeit lassen sich kostenmäßig modellieren. So sind Kommunikationsverbindungen mit hoher Latenz durch ebenso hohe Kosten darstellbar, während Verbindungen mit geringerer Latenz als kostengünstiger betrachtet werden. Ebenso verhält es sich mit der kostenmäßigen Modellierung der Verarbeitungsgeschwindigkeit. Sind Geräte mit hoher Verarbeitungsgeschwindigkeit (und freien Kapazitäten) vorhanden, ist die Anwendungsausführung günstiger, als wenn Anwendungen auf bereits stark belasteten Geräten ausgeführt werden.

Ist eine Platzierungslösung möglich, bei der sowohl Anwendungen, als auch Kommunikationen günstiger ausgeführt werden können, ist die Anwendungsplatzierung hier vorzunehmen. Bei gegenläufigen Kostenentwicklungen entscheidet die Gewichtung der jeweiligen Aspekte der Kostenfunktion.

3.5.4 Kosten und Kräfte

Die adaptive Platzierung von Anwendungen in einem dynamischen Umfeld erfolgt, wie gesehen, anhand eines Kostenmodells, in das unterschiedliche Kriterien mit differenter Gewichtung einfließen. Nachdem verschiedene Optimierungsziele aufgezeigt wurden, wird am Beispiel des Optimierungsziels *Ressourceneffizienz* gezeigt, wie aus den beschriebenen Kosten ein Kräftemodell zur Ableitung einer Optimierungsstrategie genutzt werden kann.

Eine intelligente Umgebung ist typischerweise durch eine Vielzahl mobiler Geräte und drahtloser Kommunikation gekennzeichnet. Mobile Geräte jedoch haben, trotz fortschreitender Akku-Technologie noch immer das Problem, dass Energieressourcen grundsätzlich begrenzt sind. Daher ist das Ziel, die Gesamtheit der Anwendungen so zu verteilen, dass der totale Energieverbrauch aller Geräte im Netzwerk minimiert wird.

Grundlage der Optimierung ist neben dem Optimierungsziel selbst die Aufstellung eines Modells, welches die Kosten adäquat abbildet. Zur Erinnerung: Basiskosten, die ohne Ausführung von Anwendungen entstehen, werden im Kostenmodell und im abgeleiteten Optimierungsmodell nicht berücksichtigt, währenddessen die Ausführungskosten in das Optimierungsmodell einfließen. Davon werden allerdings die Kosten betrachtet, welche einen wesentlichen Einfluss auf die Ausführungskosten haben. Diese sind, wie in Abschnitt 3.5.3 beschrieben, maßgeblich die Ausführungs- und Kommunikationskosten. Diese Kosten wirken wesentlich auf die Gesamtkostensituation und damit auf die Platzierung von Anwendungen, da durch deren Verschiebung die Kostensituation verändert werden kann. Doch wie lassen sich Kosten modellieren und vor allem: Wie lässt sich ihr Einfluss auf die Anwendungsplatzierung darstellen?

Die Aufstellung der Zielfunktion zeigt, dass die Summe aller Kommunikations- und Ausführungskosten unter Einhaltung der Nebenbedingungen (statische Ziele) zu minimieren ist. Das Ziel, Kosten einzusparen, lässt sich direkt auf die Ausführung von Anwendungen herunterbrechen. Das bedeutet, die Kosten wirken auf die Platzierung jeder ausgeführten Anwendung.

Die Grundidee besteht darin, eine Anwendungsplatzierung zu finden, bei der die veränderte Anwendungsplatzierung geringere Kosten verursacht als die vorherige Platzierung. Zur Modellierung des Einflusses der Kosten auf die Anwendungsplatzierung werden Kräfte genutzt, die im Zusammenspiel auf die Platzierung der Anwendung einwirken und ihren Ausführungsort bestimmen. Die einwirkenden Kräfte werden von den einwirkenden Kosten abgeleitet. Dabei wird das Kräftemodell aus den in Abschnitt 3.5.4 beschriebenen Kosten gebildet.

Die Lösung des Optimierungsproblems kann prinzipiell durch eine allwissende Komponente erzielt werden. Eine mathematische Lösung des Optimierungsproblems wäre zwar grundsätzlich möglich, bei zunehmender Anzahl von Geräten und Anwendungen ist durch die Schwierigkeit des Problems jedoch das Finden einer solchen Lösung nicht durchführbar. Zusätzlich erschwert die Dynamik des

Systems die Lösungssuche. Die Lösung durch eine zentrale Komponente bedingt zudem, dass diese mit dem notwendigen Wissen über den Zustand des Systems versorgt werden muss. Neben der möglicherweise begrenzten Verarbeitungskapazität zieht die zentrale Platzierung einen hohen Kommunikationsaufwand nach sich, der neben der hohen Auslastung auch von schwächer dimensionierten Kommunikationspfaden zu Flaschenhälsen führen kann.

Ausgehend von diesen Nachteilen wird eine dezentrale Verbesserungsstrategie vorgeschlagen, welche lokales Wissen nutzt und somit Kommunikationsaufwand in einer beträchtlichen Größenordnung einspart. Statt der mathematischen Optimierung nutzt der in dieser Arbeit vorgestellte Ansatz eine Heuristik, um die Schwere des Problems zu umgehen und eine verbesserte Platzierung zu finden, auch wenn eine optimale Lösung nicht garantiert werden kann. In Zusammenhang mit dem eingeschränkten Wissen (nicht mehr über das gesamte Netzwerk, sondern nur noch über die Umgebung) wird das Problem beherrschbar, auch wenn die Vielzahl lokaler Verbesserungen kein Gesamtoptimum garantiert. Aufbauend auf einer initialen Verteilung ermöglicht die schrittweise Verbesserung jedoch zumindest eine Annäherung an den Idealzustand. Dieser ist erreicht, wenn alle Anwendungen in der geforderten Qualität (Nutzeranforderungen an die Anwendungen und die daraus abgeleiteten Anwendungsanforderungen bezüglich der sie ausführenden Geräte) mit minimalen Kosten ausgeführt werden.

Die Platzierungsstrategie nutzt zudem existierende Ereignisströme, so können bspw. durch die ohnehin vorhandenen Heartbeat-Events die Kostensituationen der Nachbarbroker übertragen werden.

Kostenmodell

Die Platzierung von Anwendungen erfolgt unter der Annahme, dass aus Kostendifferenzen, die durch unterschiedliche Kostensituationen entstehen, Kräfte abgeleitet werden. Diese Kräfte wirken auf unterschiedliche Weise auf die Anwendungsplatzierung ein. Die resultierenden Kräfte sind von der ermittelten Höhe und der Auswahl der zu vergleichenden Kostensituationen abhängig. Grundlage für einen Vergleich sind bspw. die Kostensituationen benachbarter oder entfernter Knoten. Eine hohe Kraft bedeutet eine hohe Kostendifferenz und damit ein Potential. Dieses Potential ist äquivalent zu den möglichen Einsparungen, die bei einer Verschiebung von Anwendungskomponenten zu erreichen sind. Je nachdem, welche Potentiale ermittelt werden, wirken sie in verschiedene Richtungen auf die Anwendungsplatzierung ein. Niedrige bzw. nicht vorhandene Potentiale schließen auf geringe bzw. keine weiteren möglichen Kosteneinsparungen und gleichzeitig auf ein ausgeglichenes System bzw. einen Gleichgewichtszustand.

Kosten. Das in dieser Arbeit genutzte Modell zur Kostenermittlung C bezüglich einer Anwendung A_z nutzt unterschiedliche Arten von Kosten, nämlich (1) Ausführungs- und (2) Speicherkosten ($C_{A_z ex}$ und $C_{A_z st}$), (3) Weiterleitungskosten ($C_{A_z fw}$) und (4) Platzierungskosten ($C_{A_z plac}$).

Die Ausführungskosten $C_{A_z ex}$ einer Anwendung A_z lassen sich durch den Verbrauch von Ressourcen während der Anwendungsausführung bestimmen. Zu diesen gehören die Nutzung von Rechenkapazität und -zeit. Die Kosten bestimmen sich nach den verbrauchten Einheiten, die zudem je nach Auslastung des ausführenden Geräts unterschiedlich teuer sind. So kostet eine verbrauchte Einheit bei einer insgesamt hohen Gesamtauslastung mehr, als wenn die Auslastung des Gerätes von vornherein geringer ist. Wie auch bei der Berechnung des Matchingwertes im Rahmen der initialen Platzierung wird das Verhältnis zwischen den verbrauchten Ressourcen und den insgesamt vorhandenen Ressourcen je Ressourcengruppe berechnet. Je höher die Belastung der Ressourcen mit der jeweiligen Anwendung ist, desto höher sind die Kosten. Die Ausführungskosten werden für jede Anwendung bestimmt, vorausgesetzt diese lassen sich (bspw. durch Monitoring) ermitteln. Die Ausführungskosten $C_{exB_y A_z}$ einer Anwendung A_z auf einem Gerät (Broker) B_y bestimmen sich aus Ressourcenverbrauch RC_{A_z} durch die Anwendung A_z und den vorhandenen Ressourcen des ausführenden Gerätes DR_{B_y}, es ergibt sich $C_{exB_y A_z} = RC_{A_z}/DR_{B_y}$. Ressourcenverbrauch und verfügbare Ressourcen werden, äquivalent zur initialen Platzierung, weiterhin in ihre Teilbereiche aufgebrochen. Bei einer Gruppierung von Anforderungen und verfügbaren Ressourcen ist das Verhältnis RC_{A_z}/DR_{B_y} gleich der Summe der jeweiligen Teilgruppen von Ressourcen bzw. Anwendungen. Bei gruppierten Anforderungen/Ressourcen ergeben sich Ausführungskosten nach den Gruppen G_i von 1 bis n als

$$\sum_{i=1}^{n} RC_{A_z i}/DR_{B_y i}.$$

Speicherkosten $C_{A_z st}$ einer Anwendung A_z entstehen durch die Bewertung des Aufwands zur Speicherung/Pufferung von benötigten Ereignissen, Ereignismustern von Anwendungen bzw. Zuständen. Wie bei anderen Ressourcen auch, sind die Speicherkapazitäten grundsätzlich begrenzt. Die Kosten werden je Anwendung A_z und wiederum auf der Grundlage der Auslastung der jeweiligen Ressourcen berechnet. Die Speicherkosten einer Anwendung ergeben sich daher als Ratio der für die Anwendung beanspruchten zu den insgesamt vorhandenen Ressourcen, es ergibt sich daher

$$C_{A_z st} = R_{mem} C_{A_z}/DR_{memx}.$$

Weiterleitungskosten $C_{A_z fw}$ einer Anwendung A_z entstehen prinzipiell für alle Nachrichten, die von einem Broker an seine Nachbarbroker weitergeleitet werden. Unter dem Begriff der Nachrichten werden Notifikationen und Nachrichten zum Austausch von Kontroll- bzw. Managementinformationen zur Organisation der Publish/Subscribe-Middleware zusammengefasst. Besonders in Umgebungen mit mobilen Knoten und drahtloser Kommunikation, bei denen die Übertragungskapazität direkt durch eine geringe Übertragungsbandbreite und indirekt über die begrenzt zur Verfügung stehende Energie für die Kommunikation beschränkt ist, spielen Weiterleitungskosten eine wichtige Rolle, ist doch

Kommunikation mehr noch als Verarbeitung der Hauptkonsument von Energie.

Für das in dieser Arbeit vorgeschlagene Platzierungsmodell sind jedoch nicht alle weitergeleiteten Nachrichten von Bedeutung, sondern lediglich solche, auf die die lokal ausgeführten Broker Einfluss nehmen können. Dazu gehören eben ausdrücklich nicht die obligatorischen Nachrichten, die zum Betrieb der Publish/Subscribe-Infrastruktur notwendig sind, sowie die Notifikationen, die lediglich weitergeleitet werden, d.h. nicht von einer Anwendung auf dem jeweiligen Broker verarbeitet bzw. erzeugt wurden. Weiterleitungskosten sind demnach lediglich von den bearbeiteten bzw. erzeugten Nachrichten der Anwendungen abhängig, die auf dem Broker ausgeführt werden. Die Kosten lassen sich auf die jeweiligen Anwendungen herunterbrechen, zumindest wenn keine Abhängigkeiten zwischen den Anwendungen in Form gemeinsam genutzter Ereignisströme bestehen.

Grundsätzlich wird in dem hier vorgestellten Modell jede versendete bzw. empfangene Nachricht, sofern sie durch die Anwendung beeinflusst werden kann, mit einem festen Schlüssel bewertet, da durch den Austausch von Nachrichten nicht nur beim Broker, der die Anwendungen beheimatet, sondern auch bei benachbarten Brokern Aufwand und damit Kosten generiert werden. Die Umrechnung von Nachrichten zu Kosteneinheiten erfolgt im Verhältnis 1 zu 1, wobei bei späteren Modellverfeinerungen dies, ähnlich wie bei der Berechnung der Matchingwerte im Rahmen der initialen Platzierung und der Berechnung von Ausführungskosten, angepasst werden kann.

Da Nachrichten ausgehend von den Anwendungen von oder in Richtung der Nachbarbroker ein- bzw. ausgehen, lassen sich Kosten in ein- und ausgehende Nachrichten $C_{A_z fwin}$ bzw. $C_{A_z fwout}$ sowie je Richtung eines benachbarten Brokers B_m $C_{A_z fwB_m}$ unterscheiden, wobei zusätzlich nach benachbartem Broker unterschieden werden kann, also $C_{A_z fwinB_m}$ bzw. $C_{A_z fwoutB_m}$. Außerdem lassen sich Ströme je benachbartem Broker zusammenfassen: B_m $C_{A_z fwB_m in}$ bzw. $C_{A_z fwB_m out}$. Eine Kosteneinheit c entspricht dabei genau einer versendeten Nachricht, es gilt $|c| = |AnzahlNachrichten|$.

Sowohl Weiterleitungskosten als auch Ausführungs- und Speicherkosten treten aus der Sicht eines Brokers zunächst unabhängig von einer durchgeführten Platzierungsadaption auf. Allerdings variiert die Höhe der aufgelaufenen Kosten von der jeweiligen Ausführungsumgebung, insbesondere bei den Ausführungs- und Speicherkosten. Sie spiegeln gemeinsam die Kosten, die durch die Anwendungsausführung entstehen, wider und sind prinzipiell abhängig von den Ressourcen des ausführenden Geräts. Bei ausreichend großen Ressourcen sind sie allerdings vernachlässigbar. Weiterleitungskosten fallen bei jedweder Verarbeitung an, einzig aus globaler Sicht können durch eine optimierte Anwendungsplatzierung Weiterleitungskosten eingespart werden.

Gegenüber der bisher dargestellten Kosten, welche während der Anwendungsausführung entstehen, reflektieren die Platzierungskosten $C_{A_z plac}$ einer Anwendung A_z die zusätzlichen Kosten, die durch die Ausführung von Anwendungsoperationen entstehen und damit den Overhead der Ausführungsortsänderung

widerspiegeln. So muss bspw. im Falle der Migration die Anwendung in der neu bestimmten Anwendungsausführungsumgebung deployt werden, was die konsistente Überführung von Zuständen und der gepufferten Ereignisse mit einschließt, ebenso aber auch den Aufwand für den Aufbau neuer und den Widerruf nicht mehr benötigter Subskriptionen. Die Kosten umfassen dabei alle Tätigkeiten zur Ressourcenermittlung, dem Vergleich, der Überführung der Anwendung in einen überführungsfähigen Zustand, der Übertragung der nötigen Ereignisse, dem Deployment der Anwendung am neuen Standort und der nötigen Anpassungen des Nachrichtenflusses der Publish/Subscribe-Kommunikationsstruktur. Da dieser Aufwand erst dann konkret ermittelt werden kann, sobald eine Änderung der Anwendungsplatzierung durchgeführt wird, die Kostenabschätzung jedoch vor Ausführung einer ebensolchen benötigt wird, erfolgt die Kostenabschätzung anhand des am bisherigen Ausführungsort verursachten Ressourcenverbrauchs (Kostenart 1 und 2) und eines pauschalen Faktors f und g, und analog dazu die bisher verursachten exklusiven Ereignisströme multipliziert mit einem Faktor h. Es ergeben sich daher Platzierungskosten C_{plac} einer Anwendung A_z mit

$$C_{A_z plac} = f \cdot C_{ex A_z} + g \cdot C_{st A_z} + h \cdot C_{A_z fw}.$$

Kostenzielfunktion

In Abschnitt 3.5.3 wurden unterschiedliche Platzierungsziele angerissen. In einer ubiquitären Umgebung unterscheiden sich teilnehmende Geräte bezüglich ihrer Position im Raum und ihrer zur Verfügung stehenden Ressourcen. Die Geräte sind vielfach mobil und kommunizieren per drahtloser Verbindung miteinander. Bei drahtlosen Geräten sind zunächst die Energieressourcen der hauptsächliche limitierende Faktor, während die Verarbeitungsressourcen bei modernen Smartphones oder auch bei Kleinstgeräten, z.B. bei einem Raspberry Pi, dagegen eher kein Mangel sind und auch bei Sensoren, die bereits nicht benötigte Ereignisse filtern, ist die Situation ähnlich. Stattdessen sind Kommunikation und der damit verbundene Energieverbrauch ressourcenintensiv und zusammen mit den begrenzten Energieressourcen der vor allem limitierende Faktor. Das in dieser Arbeit vorgestellte Optimierungskonzept zielt auf die Einsparung von Energie und damit auf die Ressourceneffizienz (siehe Abschnitt 3.5.3).

Der vorherige Abschnitt hat die Kosten ermittelt, welche anwendungsbezogen anfallen und für eine Optimierung der Anwendungsplatzierung genutzt werden können. Die Kostenarten sind (1) Ausführungs- und (2) Speicherkosten ($C_{A_z ex}$ und $C_{A_z st}$), (3) Weiterleitungskosten ($C_{A_z fw}$) und (4) Platzierungskosten ($C_{A_z plac}$). Die Kosten sind auf der hier vorgestellten Ebene, siehe Abschnitt 3.5.4, bezogen auf die Anwendung und die jeweils betrachteten Ausführungsorte. Für die Gesamtsumme aller im Netzwerk anfallenden anwendungsbezogenen Kosten gilt, dass diese prinzipiell durch Aufsummierung der jeweiligen Kosten pro Anwendung und Ausführungsort (abzüglich der doppelt gezählten Kosten, bspw. der Weiterleitungskosten) ermittelt wird.

Das Optimierungsziel Ressourceneffizienz zu erreichen heißt, bei der Abarbeitung aller Anwendungen unter Beachtung der geforderten Dienstgüteanforderungen der Nutzer an die Anwendungen sowie der Anforderungen der Anwendungen untereinander und bezüglich der sie ausführenden Geräte, möglichst wenige Ressourcen zu verbrauchen. Das bedeutet, die im Netzwerk vorhandenen und auf die Anwendungen bezogenen Kosten zu senken, was direkt mit der Senkung der Kosten für Anwendungsausführung, Speicherplatzbedarf und Kommunikation einhergeht. Platzierungskosten fallen in der Regel nur dann an, wenn eine Änderung der Anwendungsplatzierung durchgeführt wird. Unter der Annahme, dass diese einmaligen Kosten sich während der Dauer der Anwendungsausführung amortisieren und die Kostensituation grundsätzlich in einem Zustand bei ausgeführten Anwendungen betrachtet wird, werden die Platzierungskosten im Kostenmodell nicht berücksichtigt.

Oberstes Ziel ist es, die Ausführungs-, Speicher- und Weiterleitungskosten K (als Gesamtheit von C_{ex}, C_{st} und C_{fw}) ausgedrückt in der Kostenfunktion $k(x)$ in ihrer Gesamtheit, sprich die Kosten aller Anwendungen A im gesamten Netzwerk, zu minimieren. Dabei müssen sowohl die geforderte Dienstgüte Γ, als auch die Anwendungsausführungsvoraussetzungen Φ bezüglich der Anwendung x eingehalten werden.

Es gilt daher die Zielfunktion:

$$min\ \{k(x) : x\ \epsilon\ A\ mit\ \gamma(x)\ \geq \Gamma_x\ und\ \phi(x)\ \geq\ \Phi_x\}$$

Aus der Kostenfunktion K mit

$$K = k(x)$$

ergibt sich für alle $x \epsilon K$, als Summe aller C_{fw}, C_{ex} und C_{st}

$$K = k(x) = C_{fw} + C_{ex} + C_{st}.$$

C_{ex} und C_{st} ergeben sich selbst als Summen aus den Kosten, die bei der Anwendungsausführung anfallen, damit gilt

$$C_{ex} = \sum_{j=1}^{m} \sum_{i=1}^{n} i = C_{ex} B_j A_i$$

sowie

$$C_{st} = \sum_{j=1}^{m} \sum_{i=1}^{n} i = C_{ex} B_j A_i$$

mit A_i als jeweilige Anwendung und B_j als jeweiligem Ausführungsort bei n-Anwendungen und m-Ausführungsorten. Bei der Ermittlung der Weiterleitungskosten C_{fw} im gesamten Netzwerk spielt die Unterteilung der Nachrichtenströme in ein- und ausgehende Ströme keine Rolle, allein die Gesamtzahl ist für die globale Sicht entscheidend. Daher werden an dieser Stelle die Weiterleitungskosten

zwischen benachbarten Brokern B_a und B_b sowohl in Richtung B_a (B_bB_a) als auch in Richtung B_b (B_aB_b) betrachtet. C_{fw} ergibt sich daher aus den Nachrichtenströmen benachbarter Broker.

$$C_{fw} = \sum_{j=1}^{m} \sum_{i=1}^{n} iC_{fwB_iB_j} + C_{fwB_jB_i} \; \forall \; m, n \; mit \; B_mB_n \; und \; B_mB_n$$

Eine Kosteneinheit c entspricht dem Betrag einer Nachricht, es gilt damit $c = |AnzahlNachrichten|$.

Die aufgestellte Zielfunktion gilt global für das gesamte Netzwerk. Allerdings besteht ein linearer Zusammenhang zwischen der Anzahl der Broker und Anwendungen im Netzwerk und den Gesamtkosten, da sich die Gesamtkosten aus den Teilkosten zusammensetzen, welche als Summen von der Anzahl der Anwendungen und Broker bzw. Ausführungsorte abhängig sind. Der Optimierungsansatz und die hier vorgestellte Zielfunktion sind nur dann gültig, wenn sich eine Kosteneinsparung einer Kostenkomponente an einem Ausführungsort bezüglich einer Anwendung erreichen lässt, ohne dass Kosten einer anderen Anwendung an einem anderen Ausführungsort erhöht werden. Dies gilt für die Ausführungskosten, da diese nur von der Kostensituation an einem ausführenden Gerät abhängig sind. Die Annahme gilt auch für die Weiterleitungskosten, allerdings nur dann, wenn nur die Nachrichtenströme betrachtet werden, die von einer Anwendung exklusiv genutzt werden. Negative Auswirkungen auf die Kostensituation benachbarter Anwendungen und Ausführungsorte sind daher ausgeschlossen. Speicherkosten entstehen, sobald ehemals an einem Gerät ausgeführte Anwendungen an verschiedenen Orten deployt wurden und diese nun getrennt voneinander arbeiten. Diese Kosten entstehen bereits bei der ersten Aufspaltung der ehemals zusammenhängenden Anwendungen und ändern sich grundsätzlich bei weiteren Platzierungsanpassungen nicht mehr. Weitreichenderen Einfluss auf die Gesamtkosten haben stattdessen die Kommunikationskosten, falls Anwendungen sich zunehmend voneinander entfernen. Auch im Vergleich zu den Ausführungskosten sind die Speicherkosten relativ gesehen kleiner, beinhalten die Ausführungskosten doch auch die Kosten der von der Anwendung ohnehin exklusiv genutzten Speicher und die Kosten für die Nutzung weiterer Ressourcen. Aus diesem Zusammenhang ergibt sich die Relevanz der einzelnen Kostengruppen in absteigender Reihenfolge mit (1) Weiterleitungskosten, (2) Ausführungskosten und (3) Speicherkosten. Daher steht die Minimierung der Kommunikationskosten im Vordergrund der Optimierung.

$$min\{k(x)\} \Rightarrow: min\{c_{fw(x)}, \; mit \; x \; \epsilon \; A, \; \gamma(x) \; \geq \; \Gamma_x \; und \; \phi(x) \; \geq \Phi_x\}$$

Hier sei angemerkt, dass die lokale Zielfunktion nur bezüglich der jeweiligen Anwendung und nicht bezüglich des Ausführungsortes gilt.

Würde die lokale Zielfunktion auch bezüglich des Ausführungsortes gelten, ließe sich argumentieren, dass bereits die Nichtausführung einer Anwendung an einem Ausführungsort die Kosten senkt und allein durch die Nichtausführung

von Anwendungen Kosten minimiert und die Optimierung erreicht wird. Daher ist Bedingung, dass die Anwendung nach wie vor ausgeführt wird. Diese Anforderung wird in den Anforderungen zur Dienstgüte der jeweiligen Anwendungen hinterlegt. Damit ist die lokale Optimierungsstrategie nur im Vergleich und unter Mitwirkung der nachbarschaftlichen Ausführungsorte anwendbar. Es wird also nach einer lokalen Optimierung innerhalb eines begrenzten Umfelds gesucht.

Unter der Annahme, dass durch die Verlagerung von Anwendungen Änderungen des Gesamtkostenbetrags erreicht werden und diese Änderungen die Ausführungskosten und Speicherkosten überdecken, ist für die Höhe der Gesamtkosten allein die Optimierung der Weiterleitungskosten maßgebend.

Zwischen den gesamten Weiterleitungskosten und den Weiterleitungskosten je Anwendung und Ausführungsort gilt ein linearer Zusammenhang, durch die Aufsummierung der jeweiligen Nachrichtenströme lassen sich die gesamten Weiterleitungskosten ermitteln. Es stellt sich die Frage, ob durch die Optimierung einzelner Ereignisströme grundsätzlich ein globales Optimum erreicht wird. Mit anderen Worten: Wirkt sich eine positive/negative Veränderung einer lokalen Kostensituation in gegenteiliger Weise auf die Kostensituation anderer Anwendungen aus? Wenn jeweils nur unabhängige Nachrichtenströme und damit Kosten verändert werden, führt eine lokale Verbesserung der Kostensituation auch regelmäßig zu einer verbesserten Situation der Gesamtkosten, auch wenn der Einsatz einer Heuristik nicht zwangsläufig zu einem globalem Optimum führt.

Kräftemodell

Kräfte entstehen aus unterschiedlichen Kostensituationen und wirken auf die Anwendungsplatzierung ein, indem sie die Anwendung in die Richtung ziehen, in die auch die Kräfte wirken. Der Betrag der Kräfte wird im Falle der Ausführungs- und Speicherkosten anhand der Differenz zwischen den aktuellen und geschätzten alternativen Platzierungen (Ausführungs- und Speicherkosten) bestimmt, während die Weiterleitungskosten anhand der möglichen Verbesserungen der aktuellen Kostensituation ermittelt werden. Die Platzierungskosten ergeben sich anhand der geschätzten Kosten für die Neuplatzierung.

In dem in dieser Arbeit entwickelten Platzierungsmodell von Anwendungen in einem Netzwerk lassen sich die Kräfte, die auf die Anwendungsplatzierung einwirken, in drei Gruppen zusammenfassen:

1. Lassen sich durch eine alternative Anwendungsplatzierung Kosten einsparen, wirkt eine Kraft in Richtung des alternativen Ausführungsortes und weg von der bisherigen Platzierung. Solch eine Kraft initiiert die Verschiebung einer gesamten Anwendung oder führt, bei einem komponentenbasierten Anwendungsaufbau, auch zu einer Verschiebung einzelner Komponenten.

2. Die verteilte Ausführung von Anwendungen und Anwendungskomponenten kann, bspw. durch die benötigte, doppelte Haltung von Daten, zu

erhöhtem Bedarf an Speicher und/oder Synchronisationssaufwand führen. Dieser ergibt sich aus der Tatsache, dass nun bspw. Werte für Variablen an mehreren Stellen gespeichert und aktuell gehalten werden müssen. Aus diesem Aufwand heraus lässt sich ebenfalls eine Kraft ableiten, welche sich dem Auseinanderziehen von Komponenten entgegen stellt und stattdessen ein Zusammenziehen von Komponenten bewirkt. Diese Kraft kann dabei so stark ausgeprägt sein, dass ehemals miteinander verbundene Komponenten wieder zusammengeführt oder Komponenten, welche im Vergleich zu ihrem Ergebnis einen hohen Aufwand verursachen, mit ähnlichen Komponenten zu einer Einheit geformt werden.

3. Beide bisher beschriebenen Kräfte wirken auf die Anwendungsplatzierung ein, ziehen sie doch Komponenten auseinander oder zusammen. In beiden Fällen wird der Ausführungsort der Anwendung verändert. Im ungünstigsten Fall sind beide Kräfte 1) und 2) annähernd gleich groß, sodass Auseinanderziehen und Zusammenführen von Anwendungen bzw. Anwendungskomponenten alternieren. Den möglichen Kräften, resultierend aus Kosteneinsparungen durch Anwendungsverschiebung aus 1) und 2), stehen Kosten für die Platzierungsänderung gegenüber. Auch aus diesen Kosten resultiert eine Kraft, die vor einer Platzierungsänderung überwunden werden muss. Sie wirkt damit einer Platzierungsänderung entgegen.

In Pseudocode ausgedrückt bedeutet dies, dass für jede Anwendung, die von einem Broker ausgeführt wird, die Kraft bzw. die Kräfte zu ermitteln sind, welche auf die Platzierung einwirken. Daraufhin wird entschieden, ob eine bestehende Platzierung verändert werden muss. Zunächst startet für jede Anwendung auf dem Broker ein Koordinatorprozess (siehe Abbildung 3.10), welcher für die Informationssammlung und Platzierungsentscheidung verantwortlich ist.

```
11 for_all Applications a_i with i==0 until max_i
12   do begin {
13     coordinator_process(Broker this, Application a_i)
14 }
```

Abbildung 3.10: Start des Koordinators für jede laufende Anwendung a $(a_0,\ldots,a_i)$ auf dem Broker $currentBroker$

Der Koordinatorprozess ist nun für die Ermittlung der Kräfte und das Treffen der Platzierungsentscheidung zuständig (siehe Abbildung 3.11). Die Kräfteermittlung erfolgt im Nachgang der Kostenermittlung, d.h. zunächst ist die Ermittlung von Ausführungs-, Weiterleitungs- und Speicherkosten durchzuführen. Die Ermittlung der Kosten erfolgt laufend, während die Auswertung der Kräfte zu diskreten Zeitpunkten stattfindet. Die Platzierungsänderung einer Anwendung kann nur in Richtung der jeweiligen Nachbarbroker ausgeführt werden. In diese Richtungen können auch nur die ermittelten Kräfte wirken, daher werden die Kräfte je Richtung gebündelt und je Richtung ermittelt. Dies bedeutet auch,

```
11  public void coordinator_process(Broker currentBroker, Application a_i) {
12    timer t;
13    List<Value> cost_st = new List<Value>(this.getNumberNeighborBroker());
14    List<Value> cost_fw = new List<Value>(this.getNumberNeighborBroker());
15    List<Value> cost_exe = new List<Value>(this.getNumberNeighborBroker());
16    setTimer(t);
17
18    while active(){
19      cost_st = determineStorageCosts(this, a_i);
20      cost_fw = determineForwardCosts(this, a_i);
21      cost_exe = determineExecutionCosts(this, a_i);
22
23    if t expired {
24      evaluateCosts(a_i,cost_st,cost_fw,cost_exe);
25    }
26  }
```

Abbildung 3.11: Grober Ablauf des Koordinatorprozesses für jede laufende Anwendung a (a_0,...,a_i) auf dem Broker $currentBroker$

die ermittelten Kräfte in einer geeigneten Form zurückzugeben. Eine Möglichkeit dieser Darstellung ist die eindimensionale Matrix, wobei die Anordnung der Elemente eindeutig erkennbar sein muss, bspw. durch eine Liste mit einem festgelegten Ablaufplan in Bezug auf die Nachbarbroker. Dazu existiert auf jedem Broker eine Funktion $getNumberCurrentNeigbors()$ (siehe Abbildung 3.12), welche die Anzahl der Nachbarbroker ermittelt. Diese Funktion greift auf eine weitere Funktion zurück, die eine einheitliche Reihenfolge bzw. Besuchsreihenfolge der Nachbarbroker festlegt (siehe Abbildung 3.13). Darüber hinaus wird eine andere Funktion genutzt, welche einen auf einen bekannten Nachbarbroker folgenden Broker ausgibt (siehe Abbildung 3.14)

```
30 // Returns number of current neighboring broker
31 public int getNumberNeighborBroker() {
32
33     ...
34
35     return this.numberOfCurrentNeighbors();
36 }
```

Abbildung 3.12: Rückgabe der Anzahl von Nachbarbrokern eines Brokers

Zugwirkung nach außen - Expansion und Verschiebung. Durch Vergleich von Ausführungskosten $C_{A_z ex}$ und Weiterleitungskosten $C_{A_z fw}$ einer Anwendung an ihrem derzeitigen Ausführungsort und alternativer Standorte wird eine Kraft ermittelt. Daher ergibt sich für die Kostenbezeichnung ein weiterer Index, der den

```
40 // Returns a fixed list of neighboring broker of the current broker
41 public List<Broker> getNeighborList(){
42
43   ...
44
45   return this.getBrokerList();
46 }
```

Abbildung 3.13: Rückgabe einer Liste von Nachbarbrokern

```
50 // Returns next neighbor broker from actual neighbor
51 public Broker getNextNeighborBroker(Broker actualNeighbor){
52
53   ...
54
55   // Calls internal function to present next neighbor
56   // Note: If actual neighbor is null,
57   // the function returns first neighbor if exists
58   return this.getBrokerList().getNextNeighbor(this,actualNeighbor);
59 }
```

Abbildung 3.14: Rückgabe eines Nachbarbrokers als Nachfolger eines bereits bekannten Brokers

Ausführungsort (Broker) B kennzeichnet ($C_{A_z Bex}$ bzw. $C_{A_z Bfw}$). Der Vergleich der Ausführungskosten erfolgt, wie im Abschnitt 3.5.4 beschrieben, als Quotient aus tatsächlichem Ressourcenverbrauch (ermittelt) und vorhandenen Ressourcen (ist ebenfalls bekannt). Die Berechnung wird unter der Annahme durchgeführt, dass durch einen Wechsel des Ausführungsortes keine wesentliche Änderung des tatsächlichen Ressourcenverbrauchs eintritt. Durch die relative Berechnung der Kosten sind diese bereits gewichtet und werden äquivalent auf die Kräfte umgerechnet. Da die Anwendung am alternativen Ausführungsort nicht ausgeführt wird, lassen sich die Ausführungskosten am alternativen Standort nicht mit der gleichen Methode berechnen wie am bisherigen Ausführungsstandort. Bei einem Wechsel ist vielmehr interessant, wie sich die Ausführungskosten am alternativen Ausführungsort durch die zusätzlichen Kosten verändern würden, die Berechnung ist damit mehr eine Schätzung von möglichen Kosten. Daher werden als Bezugsgröße nicht die insgesamt vorhandenen Ressourcen, sondern die noch verfügbaren Ressourcen gewählt. Damit wird auch sichergestellt, dass überhaupt ausreichend Ressourcen vorhanden sind.

Es ergibt sich eine Ausführungskraft P_{exe} in Abhängigkeit der Kostensituation der Anwendung A_z, der tatsächlichen Platzierung B_a und der alternativen Platzierung B_{alt}: $P_{exeA_z B_a B_{alt}} = C_{A_z B_{alt} fw} - C_{A_z B_a fw}$. Die Ausführungskosten am alternativen Ausführungsort erfolgen als Schätzung aufgrund des bisherigen Ressourcenverbrauchs $RC_{A_z B_a}$ im Vergleich zu den noch verfügbaren Ressourcen am

alternativen Ausführungsort $RA_{A_z B_{alt}}$. Es ergibt sich für die Anwendung A_z auf dem Broker B_{alt} eine Kostenabschätzung $C_{A_z B_{alt} fw} = RC_{A_z B_a} / RA_{A_z B_{alt}}$. Die Kostenermittlung erfolgt gruppiert nach dem Schema $\sum_{i=1}^{n} RC_{A_z} B_a i / RA_{A_z B_{alt}} i$ mit den Gruppen G_i von 1 bis n freilich unter der Voraussetzung, dass die jeweils noch freien Ressourcen $RA_{A_z B_{alt}}$ ermittelt wurden. Auf diese Weise liegen die alternativen Ausführungskosten je Richtung vor. Zur Berechnung der alternativen Ausführungskosten besitzen die jeweiligen (Nachbar-) Broker eine entsprechende Funktion (siehe Abbildung 3.15). Darüber hinaus werden am ausführenden

```
60 // Determines fictional execution costs of application a_i at broker
61 public Value determineFictionalExecutionCosts(Application a_i){
62   // gets a list with all resources required by application a_i
63   List<Ressources> resList = new List<Ressources>();
64   resList = a_i.getRequirements();
65
66   // gets  available resources from broker regarding
67   // requirements of application a_i
68   List<Ressources> resList = new List<Ressources>();
69   resList = this.getRessources(resList);
70
71   //calculate virtual execution costs
72   return this.calculateVirtualCosts(resList);
73 }
```

Abbildung 3.15: Berechnung fiktiver Ausführungskosten einer Anwendung a_i an einem Broker

Broker auch die Kosten ermittelt, die bei der tatsächlichen Ausführung der Anwendungen entstehen (siehe Abbildung 3.16). Ergibt sich bei der Aufsummierung der alternativen und tatsächlichen Kosten ein positiver Saldo, entsteht eine Kraft, welche die Anwendung in Richtung der alternativen Platzierung zieht, bei negativem Saldo wirkt eine Kraft, welche die Anwendung an ihrem bisherigen Ausführungsort festhält. Für jeden Nachbarbroker wird, unter der Voraussetzung, dass eine Anwendungsplatzierung überhaupt möglich ist, (vgl. mit den Anforderungen für die initiale Platzierung in Abschnitt 3.4.3) die Summe der alternativen Ausführungskosten bestimmt. Ausgedrückt in Programmcode bedeutet dies, eine Aufsummierung der alternativen und tatsächlichen Kosten findet, getrennt nach Richtung des betreffenden Nachbarbrokers, statt. Die Berechnung der Kräfte/Kosten erfolgt zunächst für die Ausführungskosten. Der derzeit ausführende Broker ruft die Berechnung der fiktiven Ausführungskosten entsprechend auf (siehe Abbildung 3.17). Die Ermittlung der unterschiedlichen tatsächlichen und fiktiven Ausführungskosten findet innerhalb des Koordinatorprozesses in der Methode $evaluateCosts(cost_{st}, cost_{fw}, cost_{exe})$, innerhalb des ausführenden Brokers statt (siehe Abbildung 3.18).

```
70 // Determines actual execution costs of an application a_i running at broker
71 public Value determineActualExecutionCosts(Application a_i){
72   // gets a list with all resources used by application a_i
73   List<Ressources> reqUseList = new List<Ressources>();
74   reqUseList = a_i.getRequirements();
75
76   // gets  all available resources from broker regarding
77   // requirements of application a_i
78   List<Ressources> resList = new List<Ressources>();
79   resList = this.getRessources(resList);
80
81   // calls internal function at broker to calculate
82   // real execution costs
83   return this.calculateExecutionCosts(reqUseList, resList);
84 }
```

Abbildung 3.16: Berechnung der tatsächlichen Ausführungskosten einer Anwendung a_i an einem Broker

```
80
81 // Determines the the fictional execution costs for application a_i
82 // for all other neighboring broker
83 public List<Value> determineExecutionCosts (Application  a_i) {
84
85     List<Value> cost_exe =
86          new List<Value>[currentBroker.
87                          getNumberNeighborBroker(this, currentBroker)]();
88
89          currentNeighborBroker = null;
90
91   while (∃ this.getNextNeighbor(currentNeighborBroker)) {
92
93          cost_exe.add(this.getNextNeighbor().
94                              determineFictionalExecutionCosts(a_i));
95
96   }
97
98   return cost_exe;
99 }
```

Abbildung 3.17: Aufruf zur Berechnung fiktiver Ausführungskosten einer Anwendung a_i an Nachbarbrokern

```
10 // Evaluates the cost situation, determines resulting forces
11 public void evaluateCosts(Application a_i, Value cost_st, Value cost_fw,
12
13   List<Force> force_exe = new List<Value>(this.getNumberNeighborBroker());
14
15   ...
16
17   // Determine execution cost for application a_i
18   Value cost_exe = this.determineActualExecutionCosts(a_i);
19
20   // Determine force derived by execution costs for each
21   // neighbor broker
22   while (cost_exe.hasNext()){
23         force_exe.add(new Force(cost_exe.getNext()-cost_exe, a_i,
24                                                 this, this.getNextNeighbor());
25   }
26 }
```

Abbildung 3.18: Auswertung der Kosten und Ermittlung der Kräfte, Teil 1: Ermittlung der Ausführungskraft

Bei der Berechnung der Weiterleitungskosten kann auf die Abschätzung der alternativen Kostensituation verzichtet werden, da nur die exklusiven Nachrichtenströme ausgewertet werden müssen und bei alternativen Platzierungsorten durch eine Umverteilung möglicherweise weitere Abhängigkeiten bestehen. Die Ermittlung der Kraft aus den Weiterleitungskosten erfolgt daher exakt wie bei der Bestimmung der Weiterleitungskosten $C_{A_z fw}$, wobei für die eindeutige Bezeichnung auch hier der Ausführungsort hinzugefügt wird. Es ergibt sich $C_{A_z B_a fw}$ und damit äquivalent dazu die Kraft $P_{fwA_z B_i}$. Dabei wird die resultierende Kraft bestimmt, d.h. die Kraft bzw. die Kräfte, welche die anderen Kräfte dominieren. Neben dem Betrag der Kraft ist auch die Richtung der Kraft von Interesse (Angabe des ermittelten Nachbarbrokers). Die Ermittlung der resultierenden Kraft erfolgt, indem zunächst für jede Anwendung einzeln der Saldo der ein- und ausgehenden Nachrichten bestimmt wird, d.h. für jede Anwendung A_z wird je nach Umfeld (Anzahl der Nachbarbroker m) zunächst die Differenz aller ein- und ausgehenden Ereignisströme berechnet, d.h. für alle $m_1, \ldots, m_n$ wird die Summe aller ausgehenden Ereignisströme $C_{A_z B_a fwoutm_i} = \sum_{i=1}^{n} C_{A_z B_a fwoutm_i}$ und die Summe der eingehenden Nachrichten $C_{A_z B_a fwinm_i} = \sum_{i=1}^{n} C_{A_z B_a fwinm_i}$ ermittelt. Daraufhin wird die Differenz beider Summen gebildet, also $C_{A_z B_a fwinm_i} = C_{A_z B_a fwinm_i} - C_{A_z B_a fwinm_i}$, dies ergibt die Kraft

$$P_{fwA_z D_{b_i}}.$$

Dies zeigt der Pseudocode in Abbildung 3.19.

Vorgeschaltet dazu findet die Ermittlung der Weiterleitungskosten statt (siehe Abbildung 3.20), wozu seitens der Broker Hilfsfunktionen, wie die Ermittlung der exklusiven Ereignisströme je Anwendung am Broker (siehe Abbildung 3.21) und die Ermittlung der Summe der Ereignisse je Nachrichtenstrom (siehe Abbildung 3.22), genutzt werden.

```
10 // Evaluates the cost situation, determines resulting forces
11 public void evaluateCosts(Application a_i, Value cost_st,
12                                              Value cost_fw, Value cost_exe) {
13   ...
14
15   List<Force> force_fw = new List<Force>(this.getNumberNeighborBroker());
16
17   ...
18
19   // Determine forwarding cost for application a_i
20   // for each neighboring direction
21
22   cost_fw = this.determineForwardingCosts(a_i);
23
24
25   // Determine force derived by forwarding costs for each
26   // neighbor broker, here: equal mapping between force and costs
27    while (cost_fw.hasNext()){
28      force_fw.add(new Force(cost_exe.getNext(), a_i,
29                                              this, this.getNextNeighbor());
30   }
31 }
```

Abbildung 3.19: Auswertung der Kosten und Ermittlung der Kräfte, Teil 2: Ermittlung der Weiterleitungskraft

```
70 // Determines costs resulted by forwarding events
71 public List<Value> determineForwardCosts(Broker broker, Application a_i);
72   List<Value> cost_fw = new List<Value>(broker.getNumberNeighborBroker());
73   Value cost_fw_input, costfw_output;
74
75   for all broker.getNextNeighbor() {
76   // for each neighboring broker
77   if ( broker.getNextNeighbor().hasNext()) {
78
79       Broker n_broker = broker.getNextNeighbor();
80
81       // Determines balance between in- and outgoing event streams
82       // per direction (neighboring broker)
83
84       // a) incoming event streams from neighbor broker
85       //
86       // Determines amount of incoming event streams per direction
87       //  (neighboring broker)
88       //
89       // 1) Get all incoming event streams per application a_i
90       //    (get in- and outgoing event streams): a_i.getEventStreamsIn()
91       // 2) Determines if input event streams are exclusive:
92       //                                        getExclusiveEventStreams()
93       // 3) Determines amount of all exclusive input streams:
94       //                                broker.determineAmountEventStreams()
95       // 4) Put amount of all input events into single value
96       cost_fw_input =
97            add(broker.determineAmountEventStreams(
98                 broker.getExclusiveEventStreams(a_i.getEventStreamsIn());
99
100      // Determines amount of outcoming event streams per neighbor broker
101      //
102      // 1) Get all outcoming event streams per application a_i
103      //        (get outgoing event streams): a_i.getEventStreamsOut()
104      // 2) Determines if output event streams are exclusive:
105      //                                        getExclusiveEventStreams()
106      // 3) Determines amount of all exclusive output streams:
107      //                                broker.determineAmountEventStreams()
108      // 4) Put amount of all output events into list: cost_fw_output.add()
109      cost_fw_output = broker.determineAmountEventStreams(
110         broker.getExclusiveEventStreams(a_i.getEventStreamsOut());
111
112      // Determine balance of in- and output events and write
113          // balance into list cost_fw
114          cost_fw.add(cost_fw_input - cost_fw_output);
115    }
116  }
117 }
```

Abbildung 3.20: Berechnung der tatsächlichen Weiterleitungskosten von Ereignissen bezüglich einer Anwendung a_i auf einem Broker

```
80 public List<Stream> getExclusiveEventStreams(List<Stream> allStreams) {
81
82   List<Stream> exclusiveStreams = new List<Stream>();
83
84   return exclusiveStreams;
85 }
```

Abbildung 3.21: Ermittlung der exklusiven Ereignisströme je Anwendung auf dem Broker

```
90 // Method to determine the number of events given by a list of event streams
91 public double determineAmountEventStreams(List<Stream> streamList) {
92
93   double amount = 0;
94
95   while (streamList.hasNext()) {
96         // calls internal method that determines internally the number
97         // of events per stream
98     amount = this.calculateNumberEventsPerStream(streamList.next());
99
100  }
101
102  return amount;
103 }
```

Abbildung 3.22: Ermittlung der Anzahl von Ereignissen je Ereignisstrom

```
5  public class Application {
6
7    // Returns all input streams of an application
8    public List<Stream> getEventStreamsIn(){
9        List<Stream> inputStream = new List<Stream>();
10       if (this.inputStreams().hasNext())
11         inputStream.addAll(this.getInputStreams());
12   }
13   // Returns all input streams of an application
14   public List<Stream> getEventStreamsOut() {
15       List<Stream> outputStream = new List<Stream>();
16       if ( this.outputStreams())
17         inputStream.addAll(this.getInputStreams());
18   }
19
20   // Returns a list of related application parts
21   // i.e. former application parts,
22   // duplicated applications...
23   public List<Application> getRelatetedApps() {
24           // Calls internal method to return related applications
25       return this.getRelatedApplications();
26   }
27 }
```

Abbildung 3.23: Bereitstellung der genutzten ein- und ausgehenden Ereignisströme

Zusätzlich werden Methoden der Klasse Application genutzt, welche in Form von Listen die ein- und ausgehenden Ereignisströme bereitstellen (siehe Abbildung 3.23). Bei einem positiven Ergebnis überwiegen die eingehenden Nachrichtenströme, bei negativem Ergebnis überwiegen dagegen die ausgehenden Ströme je Richtung. Nach der Ermittlung der Beträge der jeweiligen Kraft je Kommunikationsrichtung erfolgt die Bestimmung der dominanten Kraft. Dazu muss der Betrag einer Kraft größer sein als die Summe der Beträge aller anderen Kräfte. Intern wird die Kraft ($Force$) als Konglomerat der Informationen über den Betrag der Kraft ($Value$), der Anwendung ($Application$) sowie der beteiligten Broker dargestellt.

Es bietet sich zudem an, die bereits gerichtet vorliegenden Kräfte resultierend aus den Ausführungskosten und den Weiterleitungskosten

$$P_{exe-fwA_zD_i} = P_{exeA_zB_aB_i} + P_{fwA_zB_i}$$

je Richtung i zusammenzufassen. Dies geschieht innerhalb der Methode $evaluateCosts(Application\ a\_i,\ Value\ cost\_st,\ Value\ cost_fw,\ Value\ cost\_exe)$ des Brokers (siehe Abbildung 3.24).

```
5  // Evaluates the cost situation, determines resulting forces
6  public void evaluateCosts(Application a_i, Value cost_st, Value cost_fw,
7
8   ...
9
10  List<Force> force_exe_fw = new List<Force>(this.getNumberNeighborBroker());
11  List<Force> force_exe_fw_dom = new List<Force>();
12
13  ...
14
15  // Combine force_exe and force fw
16  while (this.getNeigborList.hasNext()){
17    force_exe_fw.add(new Force(force_exe.getNext().getValue() +
18                  force_fw.getNext().getValue(), a_i, this,
19                                          this.getNextNeighbor());
20  }
21
22  // Determine the dominating force
23  force_exe_fw_dom.addAll(this.determineDominatingForce(force_exe_fw));
24
25 }
```

Abbildung 3.24: Zusammenfassung der Weiterleitungs- und Ausführungskräfte

Sind die Geräte ähnlich ausgestattet und ähnlich ausgelastet, spielen die Ausführungskosten bei der Ermittlung der Kraft P_{exe-fw} eine eher unbedeutende Rolle und sind in diesem Zusammenhang vernachlässigbar, sodass

$$P_{exe-fwA_zB_i} = P_{fwA_zB_i}$$

gilt. Was nun folgt, ist die Ermittlung der dominanten Kraft $P_{exe-fwdomA_zB_i}$. Für jede mögliche Richtung j mit $j = 1 \ldots n$ wird Folgendes ermittelt: Wenn $|P_{exe-fwA_zB_j}| > (\sum_{i=1}^{n} |P_{exe-fwA_zB_i}|) - |P_{exe-fwA_zB_j}|$, dann ist

$$P_{exe-fwA_zD_{b_j}}$$

eine dominante Kraft. Wenn keine Kraft die jeweils anderen dominiert, lässt sich keine dominante Kraft ableiten. Genauso ist es aber auch möglich, dass nicht nur keine oder eine dominante Kraft ermittelt werden kann, sondern mehrere gleich große Kräfte. Die Ermittlung der dominierenden Kraft ist in Abbildung 3.24 durch den Aufruf der Methode $this.determineDominatingForce(force\_exe\_fw)$; auf der Ebene des Brokers abgebildet, die Methode selbst zeigt Abbildung 3.25. Sobald mehrere Kräfte aus unterschiedlichen Richtungen auf eine Anwendung

```
100 // Determines if one force (element of the input list) dominates
101 // all other forces
102 public List<Force> determineDominatingForce(List<Force> force_exe_fw) {
103   List<Force> force_dom = new List<Force>();
104
105   ...
106
107   while (force_exe_fw.hasNext()){
108
109         Force force = force_exe_fw.next()
110    // Sums up the forces of all other directions except current force
111    if (force.getValue() >= this.sumUpForcesExcept(force){
112     force_dom.add(force);
113    }
114  }
115 }
```

Abbildung 3.25: Ermittlung einer Liste dominanter Kräfte

einwirken, sind je nach Architektur der Anwendung eine Verschiebung im Ganzen oder eine Aufspaltung der Anwendungen in Komponenten und eine separate Verschiebung möglich. Es wird eine Kraft bzw. eine Liste mit Kräften zurückgegeben, welche die jeweils anderen Kräfte dominiert. Ebenfalls in die Liste werden die Kräfte aufgenommen die, falls keine dominierende Kraft vorliegt, mit gleichem Betrag auf die Anwendungsplatzierung einwirken.

Zugwirkung nach innen - Zusammenführung. Aufgrund von Kosten, verursacht durch doppelte Datenhaltung von Ereignissen, Zwischenergebnissen und Zuständen (Speicherkosten $C_{A_z st}$) entsteht eine Kraft, die ehemals zusammenhängende Anwendungskomponenten wieder zusammenzieht. Zur Berechnung der als $P_{A_z st}$ bezeichneten Kraft müssen zunächst die doppelt hinterlegten Daten und anschließend die daraus entstehenden Kosten ermittelt werden. Durch die Zusammenlegung der Komponenten ließen sich an einem Gerät Speicherkosten einsparen, wodurch eine Kraft $P_{A_z st}$ generiert wird. Die Ermittlung der Kraft erfolgt dabei in folgenden Schritten:

1. Ermittlung der (ehemals) zusammenhängenden Komponenten und Ermittlung der mehrfach gespeicherten Informationen
2. Ermittlung der durch Mehrfachspeicherung entstandenen Kosten
3. Ermittlung der einsparbaren Kosten und damit der resultierenden Kraft.

Die (1) Ermittlung der ehemals zusammenhängenden Anwendungskomponenten lässt sich bspw. durch eine entsprechende Gestaltung der Anwendungs-ID mit vergleichsweise wenig Aufwand unter der Nutzung von Publish/Subcribe-Primitiven, ähnlich dem Verfahren zur initialen Platzierung, realisieren. Dabei werden Informationen durch Notifikationen ausgetauscht, deren Interesse durch die jeweiligen IDs gefiltert werden. Lassen sich exakt gleiche Anwendungen bzw. Anwendungskomponenten ausfindig machen, d.h. sie verhalten sich exakt gleich und bearbeiten den selben Ereignisraum, ist die Bestimmung der mehrfach gespeicherten Informationen abgeschlossen, da die Nutzung der Speicherkosten $C_{A_z st}$ möglich ist. Ansonsten müssen für eine exakte Bestimmung der mehrfach gespeicherten Daten zwischen den ausführenden Geräten zusätzliche Informationen über die betroffenen, mehrfach hinterlegten Daten ausgetauscht werden. Dies umfasst dabei die relevanten Informationen über die genutzten Ereignistypen, den Ereignisraum und die Frage, ob die Informationen exklusiv genutzt werden. Dafür bietet sich ein Verfahren an, welches die benötigten Daten komprimiert übermittelt. Die (3) Kostenbewertung erfolgt entweder anhand der Kostenermittlung für die Speicherkosten $C_{A_z st}$, oder indem die genutzten Ressourcen für die ausschließlich überschneidenden Daten (R_{memA_z}) ins Verhältnis zu den verfügbaren Ressourcen (DR_{memx}) mit R_{memA_z}/DR_{memx} gesetzt werden. Die aus diesen unterschiedlichen Kostensituationen resultierende Kraft bestimmt sich wiederum als Differenz zwischen derzeitiger Kostensituation und alternativer Platzierung. So werden zunächst die derzeitigen Kostensituationen an den bisherigen Standorten ermittelt. Dabei können, auch bei absolut gleicher Belastung der Speicher, durch das Verhältnis zum verfügbaren Speicher, unterschiedliche Kosten entstehen. Im zweiten Schritt wird durch einen Vergleich der Beträge beider Kosten das Gerät zur Ausführung bestimmt, dass die geringsten Kosten verursacht. Bei zwei Brokern b_1 und b_2 (Ausführungsorte) ergibt sich dann folgender Ablauf:

$$C_{A_z st b_1} = R_{memA_z b_1}/DR_{memx b_1}$$

und

$$C_{A_z stb_2} = R_{memA_z b_2} / DR_{memxb_2}.$$

Der bevorzugte Ausführungsort b_{best} ergibt sich aus

$$b_{best} = \begin{cases} b_1, & \text{falls } |C_{A_z stb_1}| < |C_{A_z stb_2}| \\ b_2, & \text{sonst.} \end{cases}$$

Die Kraft $P_{A_z stb_1 b_2}$, die daraus resultiert, ist gleich dem Betrag des Ausführungsortes, der nicht zum Zuge gekommen ist. Es gilt daher

$$P_{A_z stb_1 b_2} = \begin{cases} |C_{A_z stb_2}|, & \text{falls } |C_{A_z stb_1}| < |C_{A_z stb_2}| \\ |C_{A_z stb_2}|, & \text{sonst.} \end{cases}$$

Die Berechnung der Speicherkosten ist ähnlich der Berechnung der fiktiven Ausführungskosten, hier vorgeschaltet wurde allerdings die Ermittlung von ehemals zusammenhängenden Komponenten. Dies erfolgt innerhalb der Anwendung selbst (siehe Abbildung 3.23).

Die Kräfteermittlung erfolgt durch eine Methode des Brokers (siehe Abbildung 3.26), wobei hier der Aufruf der Kostenermittlung der Speicherkosten vom Broker aus erfolgt, der derzeit die betrachtete Anwendung ausführt (siehe Abbildung 3.27). Diese Methode umfasst wiederum die Kostenermittlung der Speicherkosten von Anwendungen, die mit der betrachteten Anwendung in Verbindung stehen (siehe Abbildung 3.28).

Widerstand - Gegengewicht zu Platzierungsoperationen. Der Aufwand und damit die Kosten für die Versendung einer Anwendung ist/sind dabei grundsätzlich gleich, egal in welche Richtung die Anwendung verschoben werden soll. Daher wird die Kraft, die hier gegen die Zugwirkung nach außen und innen wirkt, durch die Platzierungskosten $C_{A_z plac}$ bestimmt. Sie kann als Dämpfung bzw. Gegenkraft gesehen werden. Würden die resultierenden Zugkräfte regelmäßig Veränderungen der Anwendungsplatzierung auslösen, also ohne Gegenkraft einwirken, wären ständige Oszillationen von Verschiebungen nach außen und innen möglich. Gegen solche Entwicklungen stemmt sich die Dämpfung, die vor einer Anwendungsoperation zunächst überwunden werden muss. Der Widerstand $P_{A_z res}$ mit $P_{A_z res} = C_{A_z plac}$ nimmt neben seiner Funktion als Widerstand auch die Funktion der Adaptivitätssteuerung wahr, da bei einem geringen Widerstand auch nur geringere Schwellwerte für eine Anwendungsplatzierungsänderung erreicht werden müssen. Die Platzierungskosten sind dabei abhängig vom Aufwand auf Seiten des ausführenden Brokers sowie des Aufwands auf Seiten des Platzierungsziels. Ähnlich wie bei der Ermittlung der Ausführungskosten erfolgt auch hier die Berechnung der Platzierungskosten. Die Ermittlung der Kosten erfolgt anhand des Codes, dargestellt in den Abbildungen 3.29 und 3.30. Die Ermittlung der daraus resultierenden Kraft findet sich innerhalb der Methode zur Kräftebestimmung wieder (siehe Abbildung 3.31).

```
100 // Evaluates the cost situation, determines resulting forces
101 public void evaluateCosts(Application a_i, Value cost_st, Value cost_fw,
102                                                          Value cost_exe) {
103   ...
104
105   List<Force> force_st = new List<Force>(this.getNumberNeighborBroker());
106   List<Value> cost_st;
107
108   ...
109
110   // Determine storage cost for application a_i
111   // for each neighboring direction
112
113   cost_st = this.determineStorageCosts(a_i);
114
115   // Determine force derived by storage costs for each
116   // neighbor broker, here: equal mapping between force and costs
117    while (cost_st.hasNext()){
118      force_fw.add(new Force(cost_st.getNext(), a_i, this,
119                                              this.getNextNeighbor());
120   }
121 }
```

Abbildung 3.26: Ermittlung der Speicherkraft inklusive Aufruf der Kostenberechnung

```
110 // Determines storage costs of an application a_i running at this broker
111 //   and calls the determination of storage costs at neighbor broker for
112 //   related applications
113 public List<Value> determineStorageCosts(Application a_i){
114   // gets a list of related applications from application a_i
115   List<Application> relatedAppsList = new List<Application>();
116   List<Value> = costsStorageNeighbor = new List<Value>();
117
118
119   if (¬∅ relatedAppsList) {
120     while ( this.getBrokerList().hasNext()){
121       // calls function determining the storage costs of related
122       // applications of a_i and neighbor broker
123       costsStorageNeighbor.add(this.getBrokerList().
124                                   next().determineFicStorageCosts(a_i));
125     }
126
127   }
128
129   return costsStorageNeighbor;
130 }
```

Abbildung 3.27: Ermittlung der Speicherkosten einer Anwendung, die auf dem Broker ausgeführt wird und Aufruf der Berechnung der (fiktiven) Kosten am Nachbarbroker

```
120 // Determines storage costs of application a_i related
121 // components executed at the current broker
122 public Value determineFicStorageCosts(Application a_i){
123   Value value = new Value();
124
125   // gets a list of all related application of a_i
126   List<Application> relAppList = new List<Application>();
127   relAppList = a_i.getRelatetedApps();
128
129   // call internal method to determine related
130   // applications executed at the current broker
131   // and determines costs
132
133   while (relAppList.hasNext()) {
134     Application a_n;
135     a_n = relAppList.next();
136     if (this.executes(a_n))
137       value.add(this.getStorageValue(a_n));
138   }
139
140   return value;
141 }
```

Abbildung 3.28: Ermittlung der Speicherkosten inklusive Aufruf der Kostenberechnung an Nachbarbrokern

```
150 // Determines placement costs of an application a_i
151 // running at current broker and supposed to
152 // leave towards neighbor broker
153 // Returns a list of placement costs for neighbor broker
154 public Value determinePlacementCosts(Application a_i){
155
156   List<Value> list_placement_costs = new List<Value>(
157                                 this.getNumberNeighborBroker());
158   Value cost;
159
160   // call internal function to calculate real execution costs
161   cost = this.calculatePlacementCosts(a_i);
162
163   while (this.getNeighborList.hasNext()){
164     list_placement_costs.add(cost + this.getNeighborList().
165           getNext().determineFictionalPlacementCosts(Application a_i));
166
167   }
168
169   return list_placement_costs;
170 }
```

Abbildung 3.29: Ermittlung der Platzierungskosten am bisher ausführenden Broker und Aufruf der Kostenberechnung an Nachbarbrokern

```
170 // Determines placement costs if application a_i should
171 // be executed at this broker
172 public Value determineFictionalPlacementCosts(Application a_i){
173   // gets a list with all resources required by application a_i
174   List<Ressources> resList = new List<Ressources>();
175   resList = a_i.getRequirements();
176
177   // gets  available resources from broker regarding
178   // requirements of application a_i
179   List<Ressources> resList = new List<Ressources>();
180   resList = this.getRessources(resList);
181
182   //calculate virtual placement costs
183   return this.calculateVirtualPlacementCosts(resList);
184 }
```

Abbildung 3.30: Ermittlung der Platzierungskosten am zukünftig ausführenden Broker

```
190 // Evaluates the cost situation, determines resulting forces
191 public void evaluateCosts(Application a_i, Value cost_st, Value cost_fw,
192
193   List<Force> force_place = new List<Value>(this.getNumberNeighborBroker());
194
195  ...
196
197   // Determine placement cost for application a_i
198   List<Value> cost_place = this.determinePlacementCosts(a_i);
199
200   cost_place_list = this.determinePlacementCosts();
201
202   // Determine force derived by placement costs for each
203   // neighbor broker
204   while (cost_place.hasNext()){
205         cost_place = cost_place_list.next();
206         force_place.add(new Force(cost_place.getValue(),
207                                   a_i, this, this.getNextNeighbor());
208   }
209 }
```

Abbildung 3.31: Ermittlung der Gegenkraft auf Grundlage der Platzierungskosten am zukünftig ausführenden Broker

Zusammenfassung - Einwirkende Kräfte. Abbildung 3.32 zeigt das Einwirken von Kräften auf eine Anwendung. Diese besteht aus insgesamt drei Komponenten, wobei zwei Komponenten einer zentralen Komponente Teilergebnisse liefern. Die beiden Eingangsströme werden exklusiv durch die linke Teilanwendung bzw. die rechte Teilanwendung bearbeitet. Beide Teilanwendungen liefern der zentralen Komponente Teilereignismuster, die sie weiterverarbeitet und als neues Ereignismuster veröffentlicht. Da zwischen den Teilkomponenten keine weiteren Abhängigkeiten bestehen, lassen sich nach einer Dekomposition die beiden zuliefernden Komponenten und die zentrale Komponente individuell platzieren.

Mögliche Einsparungen durch eine Platzierungsanpassung bei Weiterleitungs- und Ausführungskosten ziehen die Gesamtanwendung auseinander - die Anwendung wird in Komponenten aufgespaltet und diese unabhängig voneinander platziert (Expansion). Demgegenüber wirken mögliche Einsparungen von Speicher-

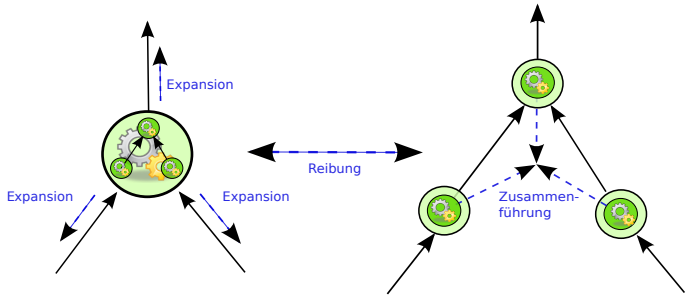


Abbildung 3.32: Einwirkende Kräfte auf die Detektorplatzierung

kosten auf die ehemals verbundenen Anwendungskomponenten ein, so dass diese wieder auf einem Gerät ausgeführt werden (Zusammenführung). Dieser Prozess von Expansion und Zusammenführung kann beliebig oft stattfinden. Um ein Oszillieren zu verhindern, wirkt die Reibung als Hemmnis bei der Ausführung von Expansion und Zusammenführung.

Gegenseitige Abbildung von Kosten und Kräften

In den bisherigen Ausführungen wurden in Abschnitt 3.5.4 sowohl das kostenseitige Ziel auf globaler und regionaler Ebene, als auch das den Platzierungsentscheidungen zu Grunde liegende Kräftemodell vorgestellt. Während Kosten, bis auf die Speicherkosten, grundsätzlich unabhängig von anderen Anwendungen anfallen, sind die wirkenden Kräfte vielmehr ein Ausdruck der unterschiedlichen Kostensituationen, also ein Vergleich zweier Kostensituationen.

Die Zielfunktion zur Optimierung der Gesamtkostensituation wurde auf eine Zielfunktion für eine einzelne Anwendung an einem Ausführungsort heruntergebrochen. Die Abbildung der Kostensituationen erfolgt auf Kräfte, welche auf die Anwendungsplatzierung einwirken. Dabei gilt lokal wie auch global, dass je

geringer die Kosten sind, desto kleiner sind die resultierenden Kräfte. Ein Ausgleich von Kräften führt zu einem Abbau von Kostenunterschieden. Der Abbau der Kostenunterschiede bewirkt eine Verschiebung der Anwendungen zu einem kostengünstigeren Ausführungsort. Da der abgebende Ausführungsort nun die Anwendung nicht mehr ausführt, spart dieser Kosten. Der aufnehmende Ausführungsort hat einen höheren Kostenaufwand als vor der Ausführung einer weiteren Anwendung. Da die Ausführung im Vergleich zur Situation zuvor jedoch kostengünstiger ist, wird das globale Ziel erreicht. Auf genau diesem Vergleich zweier Kostensituationen basiert das Kräftemodell, es bezieht sich auf die Kosten je Anwendung und es werden genau die Vergleiche angestellt, mit denen die Kostensituation des Gesamtnetzwerks verbessert werden kann. Dabei wirken drei unterschiedliche Kräfte, von denen die Weiterleitungskosten den höchsten Einfluss auf die Kosteneffizienz der Anwendungsausführung der Gesamtheit der Anwendungen haben.

3.5.5 Selbstorganisierende adaptive Anwendungsplatzierung

Nachdem sowohl die vier Basisoperationen als auch die durch die Anwendungsausführung ausgelösten Kosten und die daraus resultierenden Kräfte vorgestellt wurden, wird im Folgenden gezeigt, wie die selbstorganisierende Anwendungsplatzierungsheuristik Kosten zu Kräften und letztendlich zu Platzierungsentscheidungen mittels der Ausführung der Basisoperationen verarbeitet, um letztlich eine effiziente Anwendungsausführung zu ermöglichen.

Ablauf der Anwendungsplatzierung und Umsetzung der Basisoperationen

Den Ausgangspunkt bildet eine bereits bestehende (initiale) Platzierung, also die Zuordnung einer Anwendung zu einem sie ausführenden Gerät innerhalb des Geräteensembles. Aus dieser bestehenden Anwendungsausführung heraus werden mittels Monitoring die Kosten der aktuellen Anwendungsausführung ermittelt. Dazu gehören grundsätzlich die Weiterleitungskosten, Kommunikationskosten und Speicherkosten. Aus dieser Kostensituation werden die daraus resultierenden Kräfte ermittelt, wobei für die letztgenannten Kosten die Nachbarbroker mit einbezogen werden müssen. Im Modell wird zunächst davon ausgegangen, dass ausreichend Kapazitäten zur Ausführung von Anwendungen und Anwendungskomponenten zur Verfügung stehen. Daher ist für die Ermittlung der auf die Anwendung wirkenden Zugkraft die Anzahl der aus- und eingehenden Nachrichten entscheidend. Darüber hinaus können Ressourcenbeschränkungen auch auf andere Weise in die Optimierung einbezogen werden.

Die Optimierung erfolgt durch die Ausführung einer Sequenz von Basisoperationen (Migration, Replikation, Dekomposition und Rekombination) auf den Anwendungen. Die Anpassung der Anwendungsplatzierung durch die Ausführung von Basisoperationen ist mit Aufwand verbunden, welcher durch die Widerstandskraft ausgedrückt wird und von der resultierenden Kraft überwunden

werden muss. Wird diese Kraft überwunden, gilt die Platzierung als vorteilhaft und wird durchgeführt. Es wird davon ausgegangen, dass die durch die Widerstandskraft widergespiegelten Kosten sich im Verlauf der Anwendungsausführung amortisieren. Für jede Anwendung innerhalb ihrer Ausführungsumgebung wird ein Kräftemodell erstellt, die resultierende Kraft ermittelt und entschieden, welche Sequenz von Basisoperationen ausgeführt wird.

Die Ermittlung einer oder mehrerer resultierender Kräfte wurde bereits im Abschnitt 3.5.4 erläutert. Grundsätzlich können zwei unterschiedliche Kräfte auf eine Anwendung einwirken. Zum einen die Zugwirkung nach außen und damit weg vom bisherigen Ausführungsort zu einem benachbarten Broker, zum anderen die Zugwirkung nach innen, auch zu einem benachbarten Broker. Die Betrachtung bzw. Unterscheidung „nach innen“ bzw. „nach außen“ richtet sich nach dem Blickwinkel der Anwendung, d.h. nach der Eigenschaft einer Anwendung, Teil eines ehemals zusammenhängenden Anwendungskonglomerats zu sein. Tatsächlich werden die Anwendungen auch bei dieser Kraft von ihrem derzeitigen Ausführungsort weggezogen.

Sind die Beträge beider Kräfte bekannt, erfolgt die Ermittlung der dominierenden Kraft, wobei je benachhbartem Broker die jeweils resultierenden Kräfte P_{exe-fw} und P_{st} gegenübergestellt werden. Dies erfolgt je Richtung B_i mit $i = 1 \ldots m$, wobei m die Anzahl der benachbarten Broker darstellt. Die Kraft $P_{exe-fw-st}$ je Richtung B_i ergibt sich als $P_{exe-fw-stB_i} = P_{exe-fw} + P_{stB_i}$. Analog zur Berechnung der resultierenden Kraft in Bezug auf die Zugwirkung nach außen, erfolgt auch hier die Berechnung der dominanten Kraft mit $P_{exe-fw-stdomA_zB_i}$. Für jede mögliche Richtung B_i mit $b_1 \ldots a_n$ wird nun folgendes ermittelt: Wenn

$$|P_{exe-fw-stA_zB_j}| > (\sum_{i=1}^{n} |P_{exe-fw-stA_zB_i}|) - |P_{exe-fw-stA_zB_j}|,$$

dann ist

$$P_{exe-fw-stA_zB_j}$$

eine resultierende Kraft. Nun ist es möglich, dass eine Kraft zwar nicht größer als alle anderen, möglicherweise aber genauso groß ist wie die Kräfte der benachbarten Richtung. Es würde dann auch

$$|P_{exe-fw-stA_zB_j}| \geq (\sum_{i=1}^{n} |P_{exe-fw-stA_zB_i}|) - |P_{exe-fw-stA_zB_j}|$$

gelten. Dieser Vergleich findet im Pseudocode innerhalb der Evaluationsmethode statt, wobei zunächst die Kräfte gegenübergestellt werden und anschließend die dominierende Kraft ermittelt wird.

Als nächstes ist entscheidend, wie viele Kräfte auf die Anwendung einwirken. Wenn nur eine Kraft auf die Anwendung einwirkt, ist sie ein Kandidat für eine Migration zu einem benachbarten Broker. Die Migration wird aber nur dann

ausgeführt, wenn sie in der Zukunft zu geringeren Kosten im Gesamtnetzwerk führt. Dazu muss der Betrag der Kraft $P_{exe-fw-st}$, die sich als dominierende Kraft herausgestellt hat, größer als die Widerstandskraft P_{res} sein, die nun ermittelt wird. Liegt $P_{exe-fw-st}$ darunter oder ist gleich groß, wird keine Aktion ausgeführt. Sobald allerdings zwei oder mehr Kräfte aus verschiedenen Richtungen gleich groß sind wie die Summe der Beträge der anderen Kräfte, also keine Kraft die anderen dominiert, sind mehrere Szenarien möglich. Hier ist es wichtig zu wissen, welche Nachrichtenströme die Anwendung aus welcher Richtung erreichen oder verlassen und ob alle eingehenden Nachrichtenströme die Anwendung aus jeder Richtung erreichen, Ausgangsströme die Anwendung in mehrere Richtungen verlassen, ob eine Anwendung in weitere Teilkomponenten zerlegbar oder duplizierbar ist. Ist die Anwendung zerlegbar, wird sie zunächst in ihre Teilkomponenten zerlegt, also dekomponiert. Anschließend werden die nun eigenständigen Komponenten bezüglich der auf sie einwirkenden Kräfte bewertet. Dabei ist es möglich, dass die Komponenten bestimmte Informationen gemeinsam nutzen. Eine eigenständige Platzierung würde die bisher gemeinsam genutzten Informationen an mehreren Stellen benötigen. Zwischen den bisher, auf einem Gerät, gemeinsam ausgeführten Anwendungen sind nach einer Verschiebung einer einzelnen Komponente zusätzliche Kommunikationsströme zu erwarten. Wurden diese bisher nicht durch Monitoring erfasst und im Kräftemodell aus den Weiterleitungskosten entsprechend berücksichtigt, müssen sie als Aufwand bei den Speicherkosten einbezogen werden. Eine Migration ist in diesem Fall nur dann sinnvoll, wenn die Kraft P_{exe-fw} in einer Richtung B_x sowohl die Kraft P_{st} als auch P_{res} überwiegt, es muss daher

$$|P_{exe-fw}| > |P_{st}| + |P_{res}|$$

gelten, was durch die Berechnung der resultierenden Kräfte auch gewährleistet ist, da in diesem Fall die Vorzeichen der Kraft P_{exe-fw} und P_{st} gegensätzlich sein werden. Im Ergebnis werden Teilkomponenten abgespalten und diese einzeln migriert. Im Ergebnis ist die bisherige Anwendungsteilkomponente für ihren jeweiligen Teilereignisbereich zuständig. Dabei ist es nicht ausgeschlossen, dass die Anwendungskomponenten nach und nach in die selbe Richtung migrieren.

Ein anderer Fall ist die Replikation der gesamten bisherigen Anwendung. Die Replikation erfolgt, wenn die Komponente selbst durch den Entwickler als replizierbar gekennzeichnet und vorbereitet ist, sie also Methoden zur Aufspaltung und getrennten Weiterführung des bisherigen Zustands vorweist. Replizierte Anwendungen teilen den Ereignisraum, sind also jeweils für Teile des Gesamtnetzwerks zuständig. Erhält bzw. sendet die Anwendung Nachrichten aus bzw. an alle benachbarten Broker, so ist möglicherweise eine Duplizierung der Anwendung sinnvoll, je nach der Verstärkung der Produktionsraten in Richtung der Ereignisquellen bzw. -senken. Die Replikation geht, ebenso wie die Dekomposition, üblicherweise mit einer Migration einher. Durch die Aufteilung des Ereignisraums durch die replizierten Anwendungen ist nicht von zusätzlichen Speicherkosten und einer Kraft P_{st} auszugehen. Nach erfolgter Aufteilung des

Ereignisraums durch die Replikation erfolgt daher wiederum die Auswertung der auf die Anwendung einwirkenden Kräfte. Für diese Betrachtung wäre eine weitere Ausführung am bisherigen Ausführungsort jedoch Voraussetzung. Kann aber bei einer Aufteilung des Ereignisraums auch davon ausgegangen werden, dass keine bzw. wenige Zustandsinformationen doppelt genutzt werden, können Speicherkosten auch hier vernachlässigt werden.

Die Replikation und Migration werden vom Koordinator veranlasst, dies zeigt auch Abbildung 3.33. Werden durch die unterschiedlichen Zugwirkungen, hervorgerufen vor allem durch P_{st}, aber auch durch P_{exe-fw}, ehemals miteinander verbundene Anwendungen wieder auf einem Gerät zusammen ausgeführt, lassen sie sich rekombinieren, und somit wieder zusammenfügen. Hier existieren grundsätzlich zwei Optionen, nämlich die „echte“ und die „unechte“ Rekombination. Bei der echten Rekombination werden die Zustände, Verarbeitungsschritte sowie konsumierte und erzeugte Ereignisströme zu einer einzigen Anwendung zusammengefasst. Bei der unechten Rekombination werden die Anwendungen nur gemeinsam auf einem Gerät ausgeführt, ohne die Zusammenfassung von Nachrichtenströmen und Zuständen. Die Kommunikation verläuft nicht über gemeinsam genutzte Speicherbereiche, sondern über den Austausch von Ereignissen. Letzteres ermöglicht die Anwendung des vorgestellten Kosten- und Kräftemodells, da Ereignisströme objektiv messbar sind, während die gemeinsame Nutzung von Speicher durch den Anwendungsentwickler explizit beschrieben sein und vor einer weiteren Optimierung möglicherweise eine aufwendigere Dekomposition durchgeführt werden muss. Die Entscheidung darüber, ob eine Replikation oder Dekomposition ausgeführt wird, wird auch am die Anwendung ausführenden Broker entschieden. Dazu überprüft der Koordinator laufend die von ihm ausgeführten Anwendungen. Sind davon einige zusammenzuführen, wird die Rekombination angestoßen. Sollen die Anwendungen, gesteuert von Vorgaben des Anwendungsentwicklers, kleinteilig verteilt werden, findet eine Dekomposition statt. In Pseudocode dargestellt bedeutet dies eine regelmäßige Überprüfung der ausgeführten Anwendungen auf eine mögliche Rekombination (siehe Abbildung 3.34) und Dekomposition (siehe Abbildung 3.35).

Durch die Ausführung der nur vier grundlegenden Operationen Dekomposition, Migration, Replikation und Rekombination lassen sich allein durch Permutation der Operationen viele unterschiedliche globale Platzierungszustände erreichen. Die Entscheidung, welche Basisoperation ausgeführt wird, obliegt einer Entscheidungsinstanz (Koordinator) für die jeweilige Anwendung auf dem ausführenden Gerät. Dabei ist neben dem Wissen über den Ressourcenverbrauch der jeweiligen Anwendung und die spezifischen Kommunikationsströme auf dem ausführenden Gerät auch der Informationsaustausch mit den Koordinatoren der Nachbargeräte erforderlich, um Kosten für hypothetischen Speicher- und den Ressourcenverbrauch der Anwendungsausführung zu berechnen. Dies erfolgt, analog dem Informationsaustausch während der initialen Platzierung, über Ereignisse mit entsprechenden Name/Wert-Paaren. Am jeweiligen Ausführungsort ist dazu im Vorfeld ein Monitoring über die verbrauchten Ressourcen durchzuführen. Zu dem Monitoring gehört auch die Ermittlung der exklusiv genutzten Ereignisströme

```
130 // Evaluates the cost situation, determines resulting forces
131 public void evaluateCosts(Application a_i, Value cost_st,
132                                              Value cost_fw, Value cost_exe) {
133
134  List<Force> force_exe_fw = new List<Force>(this.getNumberNeighborBroker());
135  List<Force> force_place = new List<Value>(this.getNumberNeighborBroker());
136  List<Force> force_st = new List<Value>(this.getNumberNeighborBroker());
137
138  // list of the resulting force
139  List<Force> force_res = new List<Value>(this.getNumberNeighborBroker());
140
141  // List of the dominating force
142  List<Force> force_dom = new List<Value>();
143
144  ...
145
146  // Combine force_exe_fw and force st
147  while (this.getNeigborList.hasNext()){
148    force_res.add(new Force(force_exe_fw.getNext().getValue() +
149      force_st.getNext().getValue(), a_i, this, this.getNextNeighbor());
150  }
151
152  // Determines the dominating force
153  force_dom = this.determineDominatingForce(force_res);
154
155  // test, if only one dominating force, this is a candidate for migration
156  if (force_dom.size() == 1) {
157    // test, if dominating force is at least as great as corresponding
158    // placement cost
159    // here, an additional internal method is used that returns the
160    // position of a broker in the neighboring list
161    Force force = force.getNext();
162    if (force.getValue() > force_place(this.getNeighborPosition()))
163      do migrate(this, force.getNeighbor(), a_i);
164  }
165
166  // test, if only more tha one dominating force, these are
167  // candidates for replication
168  if (force_dom.size() > 1) {
169    // test, if dominating forces are at least as great as
170    // corresponding placement cost
171    // here, an additional internal method is used that returns
172    // the position of a broker in the neighboring list
173    while (force_dom.hasNext()) {
174      Force force = force.getNext();
175      if (force.getValue() > force_place(this.getNeighborPosition()))
176        do replicate(this, force.getNeighbor(), a_i);
177    }
178  }
```

Abbildung 3.33: Auswertung der Kräfte und Entscheidung über Migration und Replikation

```
200 // Method at broker that checks and starts recombination
201 // of 2 related applications
202 public void checkForRecombination{
203
204   List<Application> appList;
205
206   // get list of all currently executed applications
207   appList = this.getApplicationList();
208
209   for each appList.hasNext(){
210     Application app = appList.getNext();
211
212     // searches for related apps at currently executed applications
213     if (appList.contains(app.getRelatedApplications())) {
214
215       // List for applications to be re-united with app
216       List<Application> recombList;
217       // calls internal method to get list of related applications
218       recombList = this.getAllRelatedApplications(app);
219
220         while (recombList.hasNext()){
221           app.recombine(recombList.getNext());
222         }
223     }
224   }
225 }
```

Abbildung 3.34: Durchlauf über alle ausgeführten Anwendungen; Suche nach Zusammenhängen: Suche nach Anwendungen, die zusammengefasst werden sollen und falls solche gefunden werden, Auslösen der Rekombination

```
220 // Method at broker that checks and starts decomposition of an application
221 public void checkForDecomposition{
222
223   List<Application> appList;
224
225   // get list of all currently executed applications
226   appList = this.getApplicationList();
227
228   for each appList.hasNext(){
229     Application app = appList.getNext();
230
231     if ((app is decomposable) && (globalPolicy == Decompose)) {
232
233       app.decompose();
234
235      }
236   }
237 }
```

Abbildung 3.35: Durchlauf über alle ausgeführten Anwendungen; Suche nach Anwendungen, die dekomponiert werden sollen und falls gefunden, Auslösen der Dekomposition

je Anwendung. Die Monitoringkomponente benötigt zur Bewältigung ihrer Aufgaben Wissen über die ausgeführten Anwendungen, ihren Ressourcenverbrauch und Kommunikationsströme. Eine Möglichkeit dies umzusetzen ist die Implementierung eines Anwendungscontainers, in dem die jeweilige Anwendung ausgeführt wird und Informationen über benötigte Ressourcen bereitgestellt werden.

Die Anpassung der Anwendungsplatzierung erfolgt nach laufender Beobachtung immer dann, wenn die beobachteten Kosten eine Kraft initiieren, die für eine Platzierungsänderung groß genug ist. Damit umfasst das Monitoring sowohl die Beobachtung der lokalen Kostensituation, als auch den Austausch über die möglichen Kosten bei der Platzierung von Anwendungen bei benachbarten Brokern. Dies erfordert zweifelsohne zusätzlichen Overhead. Zur Verringerung des Overheads ist es angebracht, Nachrichten über Kostensituationen nicht kontinuierlich, sondern anlassbezogen auszutauschen. Hierzu bietet sich die Definition von zeitlichen Intervallen oder Schwellwerten bzw. eine Kombination beider Methoden an. Ein zeitliches Intervall sichert die Aktualität der Platzierungsinformationen und gleicht mögliche ältere, fehlerhafte Kommunikation aus. Ähnliche Mechanismen sind Heartbeat-Events, mit denen Geräte des Netzwerks ihren aktuellen Zustand publizieren und die Validität von Kommunikationswegen sicherstellen. Eine Umsetzung mittels Schwellwerten sichert die gewünschte Adaptivität des Gesamtsystems, ähnlich bietet dies auch die Gegenkraft P_{res}. Die Kombination beider Ansätze, so z.B. durch die Initiierung einer Platzierungsänderung bei Erreichen eines Schwellwertes und bei dessen Ausbleiben die Aufnahme des regelmäßigen Austauschs (auch) im Zusammenhang der regulären Management-Nachrichten, kann die Vorteile beider Ansätze durch wenig zusätzlichen Overhead zusammenführen.

Wurde eine Entscheidung getrofffen und eine Basisoperation wurde bestimmt, steuert der Koordinator deren Ausführung. Ähnlich dem Ablauf bei der initialen Platzierung initiiert der Koordinator die Basisoperation. Der Koordinator entscheidet jedoch aus der Sicht des bisherigen Ausführungsgeräts, die Platzierungsänderung betrifft dabei allerdings auch einen bzw. mehrere benachbarte Broker. Jede Platzierungsänderung ist ein atomarer Prozess, welcher die Zusicherung und Belegung von Ressourcen, ggf. aber auch den Widerruf der angeforderten Ressourcen, beinhaltet.

Der Ablauf der Ausführung der Basisoperation ist ähnlich dem der initialen Platzierung, allerdings bezieht sich die alternative Platzierung auf das Nachbarumfeld des Brokers. Im Gegensatz zur initialen Platzierung mit ihrer Suche nach einer gültigen Platzierung geht es bei der Platzierungsadaption um eine punktuelle Verbesserung der Gesamtkostensituation. Zudem ist das Ziel einer ressourcensparenden und damit nachrichtensparenden Kommunikation nicht durch zusätzliche, global adressierte Nachrichten zu konterkarieren. Darüber hinaus gilt es zu beachten, dass eine Verschiebung einer Anwendung weiter als bis zum direkten Nachbarn weitreichende Konsequenzen bezüglich der Nachrichtenströme nach sich zieht, welche aus lokaler Sicht mit dem eingeschränkten Wissen nicht überschaubar sind.

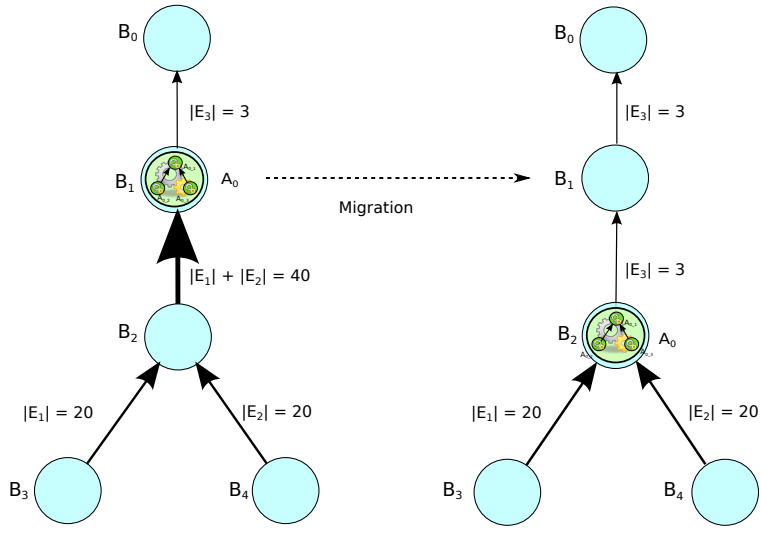


Abbildung 3.36: Einsparung von Kosten durch Migration einer Anwendung

Abbildung 3.36 zeigt den Zustand eines Brokernetzwerks mit fünf Brokern (B_0 bis B_4) und einer zunächst auf Broker B_1 platzierten Anwendung A_0 vor und nach der Migration der Anwendung zu B_2. Die Anwendung A_0 besteht aus den Teilkomponenten A_{0_1}, A_{0_2} und A_{0_3}. Die Teilanwendungen A_{0_2} und A_{0_3} subskribieren sich jeweils für Ereignisse vom Typ E_1 (A_{0_2}) und E_2 (A_{0_3}). Sie filtern die Ereignisströme und leiten sie an die Teilanwendung A_{0_1} weiter, welche die beiden eingehenden Ströme verarbeitet und als Ereignis des Typs E_3 veröffentlicht. Die Ereignisse E_3 werden von B_1 in Richtung B_0 weitergeleitet, da für diesen Ereignistyp eine Subskription vorliegt. Die Ereignisströme E_1 und E_2 stammen exklusiv aus Richtung B_3 (E_1) bzw. B_4 (E_2). Die Ereignisströme E_1 und E_2 erreichen den Broker B_2 mit jeweils 20 Ereignissen je Zeiteinheit. Somit werden 40 Ereignisse an Broker B_1 und die Anwendung A_0 weitergeleitet. Intern findet zunächst eine Filterung von E_1 und E_2 durch A_{0_2} und A_{0_3} statt. Durch A_{0_1} erfolgt die Zusammenführung der gefilterten Ereignisse, sodass im Ergebnis 3 Ereignisse vom Typ E_3 je Zeiteinheit die Anwendung A_0 und damit B_1 in Richtung B_0 verlassen. Kostenseitig ergibt sich das Bild, dass sich die Ausführungskosten von A_0 auf B_1 und B_2 nicht unterscheiden, Speicherkosten fallen in diesem Anwendungsbeispiel nicht an. Was bleibt, sind die Platzierungskosten, welche pauschal mit 5 Kosteneinheiten in das Kräftemodell eingehen, sowie die Kommunikationskosten. Durch Verschiebung von A_0 von B_1 zu B_2 lassen sich 40 Nachrichten einsparen, nur 3 Nachrichten würden zusätzlich verschickt, es ergibt sich ein Einsparpotenzial von 37. Abgeleitet von den Kosten wirkt eine Kraft auf A_0 von B_1 in Richtung B_2 mit dem Betrag von 37. Diese ist

deutlich größer als die Widerstandskraft bzw. die Reibung in Höhe von 5, sodass Anwendung A_0 zu B_2 migriert und Kosten in Höhe von 32 eingespart werden.

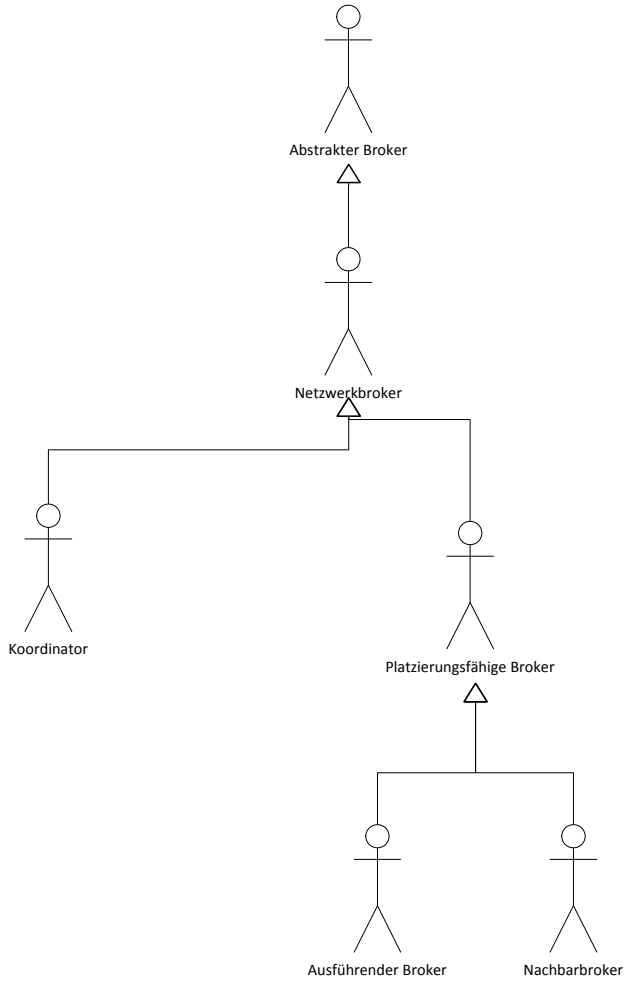


Abbildung 3.37: Akteure und Rollen der laufenden Platzierung

Der Koordinator, der sinnvollerweise auf einem Gerät läuft, auf dem auch die betrachtete Anwendung ausgeführt wird, kommuniziert nach der getroffenen Platzierungsentscheidung und einer erfolgreichen Dekomposition bzw. Replikation mit jedem betroffenen benachbarten Broker. Da diese Basisoperationen bereits ausgeführt wurden, wird im Folgenden die Migration einer Anwendungskomponente beschrieben. Da die benachbarten Broker direkt über Nachrichten miteinander kommunizieren, ist der Kommunikationsablauf einfacher als dies bei der initialen Platzierung der Fall ist. Im Vergleich zur initialen Platzierung wird an dieser Stelle nicht zwischen ausgewähltem und nicht ausgewähltem Broker

unterschieden, stattdessen ist der Nachbarbroker eine allgemeine Instanz des platzierungsfähigen Brokers, und der ausgewählte Broker entspricht in dieser Situation dem ausführenden Broker. Die Rolle des Koordinators bleibt dagegen unangetastet. Es ist jedoch davon auszugehen, dass ein Netzwerkbroker sowohl der Rolle des Koordinators, als auch der Rolle des ausführenden Brokers entspricht. Abbildung 3.37 verdeutlicht dies.

Der Ablauf der Migration wird als UML-Ablaufdiagramm (Abbildung 3.38) verdeutlicht und findet sich danach noch zusätzlich in gewohnter, tabellarischer Darstellung wieder.

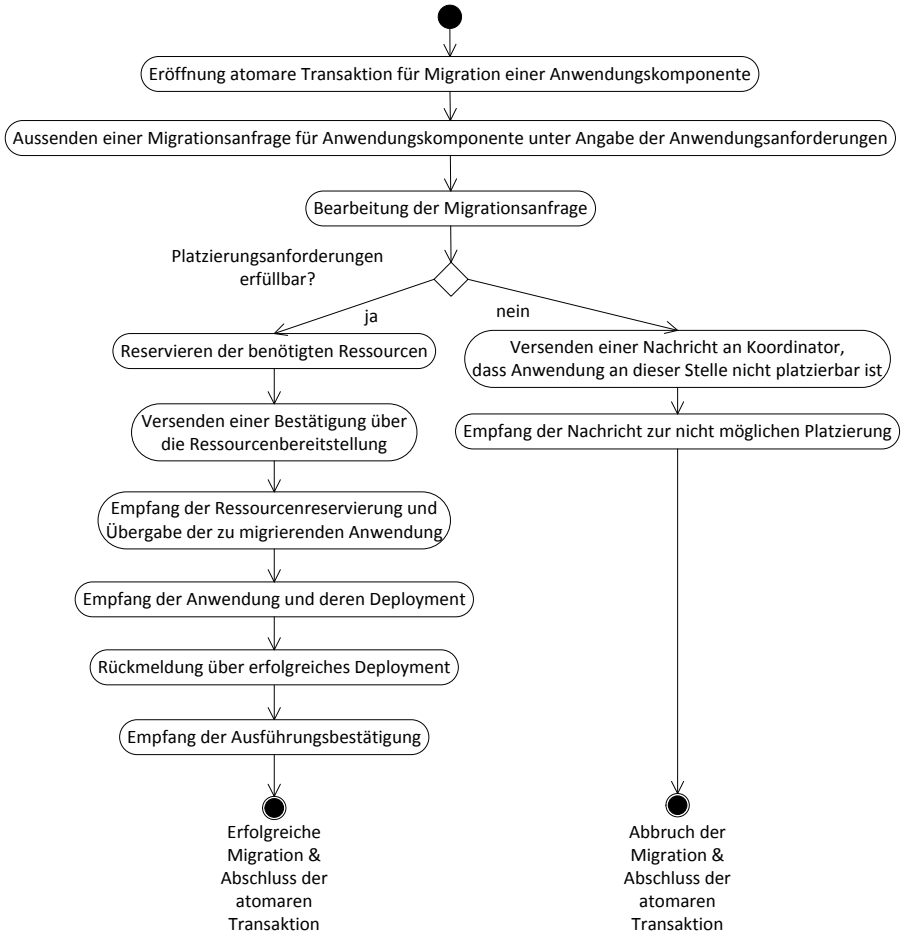


Abbildung 3.38: UML-Ablaufdiagramm zur Migration im Rahmen der laufenden Platzierungsanpassung

	Koordinator	**Nachbarbroker**
1	eröffnet atomare Einheit für Ausführung der Migration einer bestimmten Komponente	
2	Aussenden einer Migrationsanfrage für Anwendungskomponente unter Angabe der Anwendungsanforderungen	
3		Bearbeitung der Migrationsanfrage

Fallunterscheidung: Sind Anwendungsanforderungen erfüllbar?
Ja:

4a		Reservieren der benötigten Ressourcen
5a		Versenden einer Bestätigung über die Bereitstellung der benötigten Ressourcen
6a	Empfang von 5a und Übergabe der zu platzierenden Anwendung	
7a		Empfang der Anwendung und Deployment
8a		Rückmeldung an den Koordinator über das erfolgreiche Deployment
9a	Empfang der Bestätigung und Abschluss der atomaren Einheit	

Nein:

4b		Nachricht an Koordinator, dass Anwendung nicht platzierbar ist
5b	Empfang von 4b, Abbruch und Abschluss der atomaren Aktion	

Zusätzlich ist auch der Abbruch der Platzierung nach 8a durch den Koordinator mögich, falls keine Rückmeldung des vorgesehenen neuerlich ausführenden Gerätes eintrifft (Sprung zu 5b, siehe auch Abbildung 3.39). In diesem Fall wird davon ausgegangen, dass im Publish/Subscribe-Kommunikationssystem kein Ereignis verlorengeht und damit die Instanziierung der Anwendung auf dem Nachbarbroker nicht erfolgreich verlief. Eine doppelte Anwendungsausführung erfolgt damit nicht. Kann eine doppelte Anwendungsausführung jedoch nicht vermieden werden, würden gleiche Ereignisströme zu den Anwendungen geleitet werden. In diesem Fall laufen die üblichen Optimierungsschritte ab und damit wäre eine Rekombination beider Anwendungen erreichbar.

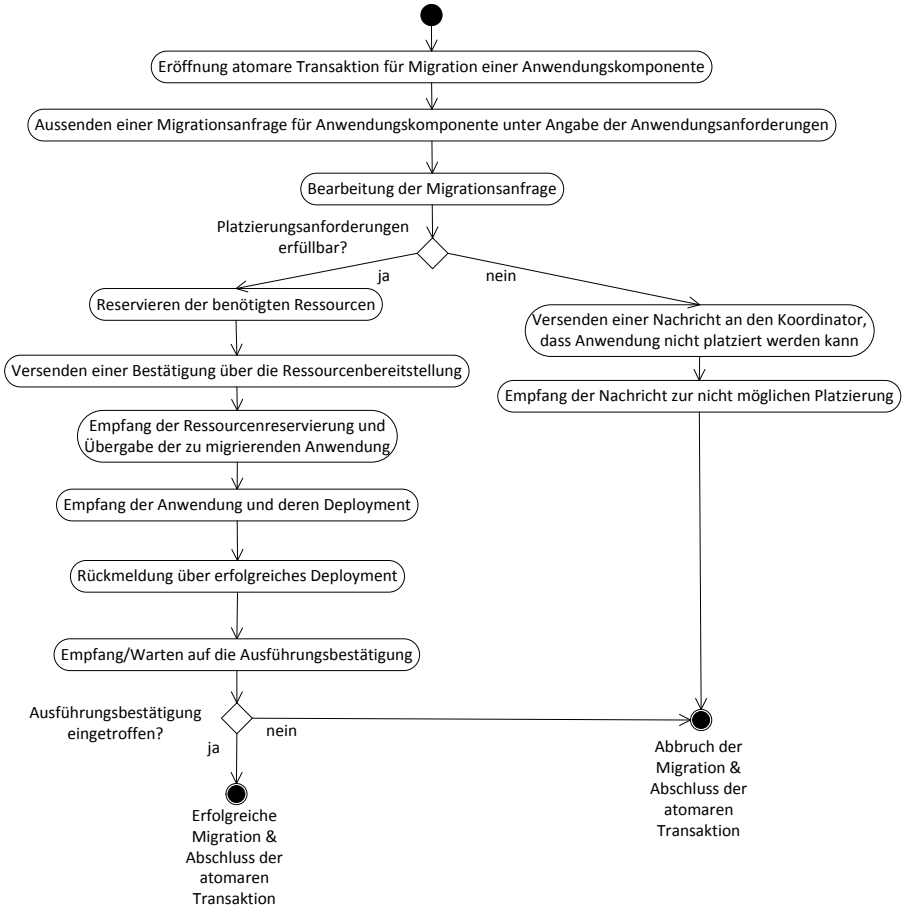


Abbildung 3.39: UML-Ablaufdiagramm zur Migration im Rahmen der laufenden Platzierungsanpassung mit Rücksprung

Grundsätzlich erfolgt die Migration unabhängig von vorherigen Platzierungsentscheidungen, sie kann jedoch auch im Verbund mit den anderen Basisoperationen agieren. So wird Migration nach den ebenfalls atomaren Aktionen Dekomposition und Replikation regelmäßig im Nachgang dieser Operationen ausgeführt. Dabei ist die Ausführung der Migration an den Erfolg der vorgelagerten atomaren Aktion gebunden und findet bei Nichterfolg von Dekomposition bzw. Replikation nicht statt. Genauso unabhängig voneinander ist eine der Migration nachgeschaltete Rekombination mit einer am Zielbroker bereits ausgeführten Anwendungskomponente. Ob eine Rekombination möglich ist, wird durch den Broker entschieden und ausgeführt.

Abbildung 3.40 führt das Beispiel aus Abbildung 3.36 weiter. Anwendung A_0 befindet sich nun auf Broker B_2 und erhält Ereignisströme aus Richtung von B_3 und B_4. Würde A_0 in Richtung B_3 oder B_4 verschoben, könnten vom Ereignisstrom E_1 bzw. E_2 jeweils 5 Kosteneinheiten eingespart werden, allerdings müssten der gegenüberliegende Strom und der Ausgangsstrom E_3 ebenfalls umgeleitet werden, was mit 20 sowie zusätzlichen 3 Einheiten insgesamt höhere Kosten nach sich ziehen würde, als die durch die Verschiebung realisierten Einsparungen. Allerdings lässt sich die Anwendung A_0 in Teilanwendungen dekomponieren, wobei A_{0_1} exklusiv für E_1 und A_{0_2} exklusiv für E_2 zuständig ist. Bei der Betrachtung der Kostensituation für A_{0_1} und A_{0_2} stellt sich heraus, dass sich bei der Verschiebung von A_{0_2} von B_2 nach B_3 und von A_{0_3} von B_2 nach B_4 jeweils Weiterleitungskosten von 15 Kosteneinheiten ergeben (20 Nachrichten nach B_2 werden um 5 vorgefilterte Nachrichten verringert). Bleiben die Ausführungskosten von A_{0_2} und A_{0_3} konstant und die Platzierungskosten bleiben ebenfalls bei 5, ergibt sich je Richtung eine Einsparung von 15 Kosteneinheiten je Zeiteinheit, wobei sich bei allen folgenden Zeiteinheiten eine Einsparung von 20 Kosteneinheiten im Vergleich zum Ausgangszustand einstellt. Zeigt die linke Seite der Abbildung 3.40 den Zustand vor der Dekomposition und Migration, wird auf der rechten Seite der Endzustand dargestellt.

Die Abbildung 3.41 verdeutlicht die Arbeitsweise der Replikation in Zusammenarbeit mit der Migration, dort ist ein Brokernetz in zwei Zuständen, vor und nach der Replikation und anschließender Migration einer Anwendung, zu sehen. Das Brokernetz besteht aus sechs miteinander verbundenen Brokern (B_1 bis B_6). Zunächst befindet sich eine Anwendung A_0 auf dem Broker B_2, wie dieses auf der linken Seite der Abbildung zu erkennen ist. Die Anwendung benötigt Ereignisströme vom Typ E_1 und erhält diese aus Richtung der Broker B_3 und B_4. Sie produziert selbst Ereignisse vom Typ E_2, wobei diese in Richtung des Brokers B_1 weitergeleitet werden, da eine Subskription für E_2 aus dieser Richtung vorliegt. Je Zeiteinheit erzeugt die Anwendung zehn Ereignisse E_2 und konsumiert jeweils die von B_3 und B_4 weitergeleiteten 20 Ereignisse E_1. Dabei wird durchschnittlich nur jedes vierte Ereignis je Eingangsstrom weiterverarbeitet. Die Ereignisströme von E_1 aus Richtung B_3 und B_4 sind unabhängig voneinander und werden durch A_0 entsprechend behandelt. Bei gleichen Ausführungskosten von A_0 an B_2, B_3 und B_4 und ohne Aufwand für Speicherkosten fällt bei der Beurteilung der einwirkenden Kräfte der Blick auf die Weiterleitungskosten. Da beide

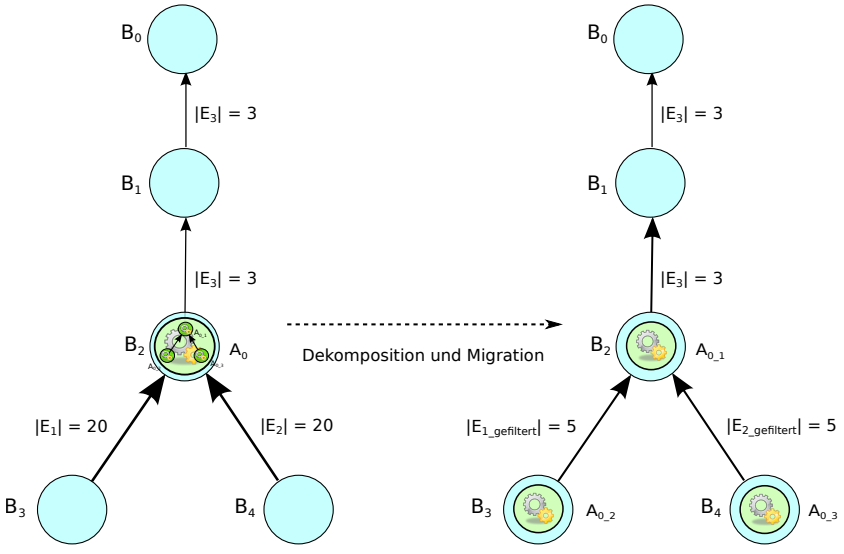


Abbildung 3.40: Einsparung von Kosten durch Dekomposition und Migration von Anwendungskomponenten

Eingangsereignisströme unabhängig voneinander ausgewertet werden, ließe sich durch die Replikation von A_0 und die Migration der replizierten Anwendung Weiterleitungsaufwand in Höhe der nicht weitergeleiteten Nachrichten einsparen. In diesem Fall filtert Anwendung A_0 ein Viertel der eingehenden Nachrichten, je Richtung würden damit, bei 20 eingehenden und 5 ausgehenden Ereignissen je Zeiteinheit, Kosten im Wert von 15 Einheiten eingespart. Da die Migration selbst nur mit je 5 Kosteneinheiten (Platzierungskosten) als Gegenkraft wirkt, werden pro Richtung im ersten Zeitabschnitt noch immer je Richtung 10 Kosteneinheiten eingespart. Die von A_0 replizierten Anwendungen A_{0-1} und A_{0-2} werden damit in Richtung B_3 bzw. B_4 gezogen.

Wie aus dem vorgestellten Migrationsablauf ersichtlich ist, werden Anwendungen nur dann verschoben, wenn auf dem ausgewählten Broker die Anwendungsanforderungen erfüllt werden können und ausreichend Kapazitäten bereitstehen. Beides wird im Rahmen der Migrationsanfrage in Form von Name/Wert-Paaren übermittelt. Es kann vorkommen, dass eine Migrationsanfrage durch ein angefragtes Gerät abgelehnt wird, weil nicht ausreichend Kapazitäten zur Verfügung stehen. Auf diesem offensichtlich überlasteten Gerät sind keine weiteren Ressourcen frei, auch wenn durch eine Verschiebung einer einzelnen Komponente Verbesserungen erreicht werden könnten. Eine Weiterverschiebung dieser Anwendung vom überlasteten Gerät weg ist nicht vorgesehen, da (negative) Seiteneffekte die Vorteilhaftigkeit der Platzierungsänderung zunichtemachen können, aber gleichzeitig mit lokalem Wissen nicht abschätzbar sind. Besonders nachteilig ist der Fall, bei dem zwei gegenüberliegende Geräte Anwendungen auf das jeweils andere Gerät verschieben möchten. Sind beide Geräte ressourcenmäßig ausgeschöpft,

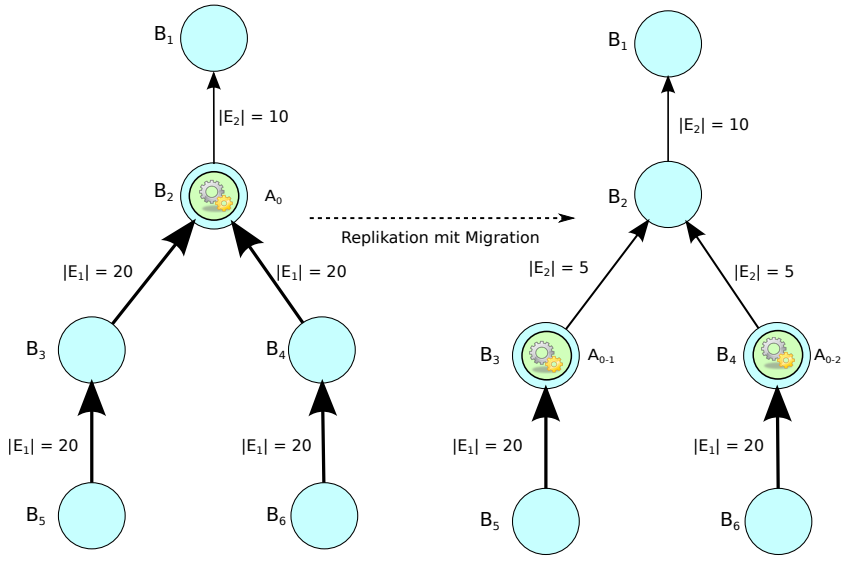


Abbildung 3.41: Einsparung von Kosten durch Replikation und Migration von Anwendungskomponenten

befinden wir uns offensichtlich in einem Deadlock, es geht für die Anwendungen weder in die eine, noch in die andere Richtung. Um diesen Stau aufzulösen, existiert eine Erweiterung des Migrationsprotokolls, nämlich ein gegenseitiger Austausch von Anwendungskomponenten. Da die jeweiligen Koordinatoren ihre Änderungsanfragen zwar losgelöst von denen der Nachbarkoordinatoren stellen, diese allerdings nach Eingang der Migrationsanfragen und damit sequentiell verarbeiten, kann ein Tausch von Komponenten einen Deadlock auflösen. Der Austausch muss jedoch ins Protokoll integriert werden. Dafür ist der Einschub bzw. die Ergänzung 4bn zwischen den Schritten 4b und 5b notwendig.

4b		Nachricht an Koordinator, dass Anwendung nicht platzierbar ist
4bn		Überprüfung, ob Anwendung zum Tausch angeboten werden kann
5b	Empfang von (4b), Abbruch und Abschluss der atomaren Aktion	

Kann keine Anwendung zum Tausch angeboten werden, erfolgt der Sprung auf 4b und den bisherigen Ablauf. Wenn nicht, wird ausgehandelt, ob eine Anwendung getauscht werden kann.

5bn		Nachricht an Koordinator, dass Anwendung nicht platzierbar ist, darin enthalten eine Anwendung, die selbst in Richtung des Initiators/Koordinators geschickt werden soll
6bn	Empfang von (5bn); Test ob Platzierung der in (5bn) vorgeschlagenen Anwendung nach Migration der Ausgangsanwendung möglich ist	

Wenn ein Tausch möglich ist:

7bn	wenn Tausch möglich: atomare Teilaktion als Migration der in (5bn) vorgeschlagenen Komponente zum Koordinator	
8bn	Migration der ursprünglich zu migrierenden Anwendung	
9bn	Abschluss der atomaren Teiltransaktion	

Ist kein Tausch möglich, werden die Schritte 7bn bis 8bn übersprungen und die Transaktion mit dem Schritt 5b abgebrochen. Diesen Ablauf verdeutlicht Abbildung 3.42.

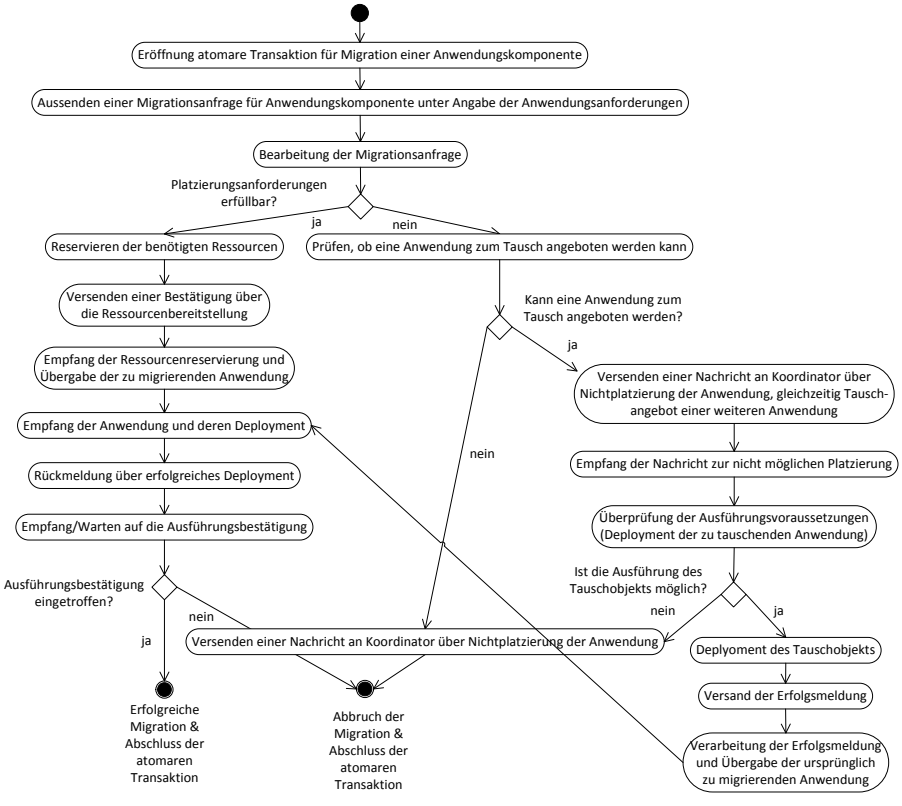


Abbildung 3.42: UML-Ablaufdiagramm zur Migration im Rahmen der laufenden Platzierungsanpassung mit Rücksprung und Tausch von Anwendungen

Einflussfaktoren auf das Kräftemodell

Wie in den vorigen Abschnitten beschrieben, wird regelmäßig die aktuelle Kostensituation ausgelesen und anschließend werden daraus die einwirkenden Kräfte ermittelt. Auf der Grundlage der ermittelten Kräfte wird entschieden, welche der vier Basisoperationen anschließend ausgeführt wird. Bisher wird die Kostensituation zu bestimmten Zeitpunkten analysiert, wobei die Nachrichtenströme ungewichtet aufsummiert werden. Da bei diesem Verfahren alle Nachrichten eine gleich hohe Gewichtung haben, werden aktuelle Entwicklungen daher als genauso wichtig gesehen wie vergangene Nachrichtenströme. Um den Grad der Adaptivität des Gesamtsystems noch feingranularer zu gestalten, bietet sich die Stellschraube der unterschiedlichen Gewichtung der Nachrichtenströme an.

Grundsätzlich lässt sich durch zwei Stellschrauben die Gewichtung der Nachrichtenströme anpassen. Zum einen die Gewichtung neuer und älterer Ereignisströme und zum anderen die Zeitpunkte, zu denen die Kostenevaluation stattfindet. Zweiteres ist jedoch nur in Zusammenarbeit mit der unterschiedlichen Gewichtung von Nachrichtenströmen sinnvoll, verstärkt allerdings den dort gewünschten Effekt. Daher wird sich im Folgenden auf den Gewichtungseffekt konzentriert. Die Gewichtung wird mit einem Glättungsfaktor realisiert, welcher auch als α bezeichnet wird. Die aktuellen Nachrichtenströme, also die Anzahl der Nachrichten im aktuellen Zeitfenster, werden mit α multipliziert und mit den historischen Nachrichtenströmen, zuvor mit dem Faktor $1-\alpha$ multipliziert, aufaddiert. Damit ergibt sich die Gewichtung der Nachrichtenströme je Zeiteinheit, wobei eine häufige Durchführung dieser Operation die gewünschte Wirkung verstärkt. Ist $\alpha > 0,5$, so werden die aktuellen Nachrichtenströme stärker gewichtet, ist $\alpha < 0,5$ werden die aktuellen Nachrichtenströme den vergangenen untergeordnet, kurzfristige Effekte spielen dann eine nachgeordnete Rolle. Bei dieser Art von Berechnung ist es jedoch notwendig, die historischen Nachrichtenströme aufzubewahren. Dazu wird eine Historie von allen Kosten vergangener Nachrichten angelegt (C^*). Diese wird im Folgenden mit C_{t-1} bezeichnet. Die Berechnung der aktuellen (generischen) Kostensituation C_t^* erfolgt damit zum Zeitpunkt t als $C_t^* = \alpha \times C_t + (1-\alpha) \times = C_{t-1}^*$. Nach jedem Berechnungsdurchlauf wird nur noch der Kostenwert für die Gesamtkosten C^* zum Zeitpunkt t, also $C*_t$ benötigt, welcher im kommenden Durchlauf als C_{t-1}^* Verwendung findet. Durch dieses Vorgehen verlieren historische Werte bei jedem Durchlauf zunehmend an Bedeutung, in diesem Zusammenhang wird auch von exponentiellem Vergessen bzw. exponentiellem Glätten gesprochen. Grundsätzlich ist diese Art von Glättung zwar bei allen besprochenen Kostenarten anwendbar, beim Vergleich von aktuellen Kostensituationen und den daraus ableitbaren Kosten ist dies jedoch nur bedingt sinnvoll. Dies betrifft Platzierungs-, Ausführungs- und Speicherkosten, es bleiben als relevantes Anwendungsfeld lediglich die Weiterleitungskosten.

Zur Nutzung der vorherigen Weiterleitungskosten führt der Broker diese Kosten und stellt sie entsprechend zur Verfügung. Eine geeignete Erweiterung des Brokers ist nötig und wird ebenso in Abbildung 3.43 angedeutet, wie der Zugriff auf den Glättungsfaktor α sowie dessen Änderung.

```
10 public class Broker implements... {
11
12   ...
13
14   // List of historical List of forwarding costs
15   private List<Value> cost_fw_hist = new List<Value>();
16
17   ...
18
19   // Smoothing factor for exponential smoothing of forwarding costs
20   private double alphaCostFw;
21
22     ...
23
24
25   /**
26   * Returns historic forwarding costs
27   **/
28   public List<Value> getCost_fw_hist() {
29     return this.cost_fw_hist
30   }
31
32   /**
33   * Sets historic forwarding costs
34   **/
35   public void List<Value> setCost_fw_hist(List<Value> cost_fw) {
36     this.cost_fw_hist.removeAll();
37     this.cost_fw_hist.addAll(cost_fw);
38   }
39
40   ...
41
42
43   /**
44   * Returns the smoothing factor for forwarding costs
45   **/
46   public double getAlphaCostFw(){
47     return this.alphaCostFw;
48   }
49
50   /**
51   * (Re-) Sets the smoothing factor for forwarding costs
52   *
53   **/
54   public void setAlphaCostFw(double newAlpha){
55     this.alphaCostFw = newAlpha;
56   }
57 }
```

Abbildung 3.43: Erweiterungen des Brokers um Funktionalitäten des exponentiellen Glättens

```
70 // Determines costs resulted by forwarding events
71 public List<Value> determineForwardCosts(Broker broker, Application a_i);
72   List<Value> cost_fw = new List<Value>(broker.getNumberNeighborBroker());
73   List<Value> cost_fw_old = broker.getCost_fw_hist();
74   Value cost_fw_input, costfw_output;
75
76   for all broker.getNextNeighbor() {
77   // for each neighboring broker
78     if ( broker.getNextNeighbor().hasNext()) {
79
80       Broker n_broker = broker.getNextNeighbor();
81
82       // Determines balance between in- and outgoing event streams
83       // per direction (neighboring broker)
84       ...
85       cost_fw_input =
86            add(broker.determineAmountEventStreams(
87                 broker.getExclusiveEventStreams(a_i.getEventStreamsIn());
88
89       // Determines amount of outcoming event streams per neighbor broker
90
91       ...
92
93       cost_fw_output = broker.determineAmountEventStreams(
94          broker.getExclusiveEventStreams(a_i.getEventStreamsOut());
95
96           // Determine balance of in- and output events
97           Value cost = cost_fw_input - cost_fw_output;
98
99           // Determine weighted historical cost
100          Value cost_hist = cost_fw_old.get(n_broker).
101                            multiplyWSmoothFac(1-(broker.getAlphaCostFw()));
102
103          // Determine overall cost with hist. cost and smoothing factor
104          cost =
105            cost.multiplyWSmoothFac(broker.getAlphaCostFw()) + cost_hist;
106
107          // write resulting cost into list cost_fw
108          cost_fw.add(cost);
109
110          // write resulting cost into historical data list
111      cost_fw_old.set(n_broker, )
112    }
113  }
114 }
```

Abbildung 3.44: Berechnung der tatsächlichen Weiterleitungskosten von Ereignissen bezüglich einer Anwendung a_i auf einem Broker unter Berücksichtigung vergangener Kostenentwicklungen

Die Veränderungen müssen sich diesbezüglich auch in der Berechnung der Weiterleitungskosten widerspiegeln, welches sich im Codebeispiel in Abbildung 3.44 zeigt. Zu dieser Berechnung wird auf Seiten der Klasse $Value$ zusätzlich eine Methode $public\ multiplyWSmoothFac(double\ smoothingFactor)$ benötigt, welche den jeweiligen Kostenwert mit dem Glättungsfaktor multipliziert.

3.6 Diskussion

Das manuelle Deployment von Anwendungen in Verantwortung des Nutzers führt in intelligenten, dynamischen Umgebungen mit unterschiedlich leistungsfähigen Geräten und Kommunikationsverbindungen zu devergierenden Auslastungen der einzelnen Komponenten. Die daraus resultierende Bildung von Engstellen fördert die unnötige Verknappung von Ressourcen, sodass sowohl die Servicequalität als auch die Kostensituation bezogen auf das gesamte Geräte- und Anwendungsensemble leiden. Das automatische Anwendungsdeployment entlastet einerseits den Nutzer von ungeliebten Konfigurierungs- und Managementaufgaben, ermöglicht andererseits dem System die gleichmäßigere Auslastung von Komponenten, ohne dass Anforderungen der Anwendungen an das sie ausführende Gerät nicht erfüllt werden können. Die Einbettung dieses Dienstes in die Publish/Subscribe-Middleware ermöglicht zudem eine Anwendungsplatzierung ohne spezifische Anpassung an die Plattform des ausführenden Gerätes und die Nutzung bestehender Publish/Subscribe-Dienste, wie der Methoden der Selbststabilisierung [110, 111] und der Bewertung und Anpassung von Kommunikationsströmen [154, 191]. Eine einmal gefundene Platzierung ist in einer dynamischen Umgebung mit hinzukommenden und wegfallenden Geräten und Anwendungen nicht dauerhaft gültig und muss bei Änderungen im Dienste der Sicherung der Nutzer- und damit Anwendungsanforderungen angepasst werden können. Neben den Qualitätsanforderungen lässt sich die Anwendungsplatzierung so gestalten, dass die geforderten Anwendungen effizient ausgeführt werden, wobei Effizienz von der Sichtweise abhängt und bei dem hier vorgestellten Ansatz vor allem Kosteneffizienz im Sinne von Einsparungen von Kommunikationsaufwand bedeutet. Dies ist bei mobilen Geräten zumeist der bestimmende, begrenzende Faktor, da Kommunikationsaufwand die Hauptbelastung für die begrenzten Ressourcen des Akkus darstellt. Zur Verlängerung der Laufzeit der Anwendungen und Geräte ist daher neben einer gültigen, eine möglichst gute Platzierung unentbehrlich.

In der Literatur existieren verschiedene Vorgehensweise für eine (möglichst) optimale Platzierung von Komponenten. Dabei ist jedoch zu beachten, dass aufgrund der Komplexität des Optimierungsproblems und der Abwägung zwischen Ressourcenverbrauch für den Optimierungsvorgang und der Dauerhaftigkeit der gefundenen Lösung in einer dynamischen Umgebung für die laufende Optimierung eine Heuristik auf der Grundlage lokalen Wissens sinnvoll ist. Auch aus der Abwägung zwischen dem Aufwand zur Suche einer optimalen Lösung und

deren Dauerhaftigkeit verfolgt das in dieser Arbeit vorgestellte Platzierungskonzept zunächst die Idee, eine gültige Platzierung zu finden und diese schrittweise zu verbessern. Daher ist auch dieses Verfahren eine Mischung zwischen zentraler und dezentraler Platzierung, nämlich der zentralen initialen Platzierung und der dezentralen laufenden Anpassung.

Während der initialen Platzierung und der Suche nach einer gültigen Platzierung übernimmt ein zentraler Koordinator die Platzierung. Durch die Abbildung der jeweiligen Schritte auf Publish/Subscribe-Primitive wird in mehreren Schritten nach einer gültigen Platzierung gesucht und wenn nötig, die Anwendung bereits in Komponenten zerlegt. Die Zerlegung erfolgt jedoch nur, wenn dies durch den Anwendungsentwickler ermöglicht wurde, ein eigenständiges Aufteilen der Anwendung in Komponenten ist ein zukünftiges Forschungsfeld. Die Nutzung der Publish/Subscribe-Primitive erlaubt zudem eine ressourcenschonende Verteilung. Kann keine gültige Platzierung gefunden werden, wird die Platzierung abgebrochen und der Nutzer entsprechend informiert. Alternativ könnte, unter Inkaufnahme von Überlast, die Ausführung am Ort des Koordinators erfolgen. Im Vergleich zur manuellen Platzierung an diesem Ort wäre damit eine vergleichbare Situation geschaffen, vorausgesetzt der Nutzer würde die Anwendung auch an dieser Stelle ausführen wollen. Die initiale Platzierung ist verhältnismäßig schlank organisiert, Optimierungen werden an dieser Stelle nicht vorgenommen.

In der zweiten Phase erfolgt die laufende Anpassung der Anwendungsplatzierung. Die dabei genutzte Heuristik der sich entspannenden Kräfte ist durch die Nutzung in anderen Forschungsprojekten bereits erprobt [97]. Auch hier soll mit möglichst wenig Overhead ein gutes Ergebnis erreicht werden. Es wird daher darauf verzichtet, einen globalen Kostenraum zu definieren und Anwendungen in diesem zu platzieren, auch wenn dann keine weiteren globalen Optimierungsstrategien ausgeführt werden können. Allerdings ist es sinnvoll, je nach Gestaltung der Anwendungen, diese in Komponenten zu zerlegen und einzeln zu platzieren. Die auf die Platzierung einwirkenden Kräfte werden durch die Kosten bestimmt, die in der Anwendungsausführung durch eine geänderte Platzierung eingespart werden können. Durch das dynamische Anwendungsumfeld sind dies vor allem Weiterleitungskosten, während Ausführungs- und Speicherkosten eine untergeordnete Rolle spielen. Während bei den Weiterleitungskosten vor Ort abgeschätzt werden kann, welche Kosten eingespart werden können, wird bei den anderen Kostenarten die Differenz zwischen tatsächlichem und fiktivem Aufwand als Berechnungsgrundlage genutzt. Bei dem fiktiven Aufwand wird zudem mit den Grenzkosten gerechnet, womit dem Ziel, möglichst keine Lastspitzen zu erzeugen, ebenfalls Rechnung getragen wird.

Kapitel 4

Implementierung

Inhalt

4.1 Einleitung

Systeme, die die Interaktion von Anwendungen und Geräten als Ensemble in einer intelligenten Umgebung ermöglichen, müssen die dem System inhärente Dynamik ebenso unterstützen wie unterschiedliche Geräteplattformen. Zusätzlich müssen weitere Anforderungen, bspw. nach effizienten Routingverfahren von Daten oder nach der Fehlertoleranz, angemessen berücksichtigt werden. Publish/Subscribe als Kommunikationsparadigma ermöglicht die inhaltsbasierte Kommunikation und die lose Kopplung von Geräten und Anwendungen. Damit eignet sich Publish/Subscribe grundlegend als Kommunikationsparadigma in dynamischen, verteilten Systemen. Publish/Subscribe leitet eine Zustandsänderung an alle Geräte und Anwendungen weiter, die sich für diese Änderung interessieren, und zwar unabhängig von ihrer ID oder ihrem Standort im Netzwerk. Neu hinzukommende Geräte und Anwendungen geben bei Interesse an dem Ereignis eine entsprechende Subskription ab und erhalten bis zum Widerruf der Subskription die entsprechenden Ereignisse. Je mehr Geräte und Anwendungen jedoch auf eine Publish/Subscribe-basierte Kommunikationsinfrastruktur zurückgreifen, desto höher ist die Wahrscheinlichkeit, dass unterschiedliche Anforderungen zu erfüllen sind. So sind bspw. je nach Verhältnis von gleichen zu verschiedenen Notifikationen unterschiedliche Routingmechanismen für Notifikationen sinnvoll. Auch der Einsatz dieses oder jenes Filtermechanismus für Ankündigungen, Subskriptionen und Notifikationen ist von der jeweiligen Einsatzumgebung abhängig. Eine Middleware für alle Einsatzszenarien - derzeitige und zukünftige - zu entwerfen und zu implementieren ist jedoch ein mehr als schwieriges Unterfangen, können doch im Voraus nicht alle Anforderungen an die Middleware formuliert werden. Falls wider Erwarten doch alle denkbaren Anforderungen für alle möglichen Anwendungsfälle zusammengetragen werden könnten, führt die Umsetzung der Anforderungen zu einem sehr komplexen Systementwurf, wobei bei einem weitreichenden Anforderungskatalog einzelne Anforderungen sich nicht nur überlappen, sondern auch konträr gegenüber stehen können. Dies führt in der Folge ebenso regelmäßig zu Priorisierungen, Kompromissen und nicht beherrschbaren Seiteneffekten.

Rebeca ist eine Middleware, welche in der Forschung verwendet wird und auch die Grundlage der Implementierung des in dieser Arbeit vorgestellten Konzepts bildet. Aufgrund der unterschiedlichen Anforderungen an eine Middleware im Allgemeinen und vor allem in der Forschung [170] wird sie komponentenbasiert entwickelt, sodass keine allgemeingültige und umfassende Middleware konzipiert und implementiert wird. Vielmehr wird eine modulare, den jeweiligen Anforderungen anpassbare, Middleware geschaffen. Dies wird erreicht, indem die Middleware als erweiterbares Gerüst konzipiert wird. So existiert ein funktionaler Rumpf, dem je nach Einsatz- und Forschungszweck die gewünschten und benötigten Module hinzugefügt werden. Dieses Konzept wird zunächst in Grundzügen in Abschnitt 4.2 vorgestellt. Dabei werden die grundlegenden Entwurfstechniken benannt und deren Umsetzung in Rebeca anschließend in Abschnitt 4.3 beschrieben. Hauptaspekt der dort beschriebenen Betrachtung bil-

det die Komposition von Plugins an den Brokern. Zu diesen Plugins gehören auch diejenigen, welche die Migration, Replikation und die der Ausführung der Basiskomponenten vorausgehende Strategie umsetzen. Dies ist Bestandteil des Abschnitts 4.4. Im letzten Abschnitt des Kapitels (4.5) findet eine abschließende Diskussion der Implementierung statt.

4.2 Grundlegende Techniken und Entwicklungen

Die Implementierung von REBECA ist zunächst einmal die eines Forschungsprototyps. Das bedeutet vor allem, dass eine Middleware für unterschiedliche Forschungsfragen gestaltet wurde und auch zukünftig genutzt wird. Damit sind zwei grundlegende Aspekte verbunden, zum einen (1) die Anpassung der Anforderungen an die Middleware an aktuelle Forschungsfragen, zum anderen (2) eine ständige Überarbeitung der Middleware. Daraus lässt sich ableiten, dass REBECA immer die Anforderungen der jeweiligen Forschungsfragen erfüllen muss, und nicht alle möglichen Anforderungen zur gleichen Zeit. Gleichzeitig wächst die Funktionalität der Middleware. Führt vor allem der erste Aspekt zu einer Architektur, die ständig erweitert wird, benötigt der zweite Aspekt regelmäßig eine Überarbeitung des Gesamtsystems, um die wachsende Komplexität beherrschbar zu halten und den aktuellen Anforderungen der Anwendungsentwicklung Genüge zu tun.

Die Implementierung von REBECA erfolgt objektorientiert mit der Programmiersprache Java [82]. Nach den Prinzipien der Objektorientierung inkl. der Vererbung erfolgt die Trennung von Schnittstellendefinition und Implementierung. So werden für die jeweiligen Klassen zunächst die Interface-Klassen angelegt, welche anschließend von einer abstrakten Basisklasse implementiert werden. Von dieser abstrakten Klasse erbt anschließend eine Default-Klasse, welche die grundlegendsten Schnittstellen implementiert. Von diesen Default-Klassen lassen sich entsprechend weitere Klassen mit individuellen Funktionalitäten ableiten.

Die Architektur von REBECA, mit dem Ziel der Unterstützung unterschiedlicher und wechselnder Funktionalitäten, basiert auf der Idee der flexiblen Komposition von Funktionalitäten, welche in Bezug auf Software auch als *Features* bezeichnet werden. Im Umfeld der Softwaretechnik wird mit dem Begriff Feature ein unterscheidbares Merkmal oder eine Eigenschaft eines Softwareartefaktes beschrieben und gleichzeitig die funktionale Gliederung der Software in den Vordergrund gestellt [13]. So steht ein Feature für eine implementierte Funktionalität eines Softwaresystems. Funktionalitäten sind jedoch keineswegs isoliert, sondern können miteinander kombiniert werden bzw. lassen sich aufeinander aufbauen. Die Kombination von Funktionalitäten wird in der Literatur auch als *Feature Composition* beschrieben, dabei werden Funktionalitäten und die damit zusammenhängenden Softwareartefakte zu neuen Artefakten zusammengefasst, welche die Funktionalitäten der eingebrachten Artefakte zusammenfassen. Allerdings sind Funktionalitäten nicht zwangsläufig miteinander kombinierbar, ebenso möglich ist es, dass sich Features bewusst konträr zueinander verhalten, sich

ungewollte Seiteneffekte zwischen Features herausstellen oder sich die Features bestenfalls neutral zueinander verhalten. Die Aufgabe sicherzustellen, dass Features und ihre Umsetzungen in Form von Plugins sich nicht gegenseitig negativ beeinflussen, obliegt primär den Featureentwicklern und Verantwortlichen für die Featurekomposition.

Der Idee der Feature Composition folgend, besteht REBECA aus einem Kernsystem mit Basisfunktionalität wie Filterung, Matching und grundlegenden Routingmechanismen sowie weiteren Artefakten, welche flexibel die gewünschten Funktionalitäten bereitstellen und dem Rumpfsystem hinzugefügt werden. Dazu gehören bspw. die Integration von Advertisements und Mechanismen zur Integration von Ausfallsicherheit oder das Caching von Ereignisnotifikationen.

Die Bereitstellung von Funktionalitäten, seien es Basis- oder zusätzliche Funktionalitäten, erfolgt in gegossener, sprich implementierter Form, wozu auch strukturelle Aspekte gehören, schließlich ist die Funktionalität durch Module oder Komponenten implementiert und verfügbar. Zuzüglich der eigentlichen Funktionalität benötigen die Module/Komponenten Funktionalitäten bzw. Andockpunkte zur Verbindung mit der Basisfunktionalität und weiteren (funktionalen) Komponenten. Die Einpassung der Komponenten muss selbstverständlich bereits beim Entwurf des Systems eingearbeitet werden, um sicherzustellen, dass die Basisfunktionalität und die Komponenten interagieren können. Die dabei entstehende Struktur ist die *Softwarearchitektur*. Diese bestimmt die Struktur des Softwaresystems, deren Elemente und die Beziehungen zwischen diesen [87]. Eine besondere Anforderung an die Architektur ist dabei die Komponierbarkeit der Architektur und damit die Möglichkeit, Komponenten und damit Funktionalitäten in einem System zusammenzufügen. Eine Softwarearchitektur ist dann komponierbar, wenn sie in der Lage ist, ihre Elemente so auf eine definierte Art und Weise zusammenzufügen, dass im Ergebnis ein neues und komplexeres System oder Architekturelement entsteht [169].

Aus den genannten Grundzügen des Entwurfs von REBECA lassen sich demnach drei Hauptanforderungen ableiten. Zunächst müssen (1) die gewünschten Funktionalitäten in gekapselter Form als Komponenten vorliegen, welche (2) gemäß ihres Designs und der vorhandenen Schnittstellen miteinander kombiniert werden können und (3) die zusammengeschlossenen Komponenten dürfen sich nicht ungewollt negativ beeinflussen, sodass die gewünschten Funktionalitäten umgesetzt werden können [169].

4.3 Rebeca - Broker und Plugins

Kernstück der Architektur der Implementierung von REBECA ist die Komposition von Funktionalitäten. Da Funktionalitäten durch die Broker bereitgestellt und damit implementiert werden, findet sich die Architektur bzw. die Komposition der REBECA Middleware hier wieder [170]. Die Idee der Feature Composition wird dadurch umgesetzt, dass nur noch zwei Arten von Elementen exis-

tieren, nämlich Broker und Plugins. Der Broker implementiert die Basisfunktionalität, an der die Plugins angedockt werden. Die eigentliche Middleware- bzw. Publish/Subscribe-Funktionalität wird dabei durch die Plugins realisiert. Die Aufgabe des Brokers ist hingegen lediglich für die Vernetzung und Verwaltung der Plugins zu sorgen, eine weitergehende Funktionalität bietet er nicht. Eine weitere Eigenschaft von REBECA ist es, dass die Entscheidung darüber, welche Plugins und damit welche Funktionalitäten ausgewählt werden, nicht allein zur Entwurfszeit, sondern auch noch zur Laufzeit getroffen wird. Diese Besonderheit erleichtert die Anpassung der bereitgestellten Funktionalität. Allerdings erfordert dies auch ein Konzept, wie der Broker, der sich nicht umfangreicher als ein Anwendungscontainer darstellt, die Plugins zur Laufzeit einbindet und verwaltet. Dazu gehört auch die Versorgung mit den benötigten Ereignisnachrichten sowie die Weiterleitung der produzierten Ereignisse. Die Brokerarchitektur basiert auf den Konzepten der Ereigniskanäle und Verarbeitungsschritte [169]. Die Verarbeitung von Ereignisnachrichten verläuft dabei stufenweise in drei Phasen. In der ersten Phase werden Nachrichten entgegengenommen, in der zweiten verarbeitet und in der dritten und letzten an benachbarte Broker bzw. Clients weitergeleitet. Die Publish/Subscribe-Logik wird in der mittleren Stufe vollzogen, während in der ersten vor allem Empfangsbestätigungen für Nachrichten versendet und Nachrichten herausgelöst bzw. deserialisiert werden. Zur Publish/Subscribe-Logik gehören auch die Routingentscheidungen, Filter und weitere Komponenten. In der dritten Stufe werden die Nachrichten für den Versand vorbereitet, bspw. durch Verschlüsselung und Serialisierung und den Ausgangskanälen übergeben. Zwischen den einzelnen Verarbeitungsstufen existieren unterschiedlich viele Kanäle. So erreichen einen Broker mehrere Eingangskanäle in der ersten Stufe, ebenso verlassen den Broker in der dritten Verarbeitungsstufe wieder mehrere Ausgangskanäle. So gehen Nachrichten in Stufe eins ein, werden in Stufe zwei verarbeitet und verlassen den Broker in Stufe drei wieder. Lediglich in Stufe zwei während der Verarbeitung existiert nur ein durchgehender Kommunikationskanal. Innerhalb dieses Kanals wird die Verarbeitung der Nachrichten vorgenommen, indem sie schrittweise die eingebundenen Komponenten durchlaufen. Die eingehenden Nachrichten, ihre Verarbeitung und die ausgehenden Nachrichten erfordern die Umwandlung mehrerer Eingangsströme in einen Strom sowie die Rückumwandlung in mehrere Ausgangsströme. Die erste Umwandlung erfolgt mit Hilfe einer Warteschlange, in der alle eingehenden Nachrichten nach dem Eintreffen geordnet zwischengespeichert und anschließend in der zweiten Stufe individuell verarbeitet werden. Nach erfolgter Verarbeitung werden die Nachrichten, geordnet nach dem jeweiligen Empfänger, in den entsprechenden Ausgangskanal einsortiert und danach weitergeleitet. Abbildung 4.1 zeigt den schematischen Aufbau eines Brokers mit den genannten drei Verarbeitungsstufen und den hinzugefügten Komponenten sowie den korrespondierenden Ein- und Ausgabekanälen.

Die jeweiligen Komponenten, die die Nachrichten in Stufe 2 bearbeiten, haben die Möglichkeit, die Nachrichten global zu verändern, aus dem Verarbeitungsstrom zu entfernen oder neue Nachrichten hinzuzufügen. Daher spielen die

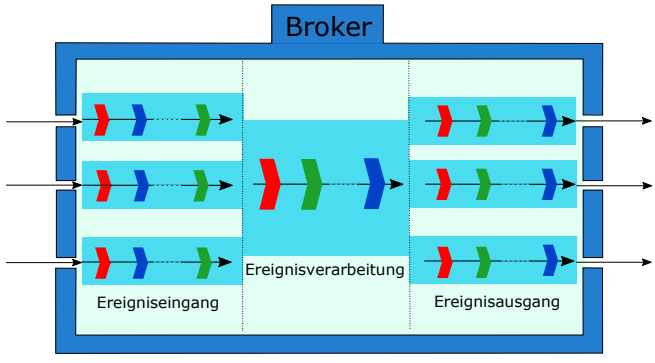


Abbildung 4.1: Schematische Brokerdarstellung mit hinzugefügten Komponenten und korrespondierenden Ein- und Ausgabekanälen [169]

Reihenfolge der hinzugefügten Komponenten und die Folge der Nachrichten innerhalb der Eingangsströme und insbesondere im zentralen Verarbeitungsstrom eine wichtige Rolle. Denn das Verändern, das Hinzufügen oder Herauslösen einzelner Ereignisnachrichten durch ein Modul erfassen auch die nachgeordneten Funktionsmodule und haben damit globalen Einfluss.

Neben der Zuordnung der Funktionsmodule an die jeweiligen Plätze übernimmt der Broker auch die Umwandlung der unterschiedlichen Eingangsströme in den zentralen Verarbeitungsstrom und die Rückübertragung aus dem zentralen in die verschiedenen Ausgangsströme. Durch eine exakte Konfiguration ist brokerseitig festgelegt, welche Komponente an welche Stelle und welchen Kanal eingefügt wird. Damit wird sichergestellt, dass die Komponenten und die Komposition von Funktionalitäten wie gewünscht arbeiten und das definierte Verhalten des Brokers gewährleistet ist [169].

Da die Plugins die eigentliche Funktionalität implementieren, ermöglicht erst deren Einbindung in die Broker die Umsetzung der gewünschten Eigenschaften. Die Einbindung zur Laufzeit ermöglicht zudem ein anpassungsfähiges Verhalten der Broker je nach Umfang und Anforderung. Grundsätzlich sollte ein Plugin eine Funktionalität bereitstellen und an den Broker angebunden sein. Bei komplexen Funktionalitäten stößt dieses Vorgehen jedoch an seine Grenzen, da eine umfangreiche Funktionalität damit einhergeht, dass ebenso vielfältige Zugriffs- und Änderungsprozesse an Nachrichten notwendig werden. An dieser Stelle ist wiederum auf die Koordination der einzelnen Funktionsmodule zu achten. Eine Aneinanderschachtelung einzelner Komponenten löst daher nicht das Problem, unterschiedliche und vor allem komplexe Funktionalitäten abzubilden. Die Lösung besteht darin, dass unter Beibehaltung der ursprünglichen dreistufigen Verarbeitung, jedes Plugin eine Komponente für jeden Abarbeitungsschritt und Nachrichtenkanal vorhält. Die in die Hauptverarbeitungskette eingebetteten Plugins stehen dabei an einer ihnen zugewiesenen Position und haben, das sei hier nochmals betont, die Möglichkeit, Ereignisnachrichten mit globaler Wir-

kung zu verändern, zu löschen oder neue Nachrichten hinzuzufügen. Um dies durchführen zu können, werden zur Umsetzung der Funktionalität entsprechende Datenstrukturen und implementierte Verarbeitungen der Ereignisnachrichten benötigt. Aufgrund des Einflusses auf die Aktivitäten und Ergebnisse des Brokers werden diese Verarbeitungsblöcke auch als *Engines* bzw. *Broker Engines* bezeichnet [169], bspw. implementiert die Routing Engine den ausgewählten Routing Algorithmus. Die Aktivitäten, die außerhalb der eigentlichen Funktionalität liegen, finden sich jedoch außerhalb der Engines in anderen Stufen wieder. Hierbei orientiert sich der Aufbau der Plugins an der stufenorientierten Architektur, welche bspw. auch dem OSI-Referenzmodell [218] oder generell einem Strukturmodell wie einer Middleware oder einem Netzwerkmodell zu Grunde liegt. Zu diesen nicht-funktionalen Tätigkeiten rund um die Organisation ein- und ausgehender Nachrichten gehören bspw. die Deserialisierung der eingehenden und Serialisierung der ausgehenden Nachrichten. Ein Plugin besitzt stets sowohl einals auch ausgehende Nachrichtenströme, sodass einer Teilkomponente für eingehende Nachrichten stets eine Teilkomponente für ausgehende Ereignisströme gegenüber steht. Beide zusammenhängenden Elemente bilden dabei eine gemeinsame Ereignissenke, bzw. *Event Sink*. Ereignissenken selbst sind wiederum stapelbar und erlauben so die gezielte Bearbeitung von Ereignisnachrichten. Ein REBECA-Plugin besteht daher immer aus einer Engine und einer dazugehörigen Ereignissenke. Während die Engine in der zentralen Verarbeitungsroutine eingebunden ist, befinden sich die Senken in den Verbindungen der Broker bzw. der Clients. Abbildung 4.2 verdeutlicht die Integration von Plugins mit ihren Engines und den dazugehörigen Senken in einen Broker.

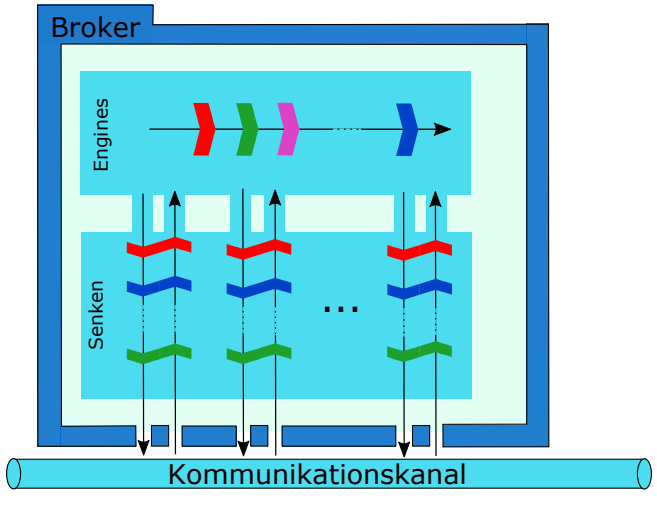


Abbildung 4.2: Schematische Brokerdarstellung mit hinzugefügten Plugins [169]

Die Kombination mehrerer Funktionalitäten und damit von Plugins ist bei REBECA auf unterschiedliche Art und Weise möglich. Zwar sind Plugins in einer

idealen Welt nur von sich selbst abhängig und damit unabhängig von anderen Plugins, in der Praxis ist dies bei komplexen Funktionalitäten nicht immer durchzuhalten. So können in REBECA Funktionalitäten durch die Anordnung der Plugins und damit durch Veränderung der Ereignisnachrichten bzw. durch zusätzliche Kontrollflussnachrichten miteinander kombiniert werden. Darüber hinaus ist die Gestaltung der Plugins nicht beschränkt, sodass sich zusätzliche Schnittstellen und gemeinsam genutzte Speicher integrieren lassen.

Der Broker als Ausführungscontainer bietet lediglich eine simple Ausführungsumgebung und Schnittstellen zum Hinzufügen und Herauslösen von Engines und Senken an, welche unabhängig voneinander hinzugefügt oder herausgelöst werden können. Während Engines als Implementierung der eigentlichen Funktionalität explizit hinzugefügt werden, erfolgt das Hinzufügen der Senken indirekt durch ausgeführte Weiterleitungsengines, sobald eine weitere Verbindung zu einem (benachbarten) Broker oder einem Client aufgebaut wird. Wie auch bei der Einbindung und Ausgliederung der Engines ist der Broker ebenso für die Einbindung der Senken an der richtigen Stelle innerhalb des Brokers verantwortlich. Wird eine Verbindung abgebaut, erfolgen auch das Herauslösen der Senke und der Verarbeitungskette aus dem Broker. Neben dem Einbinden und Herauslösen von Engines und Senken über Brokerschnittstellen ermöglicht die Verwaltung des Brokers auch das Durchsuchen der eingebundenen Komponenten, die Suche nach speziellen Komponenten und mit den Komponenten verbundenen Engines und Senken. Welche Methoden dabei implementiert werden, beschreibt das *Broker Interface*, das durch die abstrakte Klasse *Abstract Broker* implementiert wird und von dem sämtliche Broker abgeleitet sind.

Neben den Brokern und Plugins existiert eine weitere Komponente: der Client. Clients sind die Repräsentanten von Anwendungen, die den Austausch von Informationen durch die Kommunikationswege der Middleware nutzen. Die Anwendungen selbst sind nicht Bestandteil der Middleware, wobei die Clients und Broker über eine einheitliche Schnittstelle kommunizieren. Im Fall von REBECA sind dies Schnittstellen zur Veröffentlichung von Interessen, Ankündigungen und Ereignisnachrichten. Solche Schnittstellen sind grundsätzlich in ihrem Umfang begrenzt, schließlich ist es die Aufgabe einer Middleware, Funktionalitäten in Richtung der Anwendungsebene zu verbergen. Allerdings kommunizieren Broker natürlich auch untereinander, benutzen dazu jedoch komplexere Protokolle für einen effizienteren Informationsaustausch. Werden sowohl komplexe als auch einfache Schnittstellen und Protokolle angeboten, müssen Broker dies unterscheiden, die zweifache Anzahl möglicher Kommunikationspartner verkompliziert den Aufbau und die Implementierung und Wartung der Protokolle. Darüber hinaus beeinträchtigen unterschiedliche Kommunikationsabläufe durch gegenseitige Abhängigkeiten die Komposition von Brokerfunktionalitäten [169].

Ähnlich wie die Implementierung der Broker erfolgt auch die Implementierung der Engines. Neben der eigentlichen Funktionalität implementieren auch die Engines bestimmte Schnittstellen, welche in einer Interface-Klasse gesammelt sind. Dazu gehört die Methode *Process* zur Bearbeitung der eingehenden Ereignis-

nachrichten, die Suche nach der nachfolgenden Engine auf Brokerebene sowie eine *Plug*-Methode zur Verkettung der benötigten Senken für jede Verbindung, resp. für jeden angeschlossenen Client. Eine explizite *Unplug*-Methode existiert allerdings nicht, stattdessen werden beim Schließen des Kommunikationskanals die gesamte Kette von Senken deaktiviert und die genutzten Ressourcen automatisch freigegeben. Spezielle, funktionale Schnittstellen sind nicht vorgegeben und werden individuell implementiert. Grundlage für die Implementierung der Engines bildet die abstrakte Klasse *AbstractEngine*.

Statt den Broker mit zwei unterschiedlichen Kommunikationsimplementierungen auszustatten, werden in REBECA Clients nicht anders als Broker behandelt, sodass die Broker nur eine einheitliche Implementierung zur Kommunikation anbieten müssen. Damit werden zwar die Clients in puncto Kommunikation komplexer, allerdings wird als Ausgleich dafür bei den Clients die gleiche Architektur verwendet wie bei den Brokern. Funktionalitäten werden damit ebenfalls durch Plugins realisiert und *Client Sinks* werden, analog zu den Ereignissenken, nach dem gleichen Prinzip hinzugefügt. Zur Implementierung einer Funktionalität arbeiten die Senken broker- und clientseitig zusammen und bilden dazu eine Einheit. So befindet sich in jedem Broker ein Element in seiner Senke, für den auch ein Element in der Senke des Clients vorhanden ist. Da in Aufbau und Funktion gleich, können die Elemente der Senke des Clients natürlich ebenfalls Ereignisnachrichten verändern, löschen oder neu hinzufügen. Die Nutzung von hinzufügbaren Ereignissenken auch auf Seiten der Clients bietet einige Vorteile [169]. Zum einen werden die Logiken von Clients und Brokern getrennt, zum anderen wird die Implementierung vereinfacht, da eine clientseitige Senke nur für eine Anwendungskomponente zuständig ist, während Broker für mehrere Anwendungen gleichzeitig zuständig sind. Senken von Clients sind zudem einfacher umzusetzen, da nur ein Teil der komplexeren Brokerfunktionalität benötigt wird. Die Implementierung der Senken erfolgt anhand einer spezifizierten Schnittstellenklasse. Dabei liegt das Hauptaugenmerk auf der Verarbeitung, dem Löschen und dem Hinzufügen von Ereignisnachrichten, welche aus dem Verarbeitungsstrom entnommen („in“) und in den Strom abgegeben („out“) werden. Wie bereits erwähnt, ist die Eingliederung der Senke in die Kette aller Senken mithin entscheidend für die weitere Bearbeitung der Ereignisnachrichten. So werden die Ereignisnachrichten nach erfolgter Bearbeitung an die nächstgelegene Senke weitergegeben. Wie auch bei den vorherigen Schnittstellenklassen existiert für die Senken eine vordefinierte Klasse mit Basisfunktionalität, die *AbstractSink*, welche grundsätzlich die Weiterleitung von Ereignisnachrichten an die folgende Senke bereits ausimplementiert vorzuliegen hat.

Die Vorgehensweise, aus Brokersicht Clients und benachbarte Broker gleich zu behandeln, hat allerdings auch Nachteile [169]. So führt der Aufwand während der Bearbeitung wie z.B. Serialisierung, Deserialisierung und der Transport von Nachrichten innerhalb der Ereignissenken immer dann zu zusätzlichem Aufwand, wenn Clients und Broker auf demselben Rechner ausgeführt werden. Allerdings besitzt REBECA auch hier eine elegante Lösungsstrategie. Befinden sich sowohl Client als auch Broker auf demselben Host, kann auf das Hinzufügen der jewei-

ligen Senken verzichtet und so die Effizienz lokaler Publish/Subscribe-Systeme signifikant erhöht werden.

4.4 Umsetzung und Strategie

Wie im vorigen Kapitel 4.3 dieser Arbeit dargelegt ist, sind die grundsätzlichen, funktional tragenden Elemente der Middlewareimplementierung zum einen der Broker, mit dessen eingebundenen Engines und Senken, und zum anderen die Komponenten, welche die Anwendungen darstellen und Schnittstellen entsprechend der Publish/Subscribe-Kommunikationsinfrastruktur vorhalten.

Beim Konzept wie auch bei der Umsetzung werden die beiden unterschiedlichen Stufen des Platzierungsansatzes beschrieben, dies sind die initiale Platzierung sowie die laufende Platzierung. Die initiale Platzierung benötigt zur Ausführung einen Koordinator, also einen Broker, der für die Platzierung der Anwendungskomponenten zuständig ist. Dieser Koordinator wird solange benötigt, bis die Anwendungskomponente platziert wurde. Daher ist die Lebensdauer des Koordinators für eine spezifizierte Anwendung begrenzt. Der Gegen- bzw. Mitspieler des Koordinators ist eine Monitoringinstanz, welche die verfügbaren Ressourcen des Brokers kennt und dem Koordinator die für die Platzierung benötigten Informationen, sprich den Matchingwert, ermittelt und zurücksendet. Gleichzeitig startet diese Instanz das Deployment der zu verteilenden Anwendungskomponente, sobald der Broker dazu den Zuschlag bekommt. Gegenüber dem Koordinator ist die Monitoring- und Managementkomponente an die Lebenszeit des Brokers geknüpft, da solange der Broker existiert, Platzierungsanfragen eingehen können und beantwortet werden müssen. Der Austausch von Informationen (z.B. Platzierungsanfrage, Anfrage und Rückgabe von Matchingwerten, Versand der Anwendung und Rückmeldungen über Platzierungen) während der initialen Platzierung wird dagegen mit Bordmitteln der Publish/Subscribe-Middleware umgesetzt (Austausch von Subscriptions, Notifications und Advertisements), sodass eine einfache Anweisungszuweisung möglich ist. Die initiale Anwendungszuweisung erfolgt in einer Engine innerhalb der Broker-Abarbeitungskette mit einer dazugehörigen Senke. Der in dieser Arbeit vorgestellte Ansatz basiert ebenfalls auf der Kombination beider Aspekte. Auf der einen Seite stehen dabei die Fähigkeiten der Anwendungen im Mittelpunkt, die Basisoperationen umzusetzen. Auf der anderen Seite steht die Fähigkeit der Middleware, die Steuerung der Umsetzung der Basisoperationen auszuführen.

4.4.1 Initiale Platzierung

Die initiale Platzierung ist so konzipiert, dass sie mit der vorhandenen Publish/Subscribe-Infrastruktur und Funktionalität umgesetzt werden kann. Es werden lediglich die Rolle des Koordinators temporär und die Rolle der platzierungsfähigen Broker dauerhaft zusätzlich ausgeführt. Für die initiale Platzierung wer-

den Ankündigungen, Ereignisnachrichten und Subskriptionen eingesetzt. Wie in Kapitel 3.4.3 vorgestellt wurde, kommuniziert ein Koordinator mit mehreren platzierungsfähigen Brokern, wobei aus der zweiten Gruppe ein oder mehrere Broker als Ausführungsorte ausgewählt werden und die nicht ausgewählten Broker die Rolle der nicht ausgewählten Broker einnehmen. Das Anwendungsfalldiagramm und das Rollenmodell zur initialen Platzierung zeigen die Abbildungen 3.3 und 3.4 in Kapitel 3 dieser Arbeit.

Ereignisebene - Kommunikation mittels Ereignissen

Während der initialen Platzierung werden folgende Kommunikationselemente genutzt: (1) Advertisement mit Angabe der notwendigen Ausführungsbedingungen, (2) Subskription auf Advertisement, (3) Anforderungsereignis mit hinreichenden Bedingungen, (4) Übergabe der individuell ermittelten Matchingwerte, (5) Zuweisungsereignis und (6) Nichtzuweisungsereignis sowie die (7) Ausführungsbestätigung bzw. die (8) Nicht-Ausführungsbestätigung. Die Subskription auf das initiale Advertisement sowie die Widerrufe von Subskriptionen und Ankündigungen werden nicht als eigene Nachrichtentypen bzw. Kommunikationsereignisse aufgefasst. Zur Erinnerung: ein Ablaufplan zur initialen Platzierung findet sich in Abbildung 3.5. Damit bleiben als eigenständige Kommunikationsklassen (1) das Advertisement, (3) das Anforderungsereignis, (4) die Ereignisse zur Übermittlung der Matchingwerte sowie die (5) Zuweisungs- und (6) Nichtzuweisungsereignisse und die (7) Ausführungsbestätigung über die erfolgreiche initiale Platzierung und (8) die Nachricht über die nicht erfolgreiche Platzierung. Eine Übersicht über die zusätzlichen Klassen findet sich in den Abbildungen 4.3 und 4.4 wieder. Grundsätzlich sind sowohl das initiale Ad-

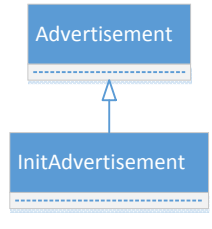


Abbildung 4.3: Klassenhierarchie Advertisement initiale Platzierung

vertisement aus Abbildung 4.3 als auch die zusätzlichen Ereignisse aus Abbildung 4.4 Name/Wert-Paare bzw. die Ankündigung dazu. Die besondere Rolle dieser Kommunikationsbestandteile wie auch die spezielle Unterscheidung in notwendige und hinreichende Anforderungen sowie Bestätigungs- und Negativbestätigungsnachrichten rechtfertigen jedoch die besondere Ausgestaltung als eigene Klassen.

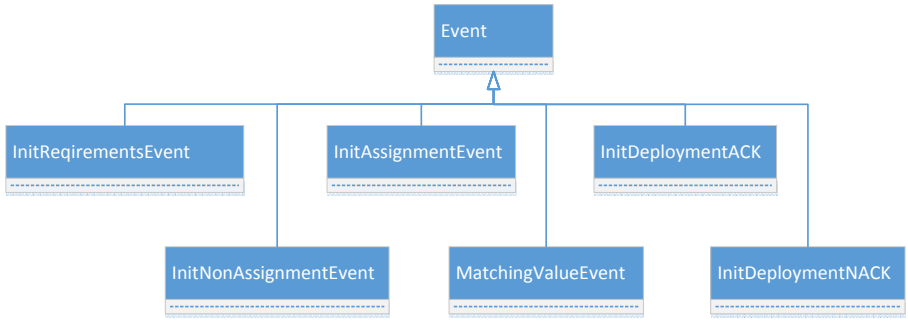


Abbildung 4.4: Klassenhierarchie Kommunikationsereignisse initiale Platzierung

Komponenten und Engines

Bei der initialen Platzierung geht es um die erstmalige Platzierung von Komponenten und darum, zunächst eine gültige Platzierung zu finden, welche anschließend durch Migration und Replikation schrittweise verbessert wird. Um die Komponentenplatzierung schrittweise zu verbessern, werden entsprechende und einheitliche Schnittstellen für die Platzierungsanpassung umgesetzt. In Bezug auf diese Arbeit sind dies die Schnittstellen *Migratable* und *Replicable*. Für die initiale Platzierung selbst wird die Implementierung der Schnittstellen allerdings nicht benötigt, die genaue Betrachtung der (Anwendungs-) Komponenten und ihrer Schnittstellen erfolgt daher erst in Abschnitt 4.4.2 im Unterabschnitt Komponenten und Engines - Migration und Replikation unter Verwendung der Abbildung 4.10.

Neben der Ebene der Komponenten ist für die Ausführung der initialen Platzierung die Ausführung von Engines notwendig, welche die Platzierungslogik sowie das Aussenden und Empfangen von initialen Platzierungsereignissen koordinieren. Jeder Broker, der an der initialen Ereignisplatzierung teilnimmt, führt die *BasicInitPlacementEngine* aus. Diese Engine implementiert die *InitPlacementEngine*. Die Hierarchie der Engines verdeutlicht Abbildung 4.5. Mit diesem Aufbau von Schnittstelle und der sie ausführenden Klasse entspricht die Implementierung der Engine dem Entwurfsprinzip von Rebeca.

Jede *BasicInitPlacementEngine* kann die Rolle des Platzierungskoordinators (als Verantwortlichen für das Finden und Zuweisen einer initialen Platzierung) und die Rolle des (potentiell) ausführenden Brokers einnehmen. Aus Gründen der Übersichtlichkeit sind die Bezeichnungen der Engines in Abbildung 4.6 durchnummeriert. Startpunkt ist der Aufruf einer Methode zur Platzierung einer bestimmten Komponente (*component*) mittels *doPlace(component)* an *Broker1* in Abbildung 4.6.

Die Abbildung zeigt die auf *Broker1*, dem Koordinator, ausgeführte *InitPlacementEngine1* sowie drei weitere Engines auf drei weiteren Brokern (*InitPlacementEngine2*, *InitPlacementEngine3* und *InitPlacementEngine_n* auf den Bro-

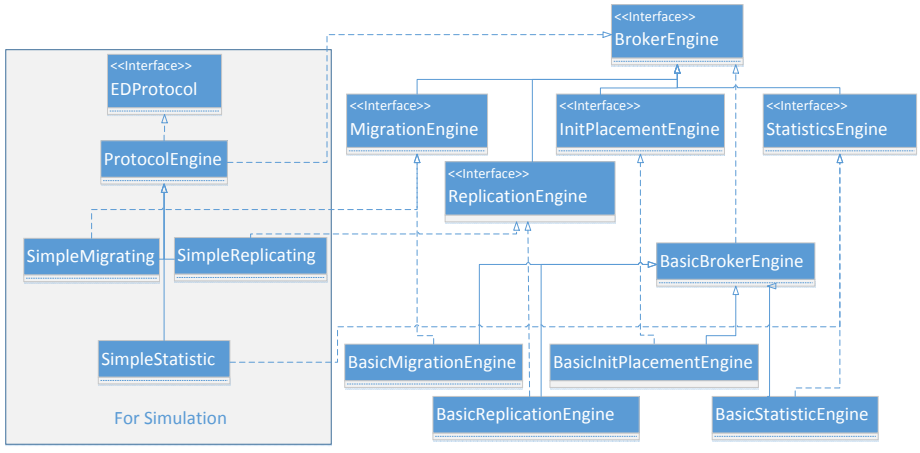


Abbildung 4.5: Klassenhierarchie Engines

kern *Broker2*, *Broker3* und *Broker_n*). Nach der Abgabe einer Ankündigung des Anforderungsereignisses und einer ersten Auswertung subskribieren sich lediglich die Broker *Broker3* und *Broker_n* mit den jeweils ausgeführten *InitPlacementEngine3* und *InitPlacementEngine_n*, wobei nach Auswertung der Matchingwerte *Broker3*, der *InitPlacementEngine3* ausführt, den Ausführungszuschlag erhält und entweder die erfolgreiche Ausführung bestätigt oder die Nichtausführung per Ereignis meldet.

4.4.2 Laufende Platzierung

Für die Ausführung der laufenden Platzierung benötigen die Broker der Middleware Erweiterungen unterschiedlicher Ebenen, wie etwa die (Anwendungs-) Komponenten selbst, die Kommunikation mittels dafür geeigneter Ereignisse, sowie die für eine Platzierungsentscheidung notwendigen Informations- und Auswertungswerkzeuge. Dazu gehören die jeweiligen Strategien und die die Platzierungsentscheidung ausführenden Engines auf Brokerebene. Weiterhin sind für die Durchführung der Experimente weitere Implementierungen nötig. Dies sind Protokolle, welche die Migration, Replikation und die Statistik im Rahmen einer Simulation umsetzen. Zunächst werden in diesem Abschnitt die Erweiterungen von REBECA selbst und damit ohne die für die Simulation notwendigen Komponenten beschrieben. Dazu werden die Ereignisebene, die Komponentenebene, die Ebene der Engines und der Strategie betrachtet. Die Beschreibung der Simulationskomponenten erfolgt in Abschnitt 5.3.3.

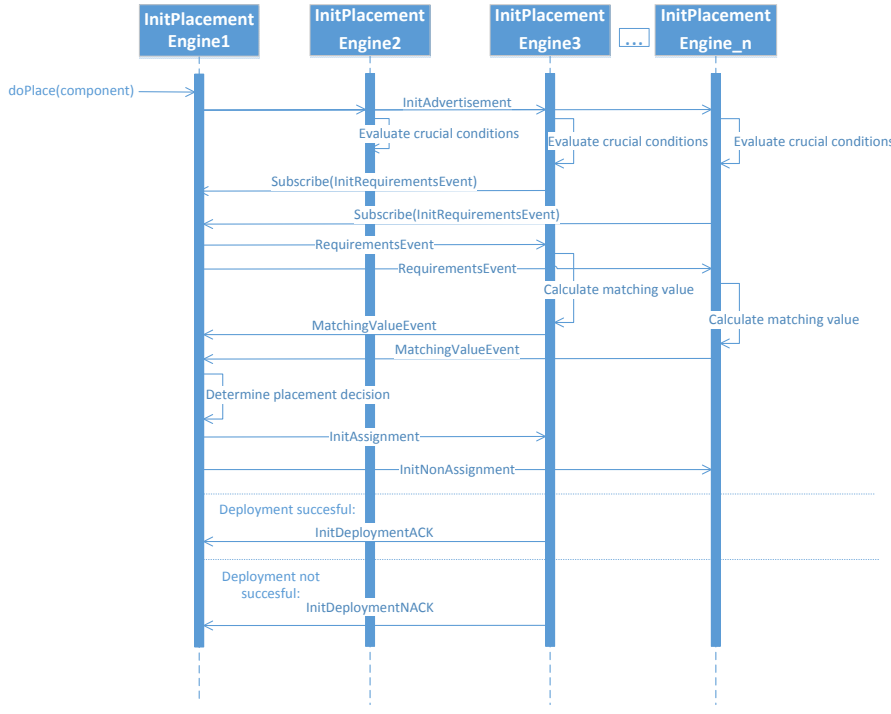


Abbildung 4.6: Sequenzdiagramm initiale Platzierung

Ereignisebene - Kommunikation mittels Ereignissen

Zur Umsetzung der laufenden Platzierung gehören auch die für die Kommunikation notwendigen Ereignisse. Nach dem Prinzip von Ankündigung, Initialisierung, Verschiebung und Bestätigung werden ihrem Einsatzzweck entsprechend typisierte Ereignisse eingesetzt und die jeweiligen Ereignisklassen von der Klasse *Event* abgeleitet. Dies unterstreicht Abbildung 4.7.

Replikation und Migration sind grundsätzlich in ihrem Ablauf ähnlich. So wird bei der Migration eine Anwendung oder Anwendungskomponente verschoben, während bei der Replikation nach erfolgter Aufteilung des Ereignisraums alle nach der Replikation entstandenen Anwendungen bzw. Anwendungskomponenten migriert werden. Damit baut die Replikation konzeptionell auf der Migration auf und die für den Ablauf von Migration und Replikation zuständigen Ereignisse orientieren sich in ihrer Bezeichnung an der Migration. Eine Übersicht der Klassen findet sich in Abbildung 4.7.

Der konzeptionelle Ablauf einer Migration wurde in Abschnitt 3.5.5 beschrieben und wird durch Abbildung 3.38 nochmals verdeutlicht. Bei der Migration kommunizieren zwei benachbarte Broker und migrieren eine vorhandene Komponente. Die Kommunikation erfolgt dabei nicht über das Publish/Subscribe-

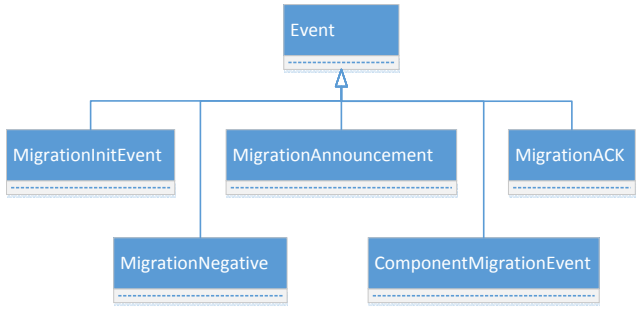


Abbildung 4.7: Klassenhierarchie Kommunikationsereignisse Migration

Netzwerk sondern bidirektional zwischen den beteiligten und benachbarten Brokern. Im Gegensatz zur losen Kopplung über das Publish/Subscribe-Netzwerk, kommunizieren die Broker bei der Migration verbindungsorientiert auf der Basis von TCP.

Bei der Migration wird möglicherweise auch ein gegenseitiger Austausch von Komponenten ausgehandelt (siehe Abbildung 3.42). Die Kommunikationsereignisse zur Umsetzung der Migration bilden den Migrationsprozess nach, eine Übersicht dazu findet sich in Abbildung 4.8. In dieser Abbildung sind zunächst die Engines zweier Broker als Akteure dargestellt (*MigrationEngine1* und *MigrationEngine2*). Während *MigrationEngine1* in der Zuständigkeit des Brokers liegt, welcher die laufende Komponente beheimatet, soll diese Komponente in Richtung des Brokers migriert werden, auf dem die *MigrationEngine2* ausgeführt wird. Dies ist nur möglich, wenn auf allen beteiligten Brokern die Migrationsengine ausgeführt wird, denn auch hier gilt das Konzept der gegenseitigen Kommunikationspartner in REBECA. Der Migrationsprozess wird durch einen expliziten Aufruf durch die *doMigrate()*-Methode oder durch Eintreffen eines *MigrationInitEvents* gestartet. Dies wird innerhalb der Strategie und Statistik entschieden, deren Ablauf im weiteren Verlauf dieses Kapitels näher erläutert wird. Nach dem Start der Migration wird zunächst eine Ankündigung, das *MigrationAnnouncement* versendet. Darin sind die Anforderungen der zu migrierenden Anwendung enthalten. Können diese erfüllt werden und ist damit eine Migration in Richtung des Brokers von *MigrationEngine2* möglich, sendet diese eine Nachricht *MigrationACK*, dass eine Migration durchführbar ist. Gleichzeitig werden die zur Anwendungsausführung benötigten Ressourcen verbindlich reserviert. Daraufhin sendet die *MigrationEngine1* ein *ComponentMigrationEvent*, welches die serialisierte Komponente enthält. Falls die Platzierung nicht möglich ist, erhält die *MigrationEngine1* ein *MigrationNegative*-Ereignis und die Migration wird abgebrochen. Da beim Versand eines *MigrationACK* die benötigten Ausführungsressourcen verbindlich reserviert wurden und dies durch die verbindungsorientierte Kommunikation als sicher angenommen wird, wird auf ein explizites Bestätigungsereignis durch die *MigrationEngine2* an die *MigrationEngine1* verzichtet. Ein Sonderfall ist der Tausch von Komponenten, falls die Ka-

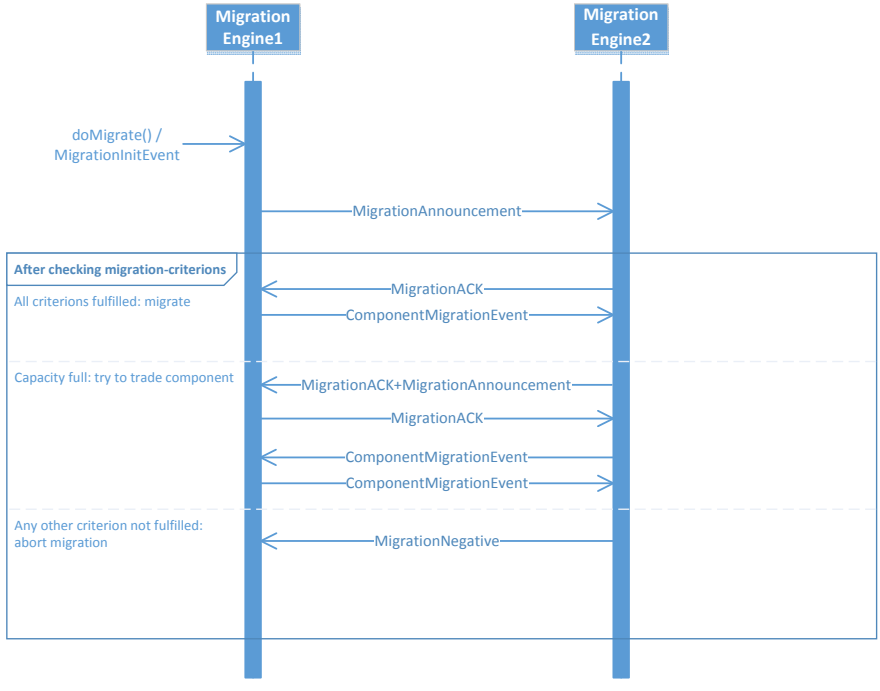


Abbildung 4.8: Kommunikationsfluss zur Migration mittels Ereignissen

pazität des aufnehmenden Brokers der *MigrationEngine2* ausgeschöpft ist. In diesem Fall wird ein zusätzliches *MigrationACK* mit *MigrationAnnouncement* in Richtung *MigrationEngine1* versendet. In der Grafik ist nur der positive Verlauf eingezeichnet, sodass am Ende gegenseitig ein *ComponentMigration*-Ereignis verschickt wird. Falls kein Tausch möglich ist, entfällt dieser Austausch und statt des zweiten *MigrationACK* wird ein *MigrationNegative* von *MigrationEngine1* zu *MigrationEngine2* versandt. Der Ablauf endet dann mit einem *MigrationNegative* in Richtung *MigrationEngine1*. In diesem Fall tritt die gleiche Situation, wie bei der missglückten Migration ohne Komponententausch, ein.

Die Replikation ohne die nachgelagerte Migration der entstandenen Komponenten umfasst lediglich die Replikation der bisherigen Anwendungskomponente in zwei oder mehr Komponenten gleicher Arbeitsweise, wobei diese jedoch allein für einen exklusiven Raum eingehender Ereignisse zuständig sind. Damit umfasst die Replikation lediglich die Aufteilung des Ereignisraums. Über die Anzahl der nach der Replikation entstehenden Anwendungskomponenten entscheidet die Strategie zusammen mit der Statistik. Die Anzahl der bestimmten disjunkten Ereignisräume bestimmt auch die Anzahl der miteinander kommunizierenden Replikationsengines. Im folgenden Beispiel wird von zwei disjunkten Ereignisräumen und damit zwei Replikationsengines ausgegangen. Entsprechend ist das Ablaufbeispiel in Abbildung 4.9 mit *ReplicationEngine1* und *ReplicationEngine2* auf-

gebaut. Die *ReplicationEngine1* startet den Migrationsprozess durch den Auf-

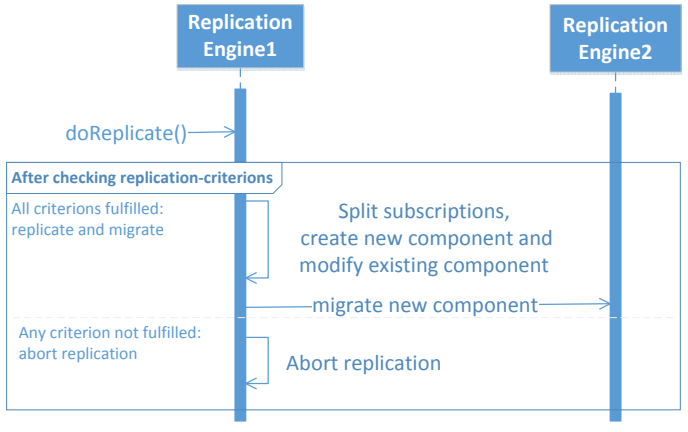


Abbildung 4.9: Kommunikationsfluss zur Replikation mittels Ereignissen

ruf aus der Statistik/Strategie heraus mit der Methode *doReplicate()*. Nach der Überprüfung, ob eine Replikation möglich ist und die Überprüfung selbst positiv ausfällt, wird zunächst die replizierte, neue Komponente aufgebaut und die entsprechend betroffenen Ereignisräume durch Aufspalten des bisherigen Ereignisraums, sprich der Anpassung der Subskriptionen, geschaffen. Die Anpassung betrifft dabei auch den Ereignisraum der ursprünglich replizierten Komponente, sodass auch deren Ereignisraum angepasst wird. Hat die eingangs erwähnte Überprüfung der möglichen Replikationsdurchführung ein negatives Ergebnis geliefert, so wird die Replikation abgebrochen. Wurde stattdessen eine Komponente, wie im Beispiel, oder wurden mehrere Komponenten im allgemeinen Fall, per Replikation neu geschaffen und der Ereignisraum der ursprünglichen Komponente angepasst, werden die neu geschaffenen und modifizierten Komponenten anschließend gezielt migriert. Im Beispiel ist eine Komponente neu entstanden und die anfangs vorhandene Komponente wurde in ihrem Ereignisraum modifiziert. Für die anschließende Migration der neu entstandenen Komponente kommuniziert *ReplicationEngine1* mit ihrem Pendant *ReplicationEngine2* auf dem Zielbroker. Anschließend füllen *ReplicationEngine1* bzw. *ReplicationEngine2* die Rollen von *MigrationEngine1* bzw. *MigrationEngine2* aus dem Migrationsprozess aus - siehe Abbildung 4.8. Über den Migrationsprozess hinausgehende Ereignisnachrichten werden nicht genutzt und sind daher auch nicht notwendig. Für jede Komponente, egal ob neu entstandene oder modifizierte, wird anschließend atomar über weitere Platzierungsanpassungen entschieden.

Komponenten und Engines - Dekomposition und Rekombination

Eine gesonderte Definition einer zusätzlichen Schnittstelle *Decomposable* und *Replicable* ist selbstverständlich möglich, im Zuge der prototypischen Implemen-

tierung standen jedoch die Funktionalitäten der Middleware im Vordergrund, sodass auf die Definition und Implementierung solcher Schnittstellen verzichtet wurde. Eine Dekomposition ist nicht nötig, da die Anwendungen bereits in atomare funktionale Komponenten aufgebrochen sind. Da sie als atomar funktionale Komponenten vorliegen und auch weiterhin verfügbar sein sollen, findet eine Rekombination als Basisoperation im Sinne des vorgestellten Konzepts aus Abschnitt 3.5.2 nicht statt. Die auf einem Broker ausgeführten Komponenten gelten bereits als rekombiniert.

Komponenten und Engines - Migration und Replikation

Die Ausführung der Basisoperationen ist ohne die Mitwirkung der Anwendungen nicht möglich, da ihnen die eigentliche Ausführung der Basisoperationen obliegt und damit auch den Anwendungsentwickler in die Pflicht nimmt. Durch die komponentenbasierte und objektorientierte Architektur sind Funktionalitäten jedoch gekapselt und sind nach außen nur über Schnittstellen erreichbar. Damit bleibt die eigentliche Implementierung einer Anwendungsmethode verborgen, was eine flexible, individuelle Ausgestaltung der Implementierungen ermöglicht.

In dieser Arbeit wurden Anwendungen in Form von Komponenten realisiert, welche als Beispiele für mögliche komplexere Anwendungskonstrukte dienen. Das Vorgehen bei der Implementierung von Komponenten (*Components*) erfolgt wie schon bei Brokern und Clients über die Implementierung von definierten Schnittstellen über eine abstrakte Komponente (*AbstractComponent*). Die Replikation und Migration werden über die zusätzlichen Schnittstellen *Replicable* und *Migratable* integriert, wobei *Migratable* eine Unterklasse von *Replicable* ist. Sowohl die Schnittstellen *Serializable* und *Component* erben von *Migratable*, da sowohl die Komponente selbst durch Bereitstellung genormter Methoden, als auch die Serialisierung und Deserialisierung von Komponenten im Zuge der Verschiebung von Komponenten betroffen sind. Eine Übersicht über die Vererbungs- und Implementierungshierarchie der Komponenten findet sich in Abbildung 4.10. Zunächst werden die Schnittstellen *Replicable* und *Migratable* definiert. Die Klasse *Replicable* erbt dabei von *Migratable*, womit auch bestimmt ist, dass alle Komponenten, die *Replicable* implementieren, auch die Schnittstelle *Migratable* implementieren. *Migratable* erbt dabei von zwei unterschiedlichen Schnittstellen, von der Schnittstelle *Component* sowie der Klasse *Serializable*. Damit ist sichergestellt, dass die Komponente sowohl die Funktionalität der bisherigen Komponente übernimmt, als auch durch die implementierte Serialisierbarkeit migrierbar ist. Die abstrakte Klasse *AbstractComponent* realisiert bereits die Schnittstellen *Component* und *Serializable*, sodass die hier genutzten Komponenten zusätzlich die Schnittstellen *Migratable* bzw. *Replicable* implementieren. Als letztes sei noch auf die Klasse *MigrationQueue* hingewiesen, welche von der abstrakten Komponente ableitet, sodass prinzipiell alle von dieser Klasse ableitenden Komponenten in eine Migrationsschlange eingeordnet und abgearbeitet werden können.

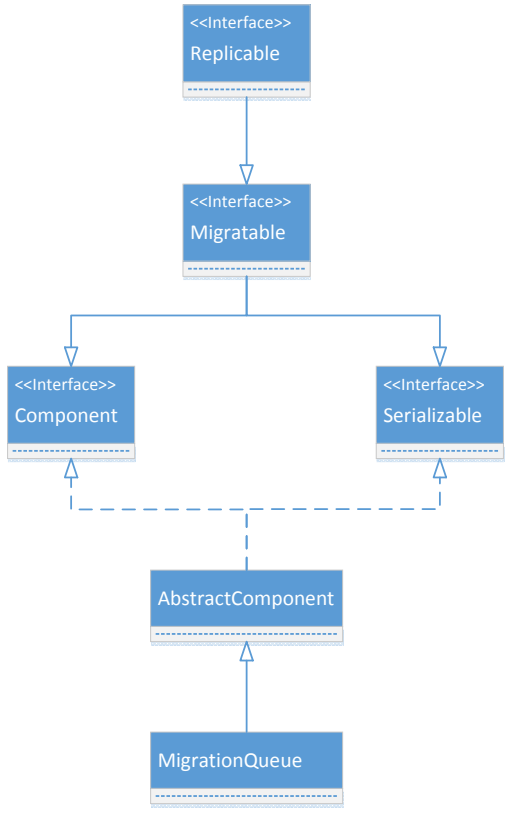


Abbildung 4.10: Klassenhierarchie Komponenten

Die Ausführung der eigentlichen Migration und Replikation obliegt den jeweiligen Engines (*BasicMigrationEngine* und *BasicReplicationEngine*), welche den Austausch der Migrations- und Replikationsereignisse (siehe Abbildungen 4.8 und 4.9) sowie letztendlich die Replikation als Aufteilung des Ereignisraums, die Serialisierung und Deserialisierung der Komponenten organisieren und durchführen. Für eine erfolgreiche Durchführung der Basisoperationen müssen die jeweiligen Engines auf allen (möglicherweise) beteiligten Brokern ausgeführt werden. Bei der Implementierung wird strikt nach dem Prinzip der Schnittstellendefinition und der anschließenden Implementierung vorgegangen, sodass stets eine einheitliche Schnittstelle gewährleistet ist und spätere Erweiterungen von Funktionalitäten und Klassen durchführbar sind. Wie in Abbildung 4.5 beschrieben ist, erben die Schnittstellen *MigrationEngine* und *ReplicationEngine* zunächst von der Schnittstellenklasse *BrokerEngine*, welche den Ausgangspunkt für alle Engines des Brokers darstellt. Die Klasse *BasicBrokerEngine* ist die grundlegende Implementierung aller Engines. Davon abgeleitet sind die Klassen *BasicMigrationEngine* und *BasicReplicationEngine*, welche zusätzlich die entsprechenden Schnittstellen für Migration und Replikation implementieren.

Statistik und Strategie - Entscheidungsfindung

Zur Entscheidungsfindung notwendig sind die Ausführung einer hinterlegten Strategie und das Sammeln von Informationen über die ein- und ausgehenden Datenströme. Die weiteren im Konzept genannten Kosten sind bei der Implementierung vernachlässigt worden, sodass lediglich die Weiterleitungskosten sowie die Reibung, sprich die Platzierungskosten, eine Rolle spielen. Sowohl die Strategie, als auch die Statistik werden im Folgenden bezüglich ihrer Umsetzung charakterisiert, wobei sowohl die Klassenstruktur, als auch die Zählweise und Auswertung der Ereignisströme betrachtet werden.

Implementierung der Klassen und des Kommunikationsflusses. Die Implementierung der Strategie erfolgt nach der bewährten zweistufigen Vorgehensweise, nämlich der Definition einer Schnittstelle und deren Implementierung. Dies erfolgt sowohl für die Migration als auch für die Replikation. Eine Übersicht dazu findet sich in Abbildung 4.11. Anhand der Abbildung ist zu erkennen, dass die konkreten Implementierungen der Schnittstellen ohne den Zwischenschritt abstrakter Klassen umgesetzt werden. Auch hier findet sich sowohl bei der Mi-

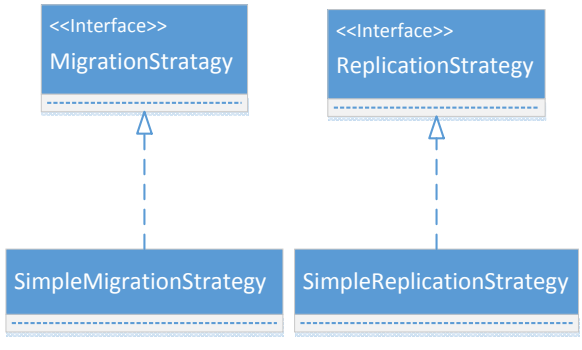


Abbildung 4.11: Klassenhierarchie Strategien

grationsstrategie, als auch bei der Replikationsstrategie die Bezeichnung „Simple“, was wiederum nicht gleichbedeutend für einfach=simpel ist, sondern die erste und derzeitige Implementierung der gewählten Strategie bezeichnet. Die Strategie entscheidet, ab wann, wie viele und in welcher Zeit Nachrichten auflaufen und eine Platzierungsanpassung auslösen. Dazu kommunizieren Strategie, Statistik und die entsprechenden Engines miteinander. Wie das geschieht, verdeutlicht Abbildung 4.12. Anhand dieser Abbildung sind der Arbeitsablauf und die Kommunikation zwischen der Strategie, den ausgeführten Komponenten und der Statistik-Engine am Beispiel der Migration dargestellt. Für die Replikation arbeitet das Verfahren in der gleichen Art und Weise, da eine zusätzliche Engine und Strategiekomponente für die Replikation durch den Broker ausgeführt werden. Ausgangspunkt ist der Broker b_1, der eine Komponente c_1 ausführt.

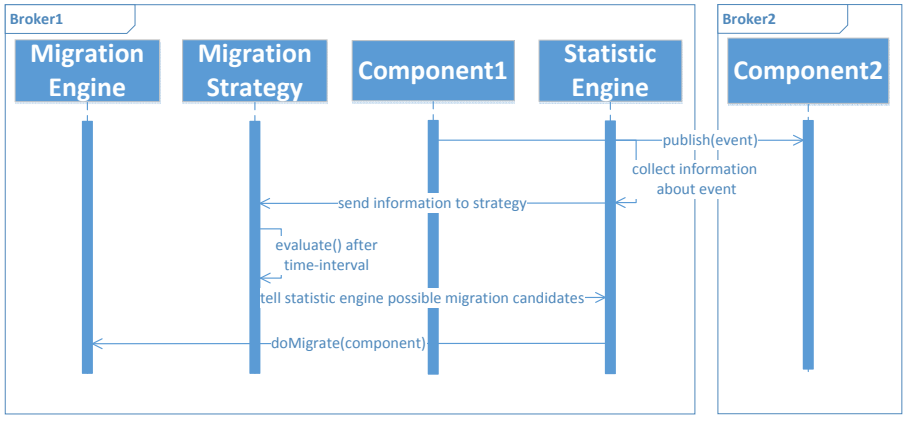


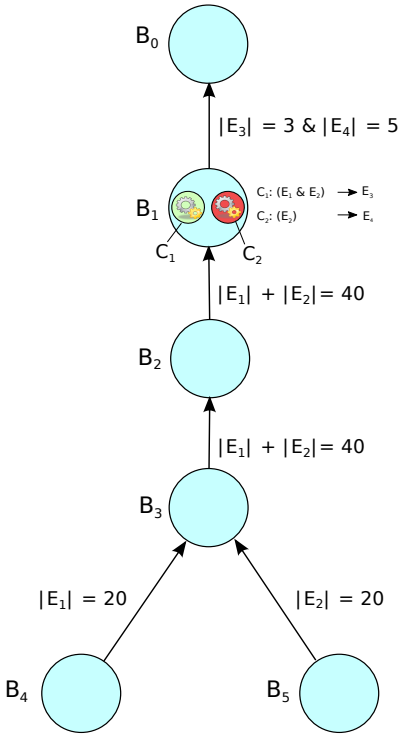
Abbildung 4.12: Kommunikation zwischen Strategie und Statistik mittels Ereignissen bei der Migration

Diese versendet ein Ereignis an eine Komponente c_2 auf Broker b_2. Dieses ausgehende Ereignis wird durch die StatisticEngine hinsichtlich Ereignistyp, Ziel und Herkunft ausgewertet. Diese Informationen werden an die *MigrationStrategy* weitergeleitet, welche nach Ablauf einer definierten Zeitspanne diese Informationen auswertet. Die Regeln dazu werden im Verlauf dieses Abschnitts genauer erläutert. Stellt die Strategie fest, dass ein Migrationskandidat vorhanden ist, teilt sie dies der *StatisticEngine* mit, welche schließlich der *MigrationEngine* die Anweisung erteilt, die ermittelte Komponente zu migrieren. An dieser Stelle sind noch die Hinweise wichtig, dass in den Broker die Engines eingefügt (geplugt) und die Komponenten vom Broker verwaltet werden. Jede Engine wird auf dem Broker genau einmal ausgeführt, unabhängig von der Anzahl der Anwendungskomponenten. Dies gilt für alle Engines eines Typs, also für Migration, Replikation und Statistik. Die Implementierung der Statistik als Engine und deren Einordnung in das Klassengefüge zeigt Abbildung 4.5. Dabei erbt die Schnittstellenklasse *StatisticEngine* von der Schnittstellenklasse *BrokerEngine*, sodass die *StatisticEngine* ebenfalls den Anforderungen einer allgemeinen Engine des Brokers entspricht. Die Implementierung der konkreten *BasicStatisticEngine* erbt von der *BasicBrokerEngine*, welche als Ausgangspunkt auch aller anderen Engines dient, und implementiert zusätzlich die Schnittstellen der Klasse *StatisticEngine*.

Ermittlung von Kräften - Strategie, Statistik und Zählweise. Für die Ermittlung von Kräften wird zunächst die Statistik über ein- und ausgehende Ereignisse geführt, auf deren Grundlage Platzierungsentscheidungen, also die Ausführung einer Migration und Replikation, getroffen werden.

Migration - Strategie und Statistik. Die grundsätzliche Idee der Migrationsstrategie ist die Ermittlung der Richtung, bei der eine Verschiebung der Anwendungskomponente die größten Einsparungen mit sich bringt. Dazu bedient sich die Strategie der internen Nachrichten der Statistik, welche die ein- und ausgehenden Ereignisse analysiert.

Für die Umsetzung der Strategie wurde ein Algorithmus implementiert, der möglichst einfach, jedes ein- und ausgehende Ereignis betrachtet. Zum besseren Verständnis der Umsetzung seien zunächst im Vorfeld der Vorstellung der Strategieumsetzung einige Anmerkungen gestattet. Ausgangspunkt ist eine Komponente c, welche die Schnittstelle *Migratable* implementiert und damit innerhalb des Netzwerks verschiebbar ist. Die Komponente c wird von Broker b_1 ausgeführt, welcher auch die Ausführung der *StatisticEngine* und der Strategie übernimmt. Broker b_i bezeichnet dagegen den jeweiligen benachbarten Broker und damit das mögliche Migrationsziel. Je Broker wird je eine Strategie und eine Statistikinstanz für die Migration geführt und damit jedes ein- und ausgehende Ereignis brokerweit analysiert und daraufhin über die Migration von Komponenten entschieden. Die als Engine implementierte Statistik besitzt eine Senke, welche die ein- und ausgehenden Ereignisse abgreift und für jedes Ereignis ein separates Datenobjekt erstellt, welches wiederum an die eigentliche Statistikengine weitergeleitet wird sowie Informationen über den Ereignistyp, die Ereignisquelle und alle Ereignissenken enthält. Mit Hilfe dieses Datenobjekts wird eine Menge S generiert, welche die genannten Informationen (Ereignistyp, Ereignisquelle und Ereignissenke(n)) aus Sicht von c an b_1 enthält. Ereignisquelle und Ereignissenken können sowohl Broker, als auch Komponenten des Brokers b_1 sein. Bei eingehenden Ereignissen wird als Ereignisquelle der Nachbarbroker genannt, von dem das Ereignis zu b_1 geleitet wurde, bei den Ereignissenken werden die (Anwendungs-) Komponenten aufgeführt, welche von b_1 ausgeführt werden. Bei ausgehenden Ereignissen ist dies umgekehrt. So wird als Ereignisquelle die Komponente aufgeführt, die von b_1 ausgeführt wird und die den Ereignisversand initiiert hat, während unter den Ereignissenken alle die Nachbarbroker von b_1 aufgelistet sind, an die das Ereignis weiterleitet wird. Ereignisse, die lediglich von einem Nachbarbroker zu anderen Nachbarbrokern weitergeleitet und nicht durch eine lokal ausgeführte Komponente verarbeitet werden, werden von der Statistik nicht als eigenes Objekt erfasst. Da diese Ereignisse nicht durch Komponenten verarbeitet werden, können die nicht vorhandenen Komponenten auch nicht in ihrer Platzierung angepasst werden. Diese Ereignisse sind für die Ressourceneffizienz nicht von Bedeutung. Neben der Einzelauswertung der ein- und ausgehenden Ereignisse inklusive der Definition der Menge S, existiert je Broker eine globale Matrix D, in der alle relevanten Ereignisse bzw. deren erzeugte Datenobjekte erfasst werden und der Einfluss einer Platzierungsanpassung der lokal ausgeführten Komponenten bestimmt wird. Die Matrix enthält alle eingetragenen potentiellen Einsparungen (positive Werte) und Zusatzkosten (negative Werte), die bei der Migration einer von Broker b_1 lokal ausgeführten Komponente (Zeile(n) von D) in Richtung eines Ziels (Nachbarbroker von b_1 als Spalte(n) von D) entstehen würden. Die linke Teilabbildung 4.13(a) der Abbildung 4.13 zeigt



(a) Graphische Darstellung der Ausgangssituation im Netzwerk

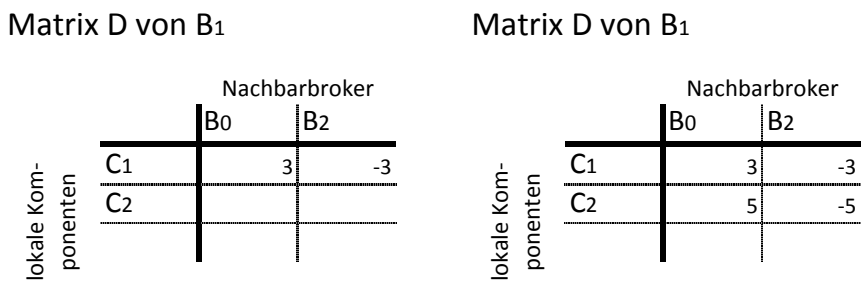
Matrix D von B1

lokale Komponenten	Nachbarbroker B0	Nachbarbroker B2
C1		
C2		

(b) Auswertungsmatrix der Statistikengine von Broker B_1

Abbildung 4.13: Ausgangssituation einer Auswertung vor der Migration

ein Brokernetzwerk mit sechs Brokern (B_0 bis B_5). Auf Broker B_1 werden mit C_1 und C_2 zwei Komponenten ausgeführt. Die Komponente C_1 subskribiert sich für die Ereignisse E_1 und E_2 und erzeugt Ereignis E_3, während sich die Komponente C_2 nur für die Ereignisse E_2 subskribiert und Ereignisse vom Typ E_4 produziert. Sowohl E_3 als auch E_4 werden aufgrund vorliegender Subskriptionen in Richtung des Brokers B_0 weitergeleitet. Die Matrix D von Broker B_1 vor Beginn der Zählung der einzelnen Ereignisse zeigt die rechte Teilabbildung 4.13(b) der Abbildung 4.13. Wie im Pseudoquelltext der Abbildung 4.16 gezeigt wird, erfolgt anschließend für jedes ein- und ausgehende Ereignis eine Auswertung des daraus erzeugten Statistikobjekts. Anhand des in Abbildung 4.13 gezeigten Beispiels wird der Ablauf erläutert. Dabei werden alle benachbarten Broker und migrierbaren Komponenten betrachtet und damit zunächst Broker B_0 und Komponente C_1. Zum ausgehenden Ereignis E_3 wurde eine Menge S erstellt, welche die Komponente C_1 als Quelle und B_0 als globales Ziel enthält. Um auszuschließen, dass kein anderer Broker oder keine Komponente an E_3 interessiert sind, werden B_0 und C_1 aus der Menge S entfernt. Bleibt eine leere Menge übrig, so ist dies gewährleistet. Der Betrag an der Stelle B_0, C_1 wird um eins erhöht, da



(a) Auswertungsmatrix der Statistikengine von Broker B_1 nach Auswertung aller E_3

(b) Auswertungsmatrix der Statistikengine von Broker B_1 nach Auswertung aller E_3 und E_4

Abbildung 4.14: Auswertungsmatrix der Statistikengine von Broker B_1 nach Auswertung von E_3 und E_4

eine Verschiebung von C_1 in Richtung von B_0 eine Einheit sparen würde. Da je Zeiteinheit drei Einheiten erzeugt und weitergeleitet werden, steht zunächst an der Stelle B_0, C_1 ein Betrag von drei. Da noch ein zweiter Broker B_2 ein Nachbarbroker von B_1 ist, wird auch dieser bei der Bewertung des Ereignisobjekts betrachtet. Der Broker B_1 befindet sich jedoch nicht in der Menge S, sodass der Betrag des Feldes B_0, C_1 um insgesamt drei Einheiten dekrementiert wird, da eine Verschiebung von C_1 nach B_2 eben diese zusätzlichen Kosten nach sich ziehen würde, da E_3 an Broker B_0 weitergeleitet wird. Da C_2 in der Menge S nicht vorkommt, werden hier keine weiteren Auswertungsschritte unternommen. Den Stand der Auswertungsmatrix nach Auswertung von E_3 zeigt Abbildung 4.14(a). Das Vorgehen bei den Ereignissen des Typs E_4 ist ähnlich dem Vorgehen bei den Ereignissen E_3, nur dass sich die Beträge unterscheiden (C_2 erzeugt vier Ereignisse vom Typ E_4 je Zeiteinheit) und die Komponente C_1 nicht betroffen ist. Die Menge S für die Ereignisse E_4 enthält den Broker B_0 als globales Ziel und die Komponente C_2 als Ereignisquelle. Ein lokales Ereignisziel wurde auch in diesem Fall nicht in die Menge S aufgenommen. Nach Auswertung ergibt sich die in Abbildung 4.14(b) gezeigte Matrix. Ähnlich wie bei den Ereignissen vom Typ E_3 und E_4 verläuft die Auswertung der Ereignisse vom Typ E_1 aus Richtung des Brokers B_2. Die Menge S für diesen Ereignistyp enthält den Broker B_2 (Ereignisquelle) und die Komponente C_1 (lokales Ziel). Ein globales Ziel ist nicht vorhanden und findet sich auch nicht in der Menge S wieder. Es finden ein Durchlauf über die Komponenten statt, da nur C_1 in der Menge vertreten ist, und dazu zwei Durchläufe über die Broker B_0 und B_1. Eine Verschiebung von C_1 in Richtung des Brokers B_0 würde die Kosten je Ereignis um eins erhöhen. Der Broker B_0 befindet sich nicht in S, womit je Ereignis der Eintrag der Matrix D an der Stelle B_0, C_1 jeweils um eins dekrementiert wird. Für die Betrachtung des Brokers B_2 ergibt sich die Situation, dass B_2 in der Menge S enthalten ist. Nach Abzug von Broker B_2 und C_1 bleibt eine leere Teilmenge übrig, womit

Matrix D von B_1

lokale Komponenten	Nachbarbroker B_0	Nachbarbroker B_2
C_1	-17	17
C_2	5	-5

(a) Auswertungsmatrix der Statistikengine von Broker B_1 nach Auswertung aller E_3, E_4 und E_1

Matrix D von B_1

lokale Komponenten	Nachbarbroker B_0	Nachbarbroker B_2
C_1	-37	37
C_2	-15	15

(b) Auswertungsmatrix der Statistikengine von Broker B_1 nach Auswertung aller E_3, E_4, E_1, und E_2

Abbildung 4.15: Auswertungsmatrix der Statistikengine von Broker B_1 nach Auswertung von E_3, E_4, E_1 und E_2

auch durch die Mengenbetrachtung deutlich wird, dass kein anderer Broker und keine weitere Komponente an Ereignissen des Typs E_1 interessiert sind. Folglich wird D an der Stelle B_2, C_1 je Ereignis vom Typ E_1 um eins inkrementiert. Die Situation nach Auswertung aller Ereignisse E_1 zeigt Abbildung 4.15(a).

Bei Ereignissen des Typs E_2 verläuft die statistische Betrachtung anders als zuvor, da sowohl die Komponente C_1, als auch die Komponente C_2 an diesen Ereignissen interessiert sind. Die Menge S für diesen Ereignistyp besteht aus dem Broker B_2 (Ereignisquelle) sowie den Komponenten C_1 und C_2 (lokale Ziele). Ein Eintrag eines globalen Ziels existiert nicht. Da insgesamt zwei Komponenten an E_2 interessiert sind, werden je Komponente beide Nachbarbroker durchlaufen. Insgesamt werden für jedes Ereignis vom Typ E_2 damit vier Durchläufe in der Statistik durchgeführt. Für jede Komponente (C_1 und C_2) werden beide Broker B_0 und B_2 betrachtet. Broker B_0 ist nicht in S enthalten, sodass in der Matrix D der an der Stelle B_0, C_1 angegebene Betrag je Ereignis E_2 um eins dekrementiert wird. Beim Durchlauf der Schleife für den Nachbarbroker B_2 wird nach der Überprüfung von S festgestellt, dass B_2 in der Menge S enthalten ist und die Teilmenge S_1 nicht leer ist. Das verbliebene Element der Menge ist mit C_2 kein Broker. Wäre in S mindestens ein Broker enthalten, wenn im Beispiel B_0 ebenfalls Ereignisse vom Typ E_2 an Broker B_0 weitergeleitet werden würden, würde das Ereignis in der Statistik nicht gezählt werden. Ebenfalls nicht in der Statistik erfasst wird es, wenn eine weitere, nicht migrierbare Komponente an E_2 interessiert wäre. Im konkreten Fall ist mit C_2 jedoch nur eine weitere, migrierbare Komponente in S enthalten, sodass eine Migration (aller Komponenten) sinnvoll sein kann. Für jede Instanz des Ereignistyps E_2 wird die Matrix D an der Stelle B_2, C_1 um eins inkrementiert. Das gleiche gezeigte Vorgehen wird auch für die Komponente C_2 angewandt. Im Ergebnis der statistischen Auswertung wird an der Stelle B_2, C_2 der Matrix D für jede Ereignisinstanz von E_2 der dortige Wert um eins inkrementiert, während an der Stelle B_0, C_2 der Wert je Ereignis um

eins dekrementiert wird. Das Ergebnis der Auswertung zeigt Abbildung 4.15(b). Die Zählweise setzt voraus, dass die Komponenten wenn nicht zeitgleich, doch zumindest zeitnah in die gleiche Richtung migrieren. Führt dies eine Komponente jedoch nicht aus, so ist zwar die zu erzielende Einsparung in der Summe am Ende geringer, jedoch entstehen keine zusätzlichen Kosten. Wäre im Beispiel die Komponente C_2 nicht migrierbar, oder E_2 würde ohnehin an B_0 weitergeleitet, ergäbe sich eine Matrix wie in Abbildung 4.15(a).

Die gesammelten, in der Distanzmatrix D zusammengefassten, Informationen dienen anschließend der Ermittlung der Migrationskandidaten. Da die Informationen nach Brokern und Komponenten sortiert vorliegen, wird für jede Komponente entschieden, ob eine Migration sinnvoll ist und wenn ja, in welche Richtung die Komponente verschoben werden soll. Dazu wird die Matrix D zeilenweise und damit komponentenweise ausgewertet. Je Zeile wird der betragsmäßig größte, positive Wert ermittelt, ein Migrationskandidat ist gefunden. Ist der ermittelte Wert größer als die zu erwartenden Platzierungskosten (Reibung aus dem Kräftemodell), so werden die Komponente und das Migrationsziel an die Migrationsengine weitergeleitet. Im Beispiel (Abbildung 4.15(b)) sind bei geschätzten Platzierungskosten von sechs Geldeinheiten sowohl C_1 als auch C_2 Migrationskandidaten in Richtung B_2. Die Evaluation erfolgt regelmäßig in bestimmten Abständen. Die bisherigen Werte werden allerdings nicht vergessen, sondern bei jedem Auswertungsschritt mit dem Vergessensfaktor α multipliziert.

Replikation - Strategie und Statistik. Die grundsätzlichen Abläufe bei der Replikation entsprechen denen der Migration. So werden zunächst die ein- und ausgehenden Ereignisse für jeden Broker durch die *BasicStatisticEngine* erfasst und an die *SimpleReplicationStrategy* weitergeleitet. Dort wird wiederum eine Matrix M geführt, welche die eingehenden Ereignisströme nach ihrer Herkunft klassifiziert und zählt. Konzeptionell ist bei dem in dieser Arbeit vorgestellten Ansatz keine Replikation einer Anwendungskomponente in Richtung der Ereignissenken möglich, da für eine Aufteilung des Ereignisraums in diese Richtungen die jeweiligen Subskriptionen der Subskribenten angepasst werden müssten. Dies wird bei der Aufteilung des Ereignisraums aus Richtung der Ereignisströme zwar auch getan, allerdings erfolgt dies an der Stelle des Brokers, zu dem die Ereignisse hinfließen. Änderungen des Ereignisraums in Richtung der Ereignissenken zu vollziehen heißt, dass die Subskribenten entsprechend modifizierte Subskriptionen abgeben. Bei einer Anpassung in Richtung der Ereignissenken bedeutet dies eine direkte Rückkopplung. Dies verstößt allerdings gegen das Prinzip der losen Kopplung in Publish/Subscribe. Aus dieser Restriktion heraus werden bei der Replikation nur die eingehenden Ereignisströme betrachtet. Die Matrix M ist nach den Komponenten des Brokers (Zeilen) und Nachbarbrokern (Spalten) gegliedert. Die Matrix gilt auch hier global für den gesamten Broker. Die Spaltenköpfe der Matrix M repräsentieren gleichzeitig die möglichen Ziele der Replikation der Komponenten, dies sind die Nachbarbroker. Neben diesen können die Komponenten allerdings auch Ereignisse von lokal ausgeführten, benachbarten Komponenten erhalten. Da aus den lokalen, benachbarten Komponenten keine

```
// c is a local migratable component
// b_i is a neighboring broker
// S is the set of broker and components containing the event source and
//    all event destinations
// D is the delta matrix with potential savings (positive)/
//    additional cost (negative)

 for each c.isMigratable() part of S do
   for Broker b.isNeighbor() do
     // b is not in set ⇒ migration to b causes an additional hop
     if b not in S then
       D[b][c]--
       continue
     endif
       // determine reduced set without currently considered
       //    component and broker
       S_1 = S \ {b,c}

     // Migration of component c to current neighbor broker b_i saves one
     //    hop and no other (neighboring) broker is interested in the event
     if S_1.isEmpty()
       D[b][c]++
       continue
     endif
     // set contains at least a second broker ⇒ migration has no effect
     if S_1 contains a broker then
       continue
     endif
     // if S_1 contains only components now
     // set contains at least one non-migratable component
     //    ⇒ migration has no effect
     if S_1.getAllComponents().haveAtLeastOneNonMigratable()
       continue
     endif
     if S_1.getAllComponents().allMigratable()
     // finally S_1 contains only migratable components
     //    ⇒ migration of all components would save a hop
           D[b][c]++
   done
 done
```

Abbildung 4.16: Umsetzung der Migrationsstrategie

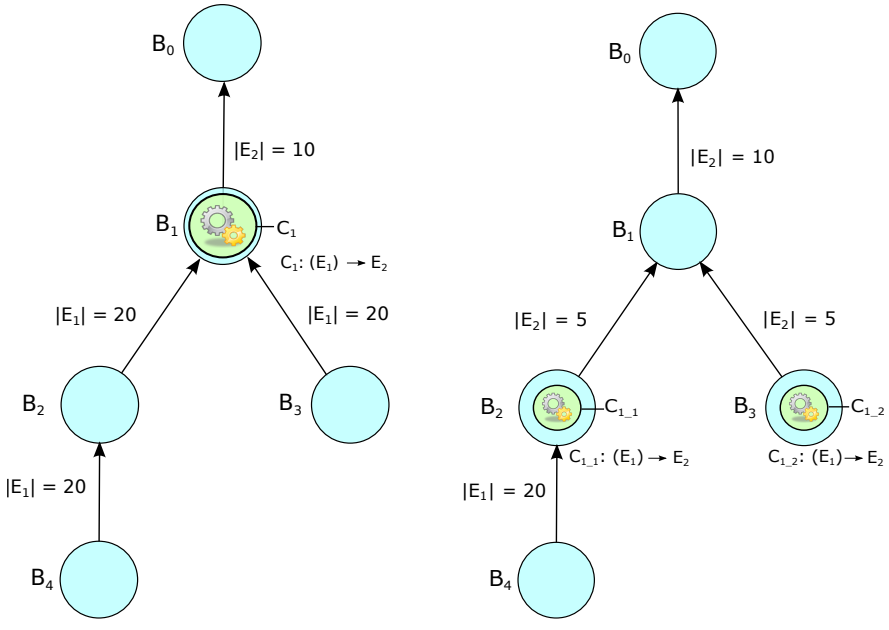
Replikationsziele erwachsen können, werden die bei der Komponente von anderen lokalen Komponenten eingehenden Ereignisse in einer gemeinsamen Spalte unter der Bezeichnung *local* zusammengefasst. Wie in der Konzeptbeschreibung erläutert wurde, erfolgt aufgrund der Aufteilung des Ereignisraums in Richtung der eingehenden Ereignisse eine Replikation mit der nachgestellten Migration le-

diglich in Richtung der Ereignisquelle. Zur Unterscheidung, ob ein Ereignis ein- oder ausgehend ist, wird eine zusätzliche Methode *classify()* genutzt, welche innerhalb der Komponente implementiert ist und durch die Statistik aufgerufen wird. Die Methode liefert die Quelle des (Name-Wert-) Ereignisses. Durch Vergleich des Rückgabewertes und der Komponente wird festgestellt, ob es sich um ein ein- oder ausgehendes Ereignis handelt, da bei eingehenden Ereignissen der von der Methode *classify()* zurückgelieferte Wert der Ereignisquelle nicht dem des die Komponente ausführenden Brokers bzw. der Komponente entspricht.

Als Beispiel wird im Folgenden das in Abbildung 4.17(a) gezeigte Brokernetzwerk betrachtet. Der Broker B_1 führt die Anwendungskomponente C_1 aus und hält Verbindungen zu den Brokern B_0, B_2 und B_3. Zwischen dem Broker B_2 und dem Broker B_4 besteht zudem eine weitere Verbindung. Die Komponente C_1 subskribiert sich für Ereignisse des Typs E_1, welche sowohl aus Richtung des Brokers B_4 via B_2, als auch aus Richtung des Brokers B_3 zu B_0 weitergeleitet werden. Komponente C_1 verarbeitet die Ereignisse E_1 vollständig und produziert Ereignisse des Typs E_2. Es besteht eine Subskription für E_2 aus Richtung des Brokers B_0, sodass alle E_2 in Richtung B_0 weitergeleitet werden. Die Intensitäten der jeweiligen Ereignisströme sind ebenfalls der Abbildung 4.17(a) zu entnehmen. Die Anzahl der Spalten Matrix M für den Broker B_1 bestimmt sich nach der Anzahl der Nachbarbroker (B_0, B_2 und B_3) und der Spalte *local*. Da auf B_1 lediglich die Komponente C_1 ausgeführt wird, enthält M nur eine Zeile. Abbildung 4.18(a) zeigt den Aufbau der Matrix M.

Erreicht ein Ereignis E_1 die Komponente C_1 am Broker B_1 vom benachbarten Broker B_2, ergibt die Methode *classify()*, dass die Ereignisquelle von E_1 in diesem Fall B_2 ist und es sich damit um ein eingehendes Ereignis handelt. An der Stelle B_2, C_1 der Matrix M wird der dort hinterlegte Wert um eins inkrementiert. Dies geschieht für jedes eingehende Ereignis E_1 aus Richtung von B_2. Das gleiche Vorgehen wird bei den Ereignissen E_1 aus Richtung von B_3 angewendet, sodass M an der Stelle B_3, C_1 ebenfalls für jedes E_1 um eins inkrementiert wird. Bei den von C_1 ausgehenden Ereignissen des Typs E_2 ergibt die Methode *classify()*, dass C_1 die Ereignisquelle ist und aufgrund der Einschränkung, dass die Replikation nur in Richtung der Ereignisquellen durchgeführt wird, werden die Einträge in M nicht modifiziert. Sollte C_1 von anderen Komponenten auf B_1 Ereignisse beziehen, würde je eingetroffenem Ereignis der Wert an der Stelle *local*, C_1 von M um eins inkrementiert.

Wie auch bei der Auswertung der Migrationsstrategie (*SimpleMigrationStrategy*) erfolgt die Auswertung der aufgestellten Tabelle regelmäßig nach Ablauf einer bestimmten, festgelegten Zeit. Ebenso festgelegt ist der Schwellwert, der die „Reibung“ bzw. die Platzierungskosten repräsentiert und von den möglichen Einsparungen zusätzlich überwunden werden muss. Dieser Ablauf unterscheidet sich nicht von dem der Migrationsstrategie. Die Auswertung erfolgt zeilenweise und je Zelle. Zunächst werden grundsätzlich alle Zahlenwerte der Zeile, mit Ausnahme der betrachteten Zelle, aufsummiert. Im zweiten Schritt werden dem Wert der betrachteten Zelle die aufsummierten Werte der anderen Zelle derselben

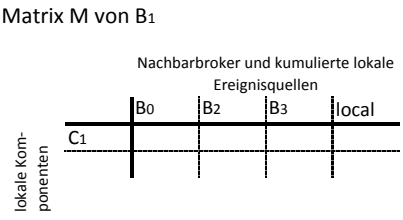


(a) Brokernetzwerk mit Komponente C_1 auf Broker B_1 vor der Replikation nach B_2 und B_3

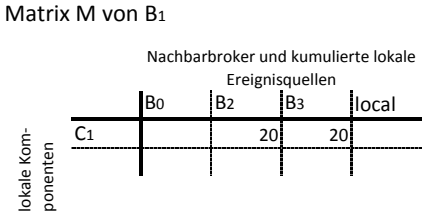
(b) Brokernetzwerk mit Komponente C_1 nach der Replikation von Broker B_1 nach B_2 und B_3

Abbildung 4.17: Brokernetzwerk mit Komponente C_1 vor und nach deren Replikation von B_1 zu den Brokern B_2 und B_3

Zeile abgezogen. Bleibt danach ein positiver Wert und ist die zu erwartende Einsparung größer als der durch die Platzierungskosten definierte Schwellwert, wird der sich aus der Zelle ergebene Spaltenkopf als Migrationsziel der betrachteten Komponente ausgewählt. Allerdings wird dieses Vorgehen bei der hier vorgestellten Implementierung nicht angewandt. Stattdessen wird sichergestellt, dass die Migration vor einer Replikation ausgeführt wird. Wie dies konkret implementiert wurde, wird im folgenden Teilabschnitt zum gegenseitigen Ausschluss zeitgleicher Migration und Replikation beschrieben. Zur Beurteilung, ob ein Replikationskandidat vorliegt, genügt es, wenn der Wert der Zelle, abzüglich der Summe aller anderen Werte der Zeile, null ergibt. Ist dies der Fall, ergibt sich eine Zugwirkung für die Komponente in Richtung des im Spaltenkopfs genannten Brokers nach einer Replikation. Ist der Wert der Zelle größer als der, der durch die Platzierungskosten bestimmten Gegenkraft (Reibung), so ist die Komponente ein Replikationskandidat mit anschließender Migration in Richtung des Brokers im Spaltenkopf. Im Beispiel von Abbildung 4.17 und der dazugehörigen Statistik in Abbildung 4.18(b) sind sowohl B_2 als auch B_3 die neuen Ausführungsorte von C_1 nach der Replikation und Migration, wenn die Platzierungskosten nicht mehr als 20 Einheiten betragen. Da die replizierten Komponenten von C_1 für



(a) Auswertungsmatrix der Statistikengine von Broker B_1 für die Replikation vor Auswertung der Ereignisströme



(b) Auswertungsmatrix der Replikationsstatistikengine von Broker B_1 für die Replikation nach Auswertung der Ereignisströme E_1

Abbildung 4.18: Auswertungsmatrix der Statistikengine von Broker B_1 vor und nach der Auswertung der Ereignisströme E_1

einen eingeschränkten Ereignisraum zuständig sind, werden sie fortan als C_{1_1} und C_{1_2} bezeichnet. Den Zustand des Netzwerks nach der Replikation von C_1 zeigt Abbildung 4.17(b).

Nach dem Durchlauf der Statistiktabelle werden die ausgelesenen Werte nicht gelöscht, sondern mit dem Vergessensfaktor α multipliziert, in der Matrix belassen und diese mit der Zählung der aktuellen Ereignisse fortgeführt. Die Multiplikation der Werte in M mit α ist unabhängig davon, ob Replikationskandidaten gefunden werden konnten oder nicht. Die historischen Summen der ein- und ausgehenden Ereignisse verlieren schrittweise ihren Einfluss auf die anstehenden Entscheidungen. Falls ein länger währender oder eher kurzlebiger Einfluss gewünscht ist, lässt sich das durch die Variation von α bewerkstelligen.

Gegenseitiger Ausschluss von zeitgleicher Migration und Replikation. Wie in den vorherigen Teilabschnitten beschrieben wurde, nutzen sowohl Migration als auch Replikation dieselben, durch die *StatisticEngine* gesammelten Informationen, welche allerdings an verschiedenen Stellen ausgewertet werden. Trotzdem könnte es dazu kommen, dass die *SimpleMigrationStrategy* und gleichzeitig die *SimpleReplicationStrategy* die Position einer Komponente verändern wollen.

Welche Konkurrenzsituationen sind denkbar? Das wären zunächst Zugriffe von *SimpleMigrationStrategy* und *SimpleReplicationStrategy* auf dieselbe Komponente, sodass diese sich also gleichzeitig replizieren und in Gänze in eine Richtung migrieren soll. Während des Entwurfs und der Implementierung wurde zwischen zwei unterschiedlichen Strategien abgewogen - Soll die Migration oder die Replikation vordergründig ausgeführt werden? Die Replikation geht mit einem größeren Aufwand einher als die Migration, da insgesamt zwei Migrationen notwendig sind, dies ist auch mit höherem Aufwand zur Anpassung der Ankündigungen und Subskriptionen verbunden und zusätzlich muss die Aufteilung des Ereignisraums (eingehende Ereignisse) durchgeführt werden. Daher

ist die Idee, Komponenten soweit wie möglich in Gänze zu migrieren und erst dann zu replizieren, sobald keine weitere Migrationsmöglichkeit mehr vorhanden ist. Dies sind bspw. Stellen, an denen zwei Eingangsströme zusammenfließen. Der Vorrang der Migration lässt sich durch zwei Möglichkeiten regeln. Zum einen lässt sie sich durch eine vorgezogene Auswertung der Strategie einleiten, indem die Migrationsstrategie vor der Replikationsstrategie ausgewertet wird. Wird während des Auswertungszeitraums der Migrationsstrategie der geforderte Schwellwert nicht erreicht, wird keine Platzierungsänderung ausgeführt und die bisher in der Statistik gesammelten Daten zu den Ereignisströmen werden dank des Vergessensfaktors α abgewertet. Möglicherweise findet eine Migration entweder verspätet oder gar nicht statt. Der Schwellwert ist so zu wählen, dass eine Platzierungsanpassung vorteilhaft ist und damit mindestens die Kosten der Platzierungsänderung amortisiert werden. Daher ist eine Verkürzung der Auswertungsdauer mit der gleichzeitigen Verringerung des Schwellwertes wenig sinnvoll und eine Verkürzung der Auswertungsdauer ohne Anpassung des Schwellwertes ebenso. Stattdessen ist zum anderen allein durch die Anpassung des Schwellwertes eine Steuerung von Migration und Replikation zu erreichen. Da die Replikation mit einem höheren Aufwand verbunden ist, liegt der Schwellwert hier bereits höher, sodass über eine Replikation als sinnvolle Platzierungsänderung erst bei höheren Werten entschieden werden kann. Konnte aufgrund des geringeren Schwellwertes der Migration diese zuvor ausgeführt werden, beginnt die Statistik neu zu zählen, sodass Replikationen erst dann stattfinden, falls absehbar keine Migration möglich ist. Falls es wider Erwarten doch zu einer Überschneidung beider Aktionen kommen sollte, ist zusätzlich ein Mechanismus implementiert, welcher die für eine Migration bzw. Replikation vorgemerkte Komponente keinen weiteren Strategieimplementierungen zugänglich macht. Es gilt das Prinzip *First Come - First Serve*.

Wie im Konzept gesehen, ist vor einer Migration oder Replikation einer Komponente von einem Broker zu einem anderen die Ausführungszusicherung des empfangenden Brokers erforderlich. Versendet ein Broker also zwei Anfragenden gleichzeitig Bestätigungen über dieselben angefragten Ressourcen, kann es bei der tatsächlichen Ausführung von Platzierungsänderungen zu Konflikten kommen. Ein optimistischer Ansatz würde an dieser Stelle eine Überlast tolerieren. Dies in der Hoffnung, dass bei späteren Platzierungsanpassungen die Überlast aufgelöst wird. Bei begrenzten Ressourcen der Broker ist jedoch damit zu rechnen, dass sich bei einzelnen Brokern an Schnittstellen längere Überlastsituationen einstellen. Bei der hier implementierten Variante wird zur Vermeidung solcher Situationen konservativ vorgegangen und Überlastsituationen werden soweit wie möglich umgangen. So kommt auch hier das *First Come - First Serve*-Prinzip zum Tragen. Ein Broker reserviert und bestätigt dem ersten Anfragenden die angeforderten Ressourcen. Wenn für eine weitere Komponentenverschiebung keine Ressourcen vorhanden sind, wird die Anfrage abgelehnt bzw. das Komponententauschprotokoll angestoßen.

4.5 Diskussion

Die Implementierung des in Abschnitt 3 vorgestellten Konzepts zur automatischen Platzierung von Anwendungen erfolgt prototypisch innerhalb der bereits vorhandenen Middleware REBECA in der Programmiersprache Java. REBECA setzt das Kommunikationsparadigma Publish/Subscribe um und ermöglicht als Forschungsprototyp die Integration unterschiedlicher Aspekte wie Fehlertoleranz, Sichtbarkeitsbereiche, den Einsatz von Ankündigungen und unterschiedlichster Routingalgorithmen. Mit diesem Ansatz bietet REBECA die Möglichkeit, ereignisbasierte Kommunikation für unterschiedlichste Anforderungen umzusetzen. Unterschiedlichen Anforderungen zu begegnen, heißt aber auch, dass sich diese mitunter sogar gegenseitig widersprechen können und ein System, das allen möglichen und damit auch zukünftigen Anforderungen genügt, nicht umsetzbar ist. REBECA greift auf das Grundprinzip der modularen Gestaltung zurück, sodass die Broker als Basis der Funktionalität nur über eine Rumpffunktionalität verfügen, der bedarfsweise die benötigten Funktionalitäten hinzugefügt werden. Die Funktionalitäten selbst sind in Module gekapselt, welche auch zur Laufzeit dem Broker hinzugefügt und/oder ausgetauscht werden können. Dabei lassen sich die in Module gegossenen Funktionalitäten auch kombinieren. Dieses Verfahren wird auch als Feature Composition beschrieben. Auf dieser Grundlage wurde das in dieser Arbeit vorgestellte Konzept in einzelne Funktionalitäten unterteilt und diese als Plugins implementiert. Dieses Vorgehen hat sich bei der Implementierung von Funktionalitäten in REBECA bewährt und wird daher auch hier angewendet. Somit ist die Anwendungsplatzierung weiterführend zu bestehenden Funktionalitäten nutzbar. Dabei wird zusätzlich zwischen Engines und Senken unterschieden, wobei die Senken in den Ereignisverarbeitungsfluss integriert sind und die Engines (in Zusammenarbeit mit ihren spezifischen Senken) die eigentliche Funktionalität implementieren. Die jeweiligen Engines und Senken sind nach ihrer Funktionalität eingeteilt.

Die initiale Platzierung wird durch vorhandene Mittel der Publish/Subscribe-Infrastruktur (Ankündigungen, Subskriptionen und Ereignispublikationen) umgesetzt. Zur besseren Abgrenzung zu den Publish/Subscribe-Primitiven, welche die eigentliche Nutzlast im Netzwerk befördern, wurden für die Publish/Subscribe-Primitive der initialen Platzierung spezifische Klassen von Ankündigungen und Ereignissen definiert. Diese setzen die initiale Platzierung in Zusammenarbeit mit der Platzierungsengine um. Auch für die laufende Anwendungsplatzierung wurden zur besseren Abgrenzung eigens spezifizierte Ereignistypen und Engines realisiert. Der Austausch von Informationen erfolgt bei der initialen wie auch bei der laufenden Platzierung ausschließlich über die dazu definierten Ereignis- und Ankündigungstypen. Auf eine separate Ausführung der Basisoperationen Dekomposition und Rekombination wird verzichtet, da dies eine Entwickleraufgabe ist, welche die Middleware lediglich durch Aufruf einer entsprechend implementierten Methode in Gang setzen würde. Da bei dem in dieser Arbeit vorgestellten Ansatz vor allem die Weiterleitungskosten zwischen den jeweiligen Brokern betrachtet werden und die interne Kommunikation nicht

analysiert wird, ist der hier gewählte Implementierungsansatz vor allem pragmatisch. Eine spätere Erweiterung ist durch die Umsetzung des Prinzips der Feature Composition jederzeit möglich.

Für die umgesetzten Platzierungsoperationen existieren ebenfalls dedizierte Engines, wodurch ein unabhängiger Einsatz der Platzierungsaspekte möglich ist. Die Engines implementieren die in einer Schnittstellenklasse definierten Funktionalitäten über den Weg einer abstrakten Klasse. Je Broker wird eine entsprechende Engine hinzugefügt, welche Informationen über die ein- und ausgehenden Ereignisse analysiert. Diese Engine sammelt die Informationen ein und gibt diese an spezifische Strategie-Klassen weiter. Diese entscheiden über die Durchführung einer Migration oder Replikation. Migration und Replikation sind unabhängig voneinander implementiert, allerdings erweitert die Replikation den vorhandenen Workflow der Migration. Die Ausführung von Migrationen und Replikationen setzt voraus, dass die zu platzierenden Anwendungen diese Basisoperationen auch unterstützen. Dazu wurden gemäß dem Entwicklungsprinzip von REBECA entsprechende Schnittstellen definiert, wobei auch hier eine Hierarchie zwischen Migration und Replikation zu erkennen ist.

Kapitel 5

Evaluation

Inhalt

5.1 Motivation

In dem in dieser Arbeit beschriebenen Anwendungsbeispiel wird eine Mischung fest installierter Geräte, als Rückgrat, und mobiler Geräte verwendet. Gerade diese mobilen Geräte, darunter nutzerindividuelle Geräte, Sensoren und Aktuatoren, besitzen unterschiedlich starke Rechenleistungen und Batteriekapazitäten. Dabei ist hervorzuheben, dass besonders Sensoren und Aktuatoren über eine begrenzte Batteriekapazität verfügen. Sowohl mobile, als auch fest installierte Geräte sind zu einem Ensemble zusammengefasst, welches gemeinsam Dienste erbringt. Diese Ensembles werden auch als intelligente Umgebungen bezeichnet [181]. Sie zeichnen sich außerdem durch eine inhärente Dynamik aus, sodass sowohl Anwendungen als auch Geräte während der Laufzeit in das Ensemble von Geräten und Anwendungen aufgenommen werden oder dieses verlassen. Zur Dynamik gehören, neben dem geplanten Ausscheiden und Dazukommen von Geräten und Anwendungen, auch die Schwankungen der Intensität von Kommunikationsflüssen, welche sich ebenfalls auf die Auslastung der Kommunikationsinfrastruktur auswirken. Statt fester Kommunikationswege ist im Bereich dynamischer Geräteensembles eine indirekte, datenorientierte Kommunikation sinnvoll. So ist zum einen die Kommunikation unabhängig von den jeweiligen Geräten und inhaltsbasiert umsetzbar, zum anderen lassen sich Änderungen durch hinzukommende und weggehende Geräte und Anwendungen einfacher umsetzen.

Das in dieser Arbeit vorgestellte Konzept zur Verteilung von Anwendungskomponenten basiert auf einer ereignisorientierten Kommunikation von Anwendungen und Geräten, deren Anwendungsfall in intelligenten Umgebungen zu finden ist. In diesen verteilten Umgebungen sind Informationen zu sammeln und auszuwerten, sodass der Nutzer in der Lage ist, gewünschte Informationen zu erkennen und darauf zu reagieren. Die Detektion von interessanten Situationen benötigt jedoch die Sammlung von Informationen aus unterschiedlichen Quellen, so dass Informationsströme entstehen und diese üblicherweise zu festgelegten Orten transportiert und dort ausgewertet werden. Je stärker die Informationsquellen verteilt sind und je zentraler die Sammlung sowie die Auswertung der Informationen erfolgt, je mehr Informationsflüsse sind im Netzwerk durchzuführen. Kann jedoch die Erkennung relevanter Informationen auch schrittweise durchgeführt werden, so können Informationsflüsse innerhalb des Netzwerks verringert werden, vorausgesetzt, es lassen sich nicht benötigte Informationen damit filtern, sodass diese nicht weitergeleitet werden. Eine Beispielanwendung dafür ist die verteilte Detektion komplexer Ereignismuster, welche den ersten Teil (Abschnitt 5.2) dieses Kapitels einnimmt.

Der anschließende Abschnitt 5.3 dieses Kapitels erörtert die Nutzbarkeit des in dieser Arbeit vorgestellten Konzepts zur Beschreibung der Einsparung von Nachrichten in einer intelligenten Umgebung. Dazu wurde das Konzept in die genutzte Middleware REBECA integriert. Mit den dort eingefügten Aspekten (siehe Abschnitt 4) werden Experimente zur Evaluation des Systems durchgeführt. Die Ergebnisse einer Evaluation sind genau dann am aussagekräftigsten,

wenn sie unter realen Bedingungen mit einem ebenso realen, d.h. deployten System durchgeführt werden. Dazu werden, neben einem verfügbaren System (mindestens im Prototypenzustand) ausreichend Ressourcen, wie Netzwerkknoten und Kommunikationsverbindungen benötigt, also müssen physisch ausreichend Ressourcen verfügbar sein. Sollen Messungen mit einem realen System durchgeführt werden, ist selbiges in dieser Zeit nicht produktiv nutzbar. Zudem ist für die Durchführung eines Experiments nicht ein einziger Lauf, sondern sind ausreichend viele Experimentläufe nötig, wodurch die Belastung des realen Systems weiter gesteigert wird. Durch das Experimentieren mit verteilten Systemen können zudem zusätzliche Seiteneffekte entstehen. Der Vorteil, dass Experimente mit realer Hardware und produktiven Softwaresystemen genauere Ergebnisse fördern können, wird durch eben diese realen Probleme wieder eingefangen. Darüber hinaus sind reale Systeme i.d.R. ebenfalls begrenzt, da ein Experiment mit sehr vielen physischen Knoten und Kommunikationsverbindungen häufig schlicht nicht zur Verfügung steht und damit auf natürliche Weise die Netzwerkgröße auf kleine bzw. mittlere Netzwerke eingrenzt, außerdem müssen die Netzwerkparameter passend zum verfügbaren Netzwerk gewählt werden [169]. Simulationen bieten demgegenüber die Möglichkeit, beliebig große Netzwerke zu erstellen und in ihrem Verhalten zu simulieren. Sie ermöglichen zudem, die Parameter wie gewünscht einzustellen und so unabhängig vom realen System zu agieren. Insbesondere bei beschränkten Kapazitäten lässt sich die Simulationsgenauigkeit gegen die verfügbaren Ressourcen abwägen [169], womit die Simulation ein probates Mittel zur Evaluierung von Konzepten ist.

Die Evaluation in diesem Kapitel ab Abschnitt 5.4 stützt sich auf die Simulation des in dieser Arbeit vorgestellten Ansatzes zur Verteilung von Anwendungskomponenten, genauer gesagt, auf die verteilte Erkennung von Ereignismustern. Durch die verteilte Detektion von Ereignismustern besteht die Möglichkeit, nicht benötigte Ereignisse bzw. Teilereignismuster frühzeitig herauszufiltern, sodass diese nicht mehr im Netzwerk weitergeleitet werden. Je früher diese Filterung stattfindet, desto mehr Ereignisse werden nicht weitergeleitet und Netzwerkressourcen werden geschont. Die Ereignismusterdetektion wird simuliert durch sog. *Worker*, welche den Ereignisfluss unterschiedlich stark verstärken sowie *Consumer* als Interessenten der Ereignismuster und *Producer* als Emittenten von Ereignissen. In diesem Umfeld wird das in dieser Arbeit vorgestellte (siehe Abschnitt 3) und umgesetzte Konzept (siehe Abschnitt 4) vor allem quantitativ betrachtet. Damit wird untersucht, ob sich durch den vorgestellten Ansatz Kommunikationsaufwand einsparen lässt. Die Sicherung der qualitativen Ansprüche wird dagegen durch die vorhandenen Komponenten der eingesetzten Middleware REBECA gewährleistet. An dieser Stelle sei der Hinweis erlaubt, dass der hier vorgestellte und implementierte Ansatz als Komponente (Plugin) dem System hinzugefügt wird und damit die übrigen Eigenschaften der Middleware nicht berührt werden.

5.2 Komplexe Ereignismuster

Anwendungen in intelligenten Umgebungen soll die Möglichkeit gegeben werden, sich kontextabhängig zu verhalten und auf Zustandsänderungen reagieren zu können. Zur Detektion von Kontexten und Zustandsänderungen lassen sich gesammelte Informationen einsetzen, welche bei der ereignisgetriebenen Kommunikation durch Ereignisse übertragen werden.

So kann bspw. die grafische Darstellung eines Stadtplanes auf einem Gerät mit kleinem Display weniger detailreich erfolgen als auf einem Tablet-PC, einem Notebook oder gar einer Videoleinwand. Dazu müssen Geräte jedoch in der Lage sein, den aktuellen Kontext zu erkennen. Das bedeutet, Ereignisse über die relevanten Zustände müssen gesammelt werden. Die Anwendung muss sich für diese Informationen subskribieren, um darauf reagieren zu können. Ereignisbasierte Kommunikation ist damit für die Erkennung von Zuständen in intelligenten Umgebungen geeignet. Durch die lose Kopplung von Sender und Empfänger lassen sich zudem neue Anwendungen, Geräte und Ereignisquellen komfortabel integrieren. Die Granularität der Ereignisse selbst ist weit gefächert. Sie reicht von einfachen, *primitiven Ereignissen*, wie Sensorwerte, Temperatur, Luftfeuchtigkeit und der Detektion von einer Bewegung bis hin zu *zusammengefassten Ereignissen* bzw. *komplexen Ereignismustern* oder *räumlich-zeitlichen Ereignismustern*. Diese werden in der englischsprachigen Literatur auch als *Complex Events*, *Composite Events* oder *spatio-temporal Events* bezeichnet. Solche Ereignismuster erkennen komplizierte Zusammenhänge wie „Beginn einer Präsentation“ oder „Person X benötigt Hilfe“.

Komplexe Ereignisse dienen dazu, Kontexte zu erkennen, sodass die Anwendungen auf die Änderungen entsprechend reagieren können. Gleichzeitig lassen sich komplexe Ereignismuster selbst als ein Beispiel für adaptives Verhalten heranziehen. So bestehen Ereignismuster regelmäßig aus Teilmustern (*partielle Ereignismuster*), die entweder selbst aus primitiven Ereignissen oder weiteren, komplexen Ereignismustern bestehen und über Operatoren miteinander verbunden sind. An den Operatoren können die Muster in Teilmuster bzw. in primitive Ereignisse zerlegt werden. Können Teilmuster bzw. primitive Ereignisse an unterschiedlichen Stellen innerhalb eines Netzwerks erkannt werden, lässt sich die technische Detektion ebenfalls dorthin verschieben, wo sie sich günstig durchführen lässt. Komplexe Ereignismuster, auch als räumlich-zeitliche Ereignismuster bezeichnet, lassen sich als Menge von Ereignissen spezifizieren, die definierten Bedingungen genügt. Die Bedingungen beziehen sich auf den Inhalt, sowie die örtlichen und zeitlichen Eigenschaften der Ereignisse. Diese werden je partiellem Ereignis ausgewertet. Stimmen sie überein, bilden sie *Kandidatenereignisse*. Genügen diese wiederum den Anforderungen der nächsten Stufe, bilden sie zusammen mit weiteren ausgewählten Kandidatenereignissen das komplexe Ereignismuster bzw. die nächste Stufe einer Menge von Kandidatenereignissen.

Die Bedingungen für die Ereignismenge werden mit Hilfe von Beschränkungen auf Variablen ausgedrückt, welche miteinander verknüpft sind und komplexe

Ausdrücke bilden können. Einzelne Variablen werden durch logische Operatoren wie $\vee$, $\wedge$ und $\neg$ sowie Existenz- und Universalquantoren wie $\exists$ und $\forall$ ausgedrückt. Zur Verknüpfung der Teilausdrücke wurden unterschiedliche *Ereignisalgebren* entwickelt, wobei jeweils verschiedene Zielstellungen im Blickpunkt stehen. Eine Entwicklungsrichtung ergibt sich aus den aktiven Datenbanken [35]. Eine andere Richtung sind Streaming-Datenbanken mit der SQL-ähnlichen Beschreibungssprache CQL (Continuous Query Language) [148]. Ein weiterer Hauptbestandteil von Ereignisalgebren ist neben den von den Quantoren abgeleiteten *Ereigniskompositionsoperatoren* (oder kurz *Ereignisoperatoren*) die Angabe der verwendeten *Konsumptionsstrategie*. Diese werden in der englischsprachigen Literatur als *Consumption Modes* bezeichnet. Ereignisoperatoren beschreiben, wie komplexe Ereignismuster komponiert werden. Damit wird spezifiziert, welche Bedingungen die zu komponierenden Ereignisse erfüllen müssen. Verbreitete Algebren enthalten bspw. Operatoren für *Disjunktion* ($E_1 \vee E_2$), *Konjunktion* ($E_1 \wedge E_2$), *Sequenz*($E_1; E_2$), *Beliebig*, $E_1^*(t_i)$ und *Nicht* ($\neg(E_1, t_1)$). Konsumptionsstrategien dagegen beschreiben, welche Ereignisse für die Komposition überhaupt zur Verfügung stehen, also welche partiellen Ereignisse vorgehalten und kombiniert werden können. Weiterhin beschreiben sie, wie nach der Komposition mit den Ereignissen bzw. den Teilereignismustern verfahren wird. Häufig geschieht die Definition der Konsumptionsstrategien in Abhängigkeit von der Zeit. So existieren bei stromgetriebenen Anwendungen häufig Zeitfenster, die bestimmen, welche Ereignisse wie lange aufgehoben und kombiniert werden können. Ein anderes Konzept verfolgen dagegen die Konsumptionsstrategien in [35], bei denen genau beschrieben wird, wie ein Kandidatenereignis aus einer Ereignismenge ausgewählt wird, mit welchen anderen Ereigniskandidaten ein komplexes Ereignis gebildet und welche Ereignisse zwischengespeichert werden.

Das Anwendungsumfeld intelligenter Umgebungen ist durch die Kommunikation und Kooperation von Anwendungen und Geräten, und somit durch eine verteilte Umgebung, gekennzeichnet. Bei der Umsetzung ereignisbasierter Kommunikation mit Publish/Subscribe existieren in einer verteilten Umgebung Weiterleitungspfade von Notifikationen, welche sich aus aktiven Subskriptionen und Ankündigungen ergeben. Die Notifikationen werden schrittweise von den Ereignisproduzenten zu den Interessenten durch Anwendung des jeweiligen Routingalgorithmus weitergeleitet.

Die Detektion komplexer Ereignismuster ist im einfachsten Fall Aufgabe der Anwendung, die an dem jeweiligen Ereignismuster interessiert ist. Hierbei subskribiert sich die Anwendung für alle primitiven Ereignisse, die möglicherweise für die Erkennung eines komplexen Ereignisses relevant sein könnten. In einer verteilten Umgebung werden Ereignisse jedoch nicht zwangsläufig in der Nähe der Anwendung detektiert, die sich für diese Ereignisse interessiert. Daher müssen die Notifikationen mittels der Publish/Subscribe-Infrastruktur vom Publisher zum Subscriber geleitet werden. Allerdings sind an der Weiterleitung der Notifikationen, gerade in Sensornetzen, Geräte mit begrenzten Ressourcen beteiligt.

Die Weiterleitung von Notifikationen bedeutet eine Belastung der Ressourcen aller an der Weiterleitung beteiligten Geräte. Werden Ereignisse verteilt im Netzwerk detektiert und an die Subskribenten weitergeleitet, ist der daraus bedingte Ressourcenverbrauch insbesondere bei mobilen Geräten ein wichtiges Kriterium hinsichtlich der Stabilität einer Gerätekonfiguration. Dazu kommt, dass viele Kandidatenereignisse kein Teil eines komplexen Ereignismusters formen, sondern aufgrund der Kriterien an das Ereignismuster verworfen werden. Vor diesem Hintergrund und den begrenzten Ressourcen sind Notifikationen, die nicht den Anforderungen an das komplexe Ereignismuster genügen, frühzeitig auszusortieren. Durch das Ausnutzen der Selektivität von komplexen Ereignismustern bzw. ihrer Teilmuster und durch die verteilte Detektion sind im Hinblick auf die Ressourceneffizienz die nicht den Kriterien entsprechenden Notifikationen und Teilmuster so früh wie möglich herauszufiltern und nicht weiterzuleiten.

Ein in die Middleware integrierter Notifikationsdienst übernimmt die Weiterleitung der Notifikationen. Von daher ist es zum Zwecke der Einsparung nicht benötigter Notifikationen sinnvoll, die verteilte Detektion von Ereignismustern als Service der Middleware zu organisieren. Dies ermöglicht der Middleware, die Detektionskomponenten eigenständig zu organisieren, sodass die Detektion bestmöglich effizient im Sinne der Konsumption von Netzwerkressourcen, Speicherplatzverbrauch und Rechenleistung organisiert werden kann. Natürlich muss die verteilte Detektion komplexer Ereignismuster auch in einem dynamischen Umfeld die gewünschte Effizienzsteigerung leisten. Durch die Dynamik entsteht ein komplexes Online Optimierungsproblem, welches NP-schwer und somit in akzektabler Zeit nur mit einer Heuristik lösbar ist. Der in dieser Arbeit vorgestellte Ansatz bietet eine solche Heuristik.

Die verteilte Detektion komplexer Ereignismuster in einer dynamischen Umgebung beruht wie der Ausgangsfall der verteilten Anwendungen auf der Dekomposition, Replikation, Migration und Rekombination und ist somit als Anwendungsbeispiel für den in dieser Arbeit beschriebenen Platzierungsansatz geeignet. Zur Lösung des Verteilungsproblems verteilter Ereignisdetektoren nutzt das in dieser Arbeit vorgestellte Konzept das Grundprinzip der Selbstorganisation [111, 157] mittels lokalem Wissen. Das Konzept wurde zunächst in [193] beschrieben und in [194] für allgemeine Anwendungskomponenten weiterentwickelt.

5.2.1 Detektoren komplexer Ereignismuster

Komplexe Ereignismuster symbolisieren, wie auch primitive Ereignisse, Zustände der Umwelt oder von Systemen. Komplexe Ereignismuster ermöglichen die Erkennung von komplexen Zuständen bzw. von Zustandsänderungen besser, als primitive Ereignisse dazu in der Lage sind. So können bestimmte Zustände detektiert und anschließend kann darauf reagiert werden, so wie es klassischerweise das Prinzip der *ECA (Event Condition Action)*-Regeln vorsieht. Die Detektion komplexer Ereignismuster beinhaltet zwei Aspekte: (1) die Spezifikation und (2)

die Detektion der Muster. Während die Spezifikation dazu dient, die Ereignismuster zu beschreiben, dient die Detektion selbst der technischen Umsetzung der Erkennung. Die Spezifikation kann bspw. durch Automaten oder eine Algebra erfolgen. Sie ermöglicht den Nutzern bzw. den Anwendungen, beteiligte Ereignisse sowie temporale, örtliche und inhaltliche Bedingungen auf den Ereignissen zu formulieren. Die technische Umsetzung der Automaten oder der Algebra erfolgt durch die Detektoren. Die bei der Detektorrealisierung eingesetzten Techniken bestimmen dann, wie Teilergebnisse zwischengespeichert werden.

Ereignisalgebra

Ereignisse als Geschehnisse von Interesse werden in Form von Notifikationen materialisiert, welche neben den reinen Ereignisinformationen i.d.R. zusätzliche Kontextinformationen enthalten. Komplexe Ereignismuster lassen sich inhaltsbasiert, aber auch anhand von temporalen und räumlichen Eigenschaften spezifizieren. Ereignisse formen damit nur dann Muster, wenn diese spezifizierten Anforderungen genügen. Hingegen werden alle jemals in einem System aufgetretenen und detektierten Ereignisse in die *Ereignishistorie* integriert. Wird ein neues Ereignis detektiert, so wird es zunächst in die Gesamtheit der Ereignishistorie aufgenommen. Anschließend wird ermittelt, ob das neue Ereignis die definierten Bedingungen an Inhalt und temporale sowie räumliche Eigenschaften erfüllt, um Teil eines komplexen Ereignisses zu werden. Erfüllt es diese, geht es als Teilmuster in ein komplexes Ereignismuster ein, wobei die einzelnen Teilmuster durch die Ausdrücke der Algebra miteinander verknüpft werden.

Die verteilte Detektion komplexer Ereignismuster ist als Anwendungsbeispiel für die Verteilung von Anwendungen geeignet, ermöglicht doch das Aufbrechen von Ereignismustern in Teilmuster die Zerlegung von Detektoren und deren individuelle Verschiebung. Mit Hilfe einer hinterlegten Kosten- und Zielfunktion lässt sich die Anwendungsverteilung zudem zielgerichtet durchführen. Grundlage für eine Zerlegung von Ereignismustern in Teilmuster und damit von Detektoren in Teildetektoren ist eine Algebra, die eine Beschreibung von Ereignismustern in der Form ermöglicht, dass Muster in Teilmuster aufgebrochen und separat detektiert werden können. Im Folgenden werden zunächst verschiedene Methoden beschrieben, wie sich Ereignismuster und somit Anforderungen an Ereignisse spezifizieren lassen.

Bedingungen (engl. *constraints*) können auf den in den Ereignisnotifikationen hinterlegten Informationen, aber auch auf temporalen und örtlichen (Zusatz-) Informationen des Ereigniskontexts definiert werden. Temporale Bedingungen auf einem Ereignis fordern bspw., dass ein Ereignis innerhalb eines bestimmten Zeitraums (z.B. innerhalb eines Tages, zwischen 12 und 13 Uhr usw.) auftritt. Ein weiteres Beispiel für eine temporale Bedingung ist das Auftreten eines Ereignisses in einem bestimmten Zeitraum nach der Detektion eines anderen Ereignisses, womit eine zeitliche Korrelation zwischen Ereignissen besteht. So wird u.a ein komplexes Ereignis erkannt, wenn ein Ereignis E_2 innerhalb einer Zeitspanne

von weniger als 10 Sekunden nach dem Auftreten von Ereignis E_1 detektiert wird:

$$\{(E_1, E_2) \mid [E_1.time < E_2.time] \wedge [(E_2.time - E_1.time) < 10\ sec]\}.$$

Andere Restriktionen fordern bspw. das Auftreten von mehr als zehn Ereignisinstanzen innerhalb einer bestimmten Zeit, oder sie schränken die Ereigniskomposition derart ein, dass nur die neuesten Ereignisinstanzen für ein komplexes Ereignismuster genutzt werden dürfen. Mittels räumlicher Bedingungen kann demgegenüber gefordert werden, dass ein Ereignis in einem bestimmten Umkreis um einen definierten örtlichen Punkt auftreten muss. Auch können räumliche Bedingungen auf mehreren Ereignissen formuliert werden, sodass z.B. die Raumnummer von zwei Ereignissen gleich sein muss, damit ein komplexes Ereignis detektiert werden kann. In der praktischen Umsetzung sind Bedingungen zumeist eine Sammlung verschiedener Typen von Anforderungen. Ein Beispiel für gemischte Bedingungen findet sich in Mühl et al. [156] durch die Definition von sog. *Funnel Functions*, zu deutsch *Trichterfunktionen*.

Spielt die Unterscheidung zwischen inhaltlichen, temporalen und räumlichen Informationen eines Ereignisses theoretisch zwar eine Rolle, werden in der Praxis die zusätzlichen Informationen jedoch zumeist, analog zu den Ereignisinformationen, in gleicher Form weitergeleitet. Die zusätzlichen Informationen sind damit ein Teil des Notifikationsinhaltes. Es hat sich durchgesetzt, die von Ereignisnotifikationen getragenen Informationen in Form von Name/Wert-Paaren (engl. *name/value pairs*) zu modellieren. So besteht jede Notifikation aus einer Menge von Name/Wert-Paaren, welche jeweils als *Attribute* bezeichnet werden. Die definierten Bedingungen beziehen sich auf die Attributwerte und müssen von ihnen erfüllt werden, um ein komplexes Ereignismuster zu formen. Die Menge der Ereignisse, welche dazu untersucht werden und potentiell ein komplexes Ereignis formen können, wird dabei als *Kanditatenmenge*, die einzelnen Ereignisse als *Kandidaten* bezeichnet.

Bedingungen, deren Auswertung auf einzelne Ereignisse herunter gebrochen werden kann, lassen sich prinzipiell einfach in die Nähe der jeweiligen Ereignisquelle verschieben. Dort ist die Filterung von den die Anforderungen nicht erfüllenden Ereignissen frühzeitig und sinnvoll im Sinne der Ressourceneffizienz durchführbar. Dies verdeutlicht das folgende Beispiel aus dem Bereich der Klimasteuerung eines Gebäudes. Gegeben sei eine Komposition zweier Ereignisse E_1 und E_2, wobei E_1 Temperaturereignisse und E_2 Luftfeuchtigkeitsereignisse repräsentieren. Die Spezifikation eines komplexen Ereignismusters

$$\{(E_1, E_2) \mid [E_1.humid > 50\%] \ \wedge \ [E_2.temp > 30\ ^\circ\mathrm{C}]\}$$

bedeutet, dass ein komplexes Ereignismuster aus allen E_1 und E_2 detektiert wird, wenn die Anforderungen an die Temperatur (30 °C) und Luftfeuchtigkeit ($>$ 50%) erfüllt sind. Da keine zeitlichen Einschränkungen formuliert wurden, wird das komplexe Ereignismuster aus allen in Frage kommenden Ereignissen E_1 und

E_2 kombiniert. Darunter fallen auch alle historischen Ereignisse. Als temporale Einschränkung sind beide Ereignisse innerhalb von 10 Sekunden unabhängig von ihrer Reihenfolge zu detektieren. Damit ergibt sich die Spezifikation

$$\{(E_1, E_2) \mid [E_1.humid > 50\%] \ \wedge \ [E_2.temp > 30\ ^\circ\text{C}] \wedge [|E_1.time - E_2.time| < 10\ sec]\}.$$

Die inhaltliche Spezifikation des Musters setzt sich aus zwei voneinander unabhängigen Teilspezifikationen zusammen, welche sich jeweils auf die Auswertung einzelner Ereignisse beziehen. So formulierte Anforderungen können in diesem Beispiel in Richtung der Ereignisquellen verschoben und nicht den formulierten Bedingungen entsprechende Ereignisse herausgefiltert werden. Komplexe Bedingungen mit gegenseitigen Abhängigkeiten, welche nur zusammen ausgewertet werden können, lassen sich nicht ohne weiteres aufspalten und separat voneinander auswerten. Als Beispiel dient wieder die Klimatisierungssteuerung von Räumen. In diesem Fall ist eine Anwendung daran interessiert, ob die Differenz der von zwei Temperatursensorsen gemessenen Werte in einem Raum mehr als 10 °C beträgt, was in der Praxis auf einen Fehler bei mindestens einem der Sensoren hinweist. In einem Raum arbeiten jeweils zwei Temperatursensoren, welche die Ereignisse E_1 und E_2 publizieren. Die Spezifikation des Musters lautet $\{(E_1, E_2) \mid [|E_1.temp - E_2.temp| > 10\ ^\circ\text{C}] \ \wedge \ [E_1.room = E_2.room]\}$. Wichtig zu sehen ist, dass das Muster nicht in einzelne Teilmuster aufgespaltet werden kann, da für die Auswertung der Bedingung beide Sensorwerte zusammen gebraucht werden. Ein komplexeres Beispiel ist das Interesse an einem Raum, bei dem sowohl die Temperatur als auch die Luftfeuchtigkeit außerhalb eines bestimmten Schwellwerts liegen. Dazu werden wieder die Ereignisse E_1 für die Messung der Temperatur und E_2 zur Bestimmung der Luftfeuchtigkeit genutzt. Um die Räume mit einer Temperatur von über 30 °C und einer Luftfeuchtigkeit über 90% zu ermitteln, wird die folgende Spezifikation formuliert:

$$\{(E_1, E_2) \mid [E_1.temp > 30\ ^\circ\text{C}] \ \wedge \ [E_2.hum > 90\%] \ \wedge \ [E_1.room = E_2.room]\}.$$

Damit lässt sich das komplexe Muster in die Komponenten für die Temperaturmessung und die Bestimmung der Luftfeuchtigkeit zerlegen. Allerdings ist in diesem Beispiel noch keine temporale Beschränkung enthalten. Aufbauend auf das Ereignismuster zur Erkennung der Temperaturdifferenz wird die Spezifikation um einen temporalen Aspekt erweitert, sodass sichergestellt wird, dass nur die neuesten Instanzen von E_1 und E_2 für die Komposition zu einem komplexen Ereignismuster genutzt werden. $\{(E_1, E_2) \mid [E_1.room \neq E_2.room] \ \wedge \ [\nexists E_3 : E_3.room = E_1.room \ \wedge \ E_3.time > E_1.time] \ \wedge \ [\nexists E_4 : E_4.room = E_2.room \ \wedge \ E_4.time > E_2.time] \ \wedge \ [|E_1.temp - E_2.temp| > 10\ ^\circ\text{C}]\}$. Dies entspricht dem Kosumptionsmodus *recent* (siehe Abschnitt 5.2.1).

Algebren zur Spezifikation von Ereignismustern haben sich analog zu den Anwendungsfeldern komplexer Ereignismuster entwickelt. Im Zuge der Entstehung von aktiven, temporären und deduktiven Datenbanken hat sich auch die Spezifikation von Ereignismustern herausgebildet. Parallel dazu entwickelten sich

Beschreibungssprachen für Ereignismuster zur Stromverarbeitung (engl. *stream processing*) [104, 105]. Eine andere Entwicklungsrichtung mit entsprechenden Beschreibungssprachen ist die Modellierung von Geschäftsprozessen [27] oder die Wissensrepräsentation [104, 105]. So nennen Hinze und Voisard in ihren Arbeiten [104, 105] Beispiele für Beschreibungssprachen in den Bereichen aktive Datenbanken [174], Data Mining, Datenstromverarbeitung und Wissensrepräsentation. Ein Hauptaugenmerk der aktiven Datenbanken ist und war demnach die Spezifikation von ECA-Regeln für eingetretene Ereignisse in aktiven Datenbanken. Aktive Datenbanken können neben den reinen Ereignisinformationen auf zusätzliche Informationen, wie Transaktionsinformationen und -kontexte reagieren und Aktionen auf der Grundlage von veränderten (Datenbank-) Zuständen, wie beim Vorhandensein alter und neuer Tupel, auslösen.

In [105] wird auf die Schwerpunkte der aufgestellten Beschreibungssprachen (insbesondere komplexe Ereignismuster [36, 76, 78]) und deren temporäre Aspekte [52] hingewiesen und unterschieden. Temporäre Aspekte spielen aus der Historie insbesondere bei der Verknüpfung von Datenbankenzuständen eine Rolle, wobei aktive Datenbanken sich auf datenbankinterne Zustände beziehen [104, 105]. Eine Abbildung zeitlicher Aspekte ist bspw. mit intervallbasierten Operatoren [3, 147] möglich. Wie [104, 105] verdeutlichen, ist neben der Ordnung auf der Grundlage von Datenbankzuständen auch die Ordnung auf der Basis von Ereignissen möglich. Ein Beispiel ist SAMOS [77], während in [215] ein genereller Überblick über die Semantik der möglichen temporären Operatoren gegeben wird. In den bisher genannten Arbeiten beschreiben die dort vorgestellten Semantiken auf temporärer Ebene den Zusammenhang von Zuständen und Ereignissen, dabei greifen sie jedoch auf die Gesamtheit der Ereignisse zu und beschränken die möglichen Ereigniskombinationen nicht. Sollen jedoch nur Zustände oder Ereignisse eines bestimmten Zeitraums/Zeitfensters betrachtet werden, muss eine andere Semantik gewählt werden. Eine weitere Gruppe von Beschreibungssprachen für die Ordnung von Ereignissen und Zuständen bilden temporale Logiken [4], welche in [147] genutzt wurden. Aus der Geschichte der Datenbanken bietet es sich an, SQL bzw. einen Dialekt als Beschreibungs- und Anfragesprache für komplexe Ereignismuster zu nutzen. So finden sich Erweiterungen von SQL um temporale Logiken in der Arbeit [199]. Ähnlich wie bei der Entwicklung der aktiven Datenbanken haben sich unterschiedliche Beschreibungssprachen für Ereignismuster auch in anderen Anwendungsbereichen herausgebildet. So nennen Hinze und Voisard [104, 105] Methoden des Data Minings durch Zeitreihenanalyse auf der Grundlage temporaler Assoziationsregeln und die Verknüpfung von Ereignissen mittels Datenstromverarbeitung.

Mit Hilfe komplexer Ereignismuster lassen sich auch Geschäftsprozesse beschreiben, die klassischerweise mittels ereignisorientierten Prozessketten oder in UML-Diagrammen notiert werden. Geschäftsprozesse werden jedoch oft nur semiformal beschrieben, Umsetzung auf die Ereignisebene erfordert eine Interpretation der Beschreibungen. In [27] wird gezeigt, wie eindeutig unterschiedliche Modellierungstechniken sind und in welchem Maße die Techniken für die ereignisorientierte Beschreibung von Geschäftsprozessen geeignet sind. Darüber hinaus

wird diskutiert, durch welche Aspekte Modellierungstechniken zu erweitern sind, um mittels Ereignismustern Geschäftsprozesse eindeutig beschreiben zu können. Dazu ist vor allem der temporale Aspekt und die Kombination/Auswahl von Ereignissen von Bedeutung, welche in den klassischen Beschreibungssprachen nur unzureichend abgebildet wurden [104].

Im Laufe der Zeit hat sich eine Vielzahl von ereignisbasierten Systemen herausgebildet, wobei jedes System zumeist eine eigene Ereignisbeschreibungssprache verwendet [51, 77, 80, 134, 138, 177]. Trotz der Vielzahl eigener Beschreibungssprachen für Ereignismuster finden sich jedoch bei allen Sprachen die gleichen Konzepte. Beispiele hierfür sind die sog. *Coupling Modes* bzw. *Consumption Modes* [51] und *Parameter Contexts* [35] in den Sprachen HiPAC bzw. SNOOP. Während Coupling Modes die mögliche zeitliche Verzögerung zwischen Ereignisdetektion, Regelauswertung und Aktionsausführung beschreiben, dienen Parameterkontexte zur Spezifizierung, welche Teilereignisse bzw. Teilereignismuster an den Operatoren miteinander kombiniert werden können.

Dem in dieser Arbeit vorgestellten Ansatz zur Aufspaltung von Komponenten, insbesondere der Aufspaltung von Ereignismustern in Teilmuster und deren Detektion, liegen operatorbasierte Algebren zu Grunde. Diese beschreiben komplexe Ereignismuster, indem Ereignisse bzw. Ereignismuster mit Hilfe von Operatoren zu komplexeren Ausdrücken zusammengefasst werden. Die Vielfalt der operatorbasierten Ereignisalgebren ist vergleichsweise groß [103, 147, 217]. Die Operatoren der Algebren sind zumeist ähnlich und beschreiben i.d.R. die Art und temporale Abfolge von Ereignissen, die zur Detektion des beschriebenen Ereignismusters notwendig sind. Klassischerweise kommen die Operatoren *Konjunktion, Sequenz* und *Negation* vor. Ein Vergleich der Operatoren in verschiedenen Algebren findet sich in [217]. Aufgrund der Unterschiede zwischen den augenscheinlich gleichen Operatoren aber unterschiedlichen Semantiken (z.B. aufgrund der eingesetzten temporalen Semantik wie Punktsemantik und Intervallsemantik), können Ausdrücke einer Algebra nicht beliebig in eine andere Algebra übersetzt werden [104].

Die Beschreibung der Ereignismuster orientiert sich oft an der jeweiligen Umsetzung und damit vor allem am Anwendungszweck. Wie Ereignismuster aus technischer Sicht erkannt werden, beschreiben die Algebren nicht, sondern definieren Ereignismuster aus konzeptioneller Sicht. Als Beschreibungssprachen für die technische Umsetzung existieren bspw. reguläre Ausdrücke [80] und endliche Zustandsautomaten [178].

Wurden Ereignismuster erfolgreich detektiert, existieren unterschiedliche Optionen, welche Informationen konkret an die Interessenten weitergegeben werden [104]. Es ist zu entscheiden, ob die Interessenten sämtlichen Inhalt der Teilereignisse bzw. Teilereignismuster zur Verfügung gestellt bekommen, oder ob die detektierten Ereignismuster eine eigene, spezifizierte Datenstruktur bilden und nur diese ausgeliefert wird. Dies schließt unter Umständen die Selektion einzelner Attribute, deren Konvertierung oder Weiterverarbeitung mit ein. Am Ende dieses Prozesses kann auch die Publikation der neuen Datenstruktur mit den

enthaltenen Informationen als neues Ereignis stehen. Auf diese Weise stehen die Informationen nicht mehr exklusiv einer Anwendung zur Verfügung, sondern können von weiteren Subskribenten (wieder-) verwendet werden.

Genutzte Algebra. Bedingungen werden auf der gesamten Menge aller (historischen) Ereignisse ausgewertet. Beschränkungen können jedoch aufgrund ihres Umfangs durch eine Vielzahl von Variablen, logischen Operatoren wie $\vee$, $\wedge$ und $\neg$ sowie Quantoren $(\exists, \forall)$ sehr komplex und schwer verständlich sein. Wie gesehen, existiert eine Vielzahl möglicher Algebren, genauso wurde aber auch gezeigt, dass sie sich inhaltlich ähneln und hauptsächlich aus *Operatoren für die Ereigniskomposition* und aus Beschreibungen der *Consumption Modes* bzw. *Konsumptionsmodi* bestehen. Während die Operatoren die Komposition von komplexen Ereignissen beschreiben, definieren die Konsumptionsmodi, welche Ereignisse für die Bildung eines Ereignismusters überhaupt zur Verfügung stehen und wie mit ihnen nach erfolgter Detektion verfahren wird.

Die Zerlegung von Ereignismustern und ihrer Detektoren ist der Anwendungsfall des in dieser Arbeit vorgestellten Konzepts. Es wird das im Folgenden vorgestellte Verhalten von Operatoren zugrunde gelegt und durch Beispiele ergänzt. Die Beispiele orientieren sich an einer Gebäudesteuerung mit Ereignissen für die Temperaturwerte (Ereignisse E_1), Luftfeuchtigkeit (Ereignisse E_2), einen Bewegungssensor (Ereignisse E_3) und einen Einbruchsmelder (Ereignisse E_4). Die Sensoren erzeugen dabei grundsätzlich nur dann Ereignisse, wenn sich Änderungen ihrer beobachteten Umwelt zugetragen haben.

- *Disjunktion* ($E_1 \ \vee \ E_2$): Ein komplexes Ereignismuster wird detektiert, wenn E_1 oder E_2 auftreten.

  Ein komplexes Ereignis E_c wird bspw. immer dann detektiert, wenn die Temperatur über 30 °C oder die Luftfeuchtigkeit des selben Raums über 90% beträgt. Es ergibt sich die Spezifikation $\{(E_1, E_2) \mid [E_1.temp > 30\ °\text{C}] \ \vee \ [E_2.hum > 90\%] \ \vee \ [E_1.room = E_2.room]\}$.

- *Konjunktion* ($E_1 \ \wedge \ E_2$): Ein komplexes Ereignismuster wird detektiert, wenn E_1 und E_2 auftreten.

  Ein komplexes Ereignis E_c wird immer dann detektiert, wenn die Temperatur über 30 °C und die Luftfeuchtigkeit des selben Raums über 90% beträgt. Es ergibt sich die Spezifikation $\{(E_1, E_2) \mid [E_1.temp > 30\ °\text{C}] \wedge [E_2.hum > 90\%] \ \wedge \ [E_1.room = E_2.room]\}$.

- *Sequenz* ($E_1 \ ; \ E_2$): Ein komplexes Ereignismuster wird detektiert, wenn zunächst E_1 und danach E_2 auftritt.

  Als Beispiel wird angenommen, dass ein komplexes Ereignis E_c einen Einbruch als Folge eines Einbruchsmeldeereignisses und einer Bewegung im

selben Raum detektiert. Das Ereignis E_c lässt sich mit der Spezifikation $\{(E_3, E_4) \mid [E_4.destroyed = true \ ; \ E_3.movement = true] \ \wedge \ [E_3.room = E_4.room]\}$ definieren.

- *Beliebig* (any$(m, E_1, E_2, \ldots, E_n)$): Ein komplexes Ereignismuster wird detektiert, wenn mindestens m der vorgegebenen n Ereignisse auftritt.

  Um Fehlalarme von Einbrüchen bei der Polizei, bspw. durch Sturmschäden, zu vermeiden, wird bei einem Einbruchsverdacht zunächst ein Wachmann zum Gebäude gerufen. Dieser soll bei einem detektierten Einbruch wie aus dem vorherigen Beispiel, aber auch bei nächtlichen (zwischen 22 Uhr und 5 Uhr) Bewegungen innerhalb des Gebäudes gerufen werden. Ein komplexes Ereignis E_c wird in diesem Fall detektiert, wenn die folgende Spezifikation erfüllt ist: $\{any(1, (((E_3, E_4) \mid [E_4.destroyed = true; E_3.movement = true] \ \wedge \ [E_3.room = E_4.room]), ((E_3)|[E_3.movement = true] \wedge [22:00 < E_3.time < 5:00])))\}$.

Die ersten drei Operatoren sind binäre Operatoren, ohne dass sie jedoch explizit ein zeitliches Intervall t_i definieren. Der Beliebig-Operator ist ein zusammengesetzter Operator, da er aus bereits bestehenden Operatoren zusammengesetzt wird. Alle vier aufgeführten Operatoren benötigen aus ihrer Spezifikation heraus keine explizite Beschreibung eines Zeitintervalls, aus praktischen Gründen wird dies jedoch benötigt, ansonsten warten *Disjunktion*, *Konjunktion*, *Sequenz* und *Beliebig* möglicherweise unendlich lang bis ein komplexes Ereignis erkannt wird. Dies ist im praktischen Einsatz nicht durchführbar, sodass eine zusätzliche temporale Beschränkung notwendig ist. Die im Folgenden aufgeführten Operatoren benötigen dagegen explizit ein zeitliches Intervall und temporale Bedingungen, wie es bereits von Chakravarthy et al. [35] vorgesehen wurde. So unterscheiden die Autoren Ereignisse, die gleichzeitig überlappend oder in Konjunktion zueinander stehen. Die nun folgenden Operatoren basieren im Wesentlichen auf der Arbeit von Gatziu et al. [77]:

- $E_1^*(t_i)$: Das komplexe Ereignis wird nur einmal detektiert, auch wenn Ereignisse vom Typ E_1 mehr als einmal innerhalb eines bestimmten Zeitfensters t_i auftreten.

  Im Falle, dass ein Temperatursensor defekt ist, liefert er gemäß seiner Spezifikation den Wert -99 °C. Damit der defekte Sensor bis zu seinem Austausch keine weiteren Berechnungen wie die der Durchschnittstemperatur verfälscht, darf der Fehlerwert -99 °C nur einmal innerhalb einer halben Stunde deketektiert werden. Solch eine Spezifikation lautet $\{[E_1.temp = -99 \ °C]^*(30 \ min)\}$

- *Mindestens* ($\times(n, E_1, t_i)$): Das komplexe Ereignis wird genau einmal detektiert, wenn das Ereignis mindestens n-mal innerhalb eines Zeitintervalls t_i auftritt.

Um zu beurteilen, ob ein Temperatursensor wirklich defekt ist oder nur zufällig falsche Werte liefert, muss der Fehlerwert von −99 °C mindestens fünfmal innerhalb von einer Minute gemessen werden. Die dazugehörige Spezifikation eines komplexen Ereignismusters E_c lautet $\{\times, [E_1.temp = -99\ °\text{C}], (1\ min)\}$

- *Nicht* ($\neg(E_1, t_i)$): Das komplexe Ereignis wird detektiert, wenn das Ereignis E_1 innerhalb eines Zeitraums t_i nicht aufgetreten ist.

  Im Beispiel ist vorgesehen, dass ein Luftfeuchtigkeitssensor auch dann regelmäßig Ereignisse erzeugt, wenn sich keine messbaren Änderungen zugetragen haben. Damit wird sichergestellt, dass der Sensor noch intakt ist. Wenn ein Sensor nicht mindestens alle drei Minuten ein Ereignis erzeugt, wird ein komplexes Ereignis E_c mit der Spezifikation $\neg(E_2, (3\ min))$ detektiert.

Die gezeigten Operatoren stammen zwar aus dem Umfeld der aktiven Datenbanken, werden aber auch im Bereich der verteilten Detektion von Ereignismustern eingesetzt. An dieser Stelle ist anzumerken, dass die Operatoren nicht präzise definiert und ihre konkreten Umsetzungen implementierungsabhängig sind.

Mit den vorgestellten Operatoren lassen sich komplexe Ereignisse spezifizieren, ihre Bedeutung hinsichtlich der Auswahl der einzelnen Ereignisse gleichen Typs bleibt jedoch unspezifisch. Beim Betrachten der Spezifikation $(E_1 \wedge E_2)$ wird deutlich, dass sobald eine neue Instanz von E_1 detektiert wird, alle Kombinationen aus verfügbaren E_1 mit sämtlichen Ereignisinstanzen von E_2, die sich in der Ereignishistorie befinden, als komplexe Ereignismuster detektiert werden. Da in diesem Fall alle Kombinationen möglich sind, wird diese Art der Mustererzeugung als *unbegrenzter Kontext* (engl. *unbounded context*) beschrieben [35]. Mit diesem Vorgehen wird die Anzahl von (möglichen) komplexen Ereignissen sehr groß. Hinzu kommt, dass in vielen Fällen nicht alle detektierten Ereignisse sinnvoll im Sinne der sie subskribierenden Anwendung sind. Um die Detektion komplexer Ereignismuster genauer spezifizieren zu können, wird zwischen den Initiator-, Detektor- und Terminatorereignissen sowie Teilnehmern unterschieden. Mit dem Initiatorereignis beginnt die Detektion und endet erfolgreich mit dem Detektorereignis. Bei einem Terminatorereignis bricht die Detektion hingegen ab. Teilnahmeereignisse sind hingegen lediglich ein Teil des Ereignismusters ohne Steuerungsfunktionalität. Für die Formulierung von Restriktionen bezüglich der Auswahl der Ereignismenge, die Ereignismuster bilden, dienen die *Konsumptionsmodi* (engl. *consumption modes*). Gängige Konsumptionsmodi sind *neueste* (engl. *recent*), *chronologisch* (engl. *chronicle*), *kontinuierlich* (engl. *continuous*) sowie *kumulativ* (engl. *cumulative*) [35]. Die jeweiligen Modi unterscheiden sich darin, welche Ereignisse die Detektion initiieren bzw. terminieren, wie und in welcher Reihenfolge Ereignisse zu Mustern zusammengefügt werden und welche Ereignisse für zukünftige Ergebniskompositionen zur Verfügung stehen. Im *neueste*-Modus initiieren nur die letzten Ereignisinstanzen die Muster-

erkennung, noch laufende Detektionen werden beim Auftauchen einer neueren Ereignisinstanz abgebrochen und ältere Ereignisinstanzen als Initiatoren verworfen. Beim *chronologischen*-Modus startet immer das älteste, noch nicht verbrauchte Ereignis die Mustererkennung. Erst nach erfolgreicher Detektion wird das Initiatorereignis verworfen und steht für weitere Detektionen nicht mehr zur Verfügung. Beim *kontinuierlichen*-Modus startet jede Ereignisinstanz die Detektion des Musters. Jede Instanz kann nur eine Detektion starten und steht danach als Initiator nicht mehr zur Verfügung. Auch beim *kumulativen* Modus ermöglichen die Ereignisinstanzen nur eine Mustererkennung. Bei dieser Kombinationsart werden mehrfache Instanzen eines Ereignistyps solange zusammengefasst, bis die Musterdetektion erfolgreich beendet oder abgebrochen wurde.

Ein gegenüber den Konsumptionsmodi anderes Konzept sind *Fenster* (engl. *windows*), welche anhand von Zeit oder der Anzahl von Ereignissen definiert werden. So kann ein Fenster bspw. alle Ereignisse innerhalb der letzen 10 Minuten nach Eintritt eines bestimmten Ereignisses oder die letzten 100 Ereignisse beinhalten. Die Konzepte unterscheiden sich vor allem darin, dass Konsumptionsmodi üblicherweise für ein gesamtes Muster, Fenster dagegen für den jeweiligen Operator angewendet werden. Die in dieser Arbeit als Beispiele genutzten Ereignismuster nutzen ab jetzt durchgehend den Konsumptionsmodus *recent*, womit stets nur die neuesten Ereignisinstanzen ein komplexes Ereignismuster formen.

Detektordeployment

Das Deployment von Detektoren erfolgt im Anwendungsbeispiel in einem Anwendungscontainer als Ausführungsumgebung, die von der Middleware REBECA bereitgestellt wird. Der Anwendungscontainer erhält wahlweise kompilierten und ausführbaren Anwendungscode bzw. serialisierte Objekte. Aufgrund der einheitlichen Middleware können Code bzw. Objekte unter Einbindung der Publish/Subscribe-Eigenschaften an anderen Stellen der Middleware ausgeführt werden. Ein händisches Nachinstallieren benötigter Unterstützungskomponenten entfällt, gleichzeitig unterstützt die Middleware unterschiedlich leistungsfähige Geräte und abstrahiert auf diese Weise von der zugrunde liegenden Hardware.

Doch auch wenn technische Aspekte wegabstrahiert wurden, bleibt die Frage, wie Detektoren den jeweiligen Middleware-Instanzen zur Ausführung zugeordnet werden. Für die Zuweisung der Anwendungen zu ihren Ausführungsorten existieren prinzipiell zwei Vorgehensweisen. Zum einen, die initiale Platzierung in der Nähe des Nutzers bzw. des Nutzergerätes. Dazu wird der Detektor auf einem dem Nutzer nächsten Gerät deployt, allerdings müssen alle Notifikationen, auch wenn sie nicht am späteren Muster teilnehmen, durch das Brokernetzwerk geleitet werden. Dies kann zu Überlastsituationen im Netzwerk oder an den Geräten führen. Eine andere Möglichkeit ist ein initiales Deployment, welches von vornherein versucht, die Platzierung möglichst vorteilhaft im Sinne des Platzierungsziels vorzunehmen. Der in der Arbeit vorgestellte Ansatz zur initialen Verteilung (siehe Kapitel 3) agiert auf der Grundlage des Austauschs von Noti-

fikationen, deren Inhalt die spezifizierten Anforderungen von Anwendungen und die verfügbaren Ressourcen der Ausführungsorte sind. Natürlich verursacht der Informationsaustausch von Anforderungen und verfügbaren Ressourcen Kommunikationsaufwand, welcher jedoch durch die möglichen zu erzielenden Einsparungen ausgeglichen werden kann. Mit der Problematik der initialen Platzierung von Detektoren für komplexe Ereignismuster haben sich bereits einige Arbeiten befasst. Die verteilte Detektion von komplexen Ereignismustern ist Thema vieler Veröffentlichungen, wobei nach der Betrachtung von Platzierungsansätzen für statische Umgebungen auch die Platzierung in dynamischen Umgebungen zunehmend an Bedeutung gewinnt. Um auf die Dynamik reagieren zu können, wurden Mechanismen entwickelt und vorgestellt. Die sehr oft verfolgte Idee war und ist es, Detektoren bzw. Detektorkomponenten in Abhängigkeit eines gegebenen Ziels zu verschieben/zu migrieren. Doch auch wenn diese Anpassung zur Laufzeit des Systems möglich ist, ist eine lange Kette einzelner Migrationen der Gesamteffizienz abträglich, bedeutet doch jede Migration einen gewissen Aufwand und eine vorübergehende Nicht-Verarbeitung des gewünschten Ereignismusters solange eine Komponente migriert wird.

Eine Idee, die initiale Platzierung von Komponenten so durchzuführen, dass eine unnötige Anzahl von Migrationen vermieden wird, findet sich u.a. in [165]. Hier wird zunächst ein Verfahren zur initialen Platzierung von Detektoren mit Hilfe von Rendezvous-Brokern vorgestellt. Rendezvous-Broker sind sozusagen Kontaktpunkte für Ereignisse eines bestimmten Typs. Sie werden bspw. mit Hilfe eines Hash-Wertes ausgesucht und übernehmen eine Vermittlerrolle zwischen Subskribenten und Publishern. Dabei werden Subskriptionen für einen bestimmten Ereignistyp an den zuständigen Rendezvous-Broker geleitet, dieser erhält außerdem die Notifikationen der Ereignisproduzenten. Neben der Anwendung für primitive Ereignisse ist das Prinzip der Rendezvous-Broker auch für komplexe Ereignismuster nutzbar. Eine sinnvolle Verteilung bzw. Anwendung der Rendezvous-Broker vorausgesetzt, kann die bestehende initiale Platzierung als Ausgangspunkt für weitere Migrationsschritte mittels lokalem Wissen dienen. Die initiale Platzierung der Rendezvous-Broker erfolgt anhand eines Kostenraums, auf welchen während der Beschreibung bekannter Ansätze für die dynamische Platzierung noch eingegangen wird. Als weitere Möglichkeit beschreibt O'Keeffe [165] die initiale Verteilung von Detektoren in einem virtuellen Kostenraum. Die Berechnung der Position des Detektors erfolgt durch eine zentrale Komponente. Der Ansatz versucht über Look-Up-Nachrichten an die jeweiligen Rendezvous-Broker herauszufinden, ob das gewünschte Ereignismuster bereits im Netzwerk existiert. Falls ja, wird eine Verbindung anhand der von der Look-Up-Nachricht aufgebauten Pfade zwischen dem Rendezvous-Broker des anfragenden und dem Rendezvous-Broker des bereits vorhandenen Detektors hergestellt. Ist dies beim ersten Versuch nicht möglich, wird die Suche nach Teildetektoren entsprechend wiederholt. Ist die Suche nach einem Detektor nicht erfolgreich, wird mit Hilfe globalen Wissens durch eine zentrale Komponente die Position des Detektors in einem virtuellen Kostenraum bestimmt und auf das tatsächlich vorhandene Netzwerk abgebildet sowie der Detektor

an dieser Stelle deployt. Parallelen in der Arbeit von O'Keeffe [165] und dem in dieser Arbeit vorgestellten Ansatz zur initialen Platzierung sind vorhanden, allerdings verzichtet der hier vorgestellte Ansatz auf die vorgeschlagene zentralisierte Positionierung [165], stattdessen wird die Abfolge vieler Migrationsschritte in Kauf genommen, anstatt Informationen zentral zusammenzuführen und auszuwerten. Zudem wird in dieser Arbeit auf eine frühzeitige, optimale Zuordnung während der initialen Platzierung verzichtet, da für eine frühzeitige Optimierung Kommunikations- und Auswertungsressourcen gebunden werden. Eine mit dann hohem Aufwand gefundene Lösung wäre jedoch durch die inhärente Dynamik des Systems frühzeitig veraltet und damit nicht gerechtfertigt. Weiterhin ist das Aufkommen von Nachrichten zu begrenzen, um darüber hinaus eine zügige Bereitstellung der benötigten Dienste zu ermöglichen. So wird zunächst nach einer gültigen, aber nicht zwangsläufig besten, sprich verbesserungsfähigen Lösung gesucht und anschließend optimiert.

Die Detektoren werden innerhalb der Publish/Subscribe-Broker ausgeführt. Innerhalb der Broker existiert ein Anwendungscontainer, der als eine Ausführungsumgebung fungiert und die notwendigen Services wie das Auflösen von Abhängigkeiten und das Nachladen von Bibliotheken bereitstellt. Zusätzlich besteht eine enge Verzahnung zwischen der Anwendungsausführung und der Publish/Subscribe-Middleware. Die Integration der Detektoren erfolgt innerhalb der Middleware Rebeca. Sie bietet vielfältige Funktionalitäten und Erweiterungsmöglichkeiten an, welche in Abschnitt 4 ausführlich beschrieben werden. Der Anwendungscontainer nimmt die Java-Objekte auf und spiegelt mit seinen zur Verfügung gestellten Ressourcen die Fähigkeiten der Hardware wider, auf der er ausgeführt wird. Je nach Leistungsfähigkeit und Hardwareanforderungen der Anwendungen ist der Container in der Lage, keine, eine oder mehrere Anwendungen auszuführen. Zusätzlich kennt der Container die lokal verfügbaren Ressourcen, bspw. die Anbindung eines bestimmten Sensors.

Das Deployment besteht aus dem Instanziieren der Anwendungsklassen aus übersetztem Java-Code, der Auflösung von Abhängigkeiten und ggf. dem Nachladen von benötigten Bibliotheken. Anforderungen bezüglich benötigter Ereignisse werden an den Broker der Middleware gestellt, welcher diese in Form von Subskriptionen weiterleitet. Analog dazu werden auch Ankündigungen und spätere Notifikationen durch den Broker in das Publish/Subscribe-Netzwerk eingespeist.

Technische Realisierung der Detektoren

Neben den verschiedenen angesprochenen Beschreibungssprachen wie reguläre Ausdrücke [80] und endliche Zustandsautomaten [178] existieren unterschiedliche, technische Verfahren zur Detektion. Pietzuch et al. [178] zeigen, wie endliche Automaten kombiniert mit einem Zeitmodell und Parametrisierungsmöglichkeiten zur Detektion eingesetzt werden können. Andere Ansätze, wie der von Ghanem et al. [81], nutzen Petri-Netze zur Darstellung und Detektion komplexer Ereignismuster. Deren Arbeit basiert auf den Vorarbeiten von Castel et al.

[34] zur komplexen Ereignisdetektion in Videos. Besonders hervorzuheben sind die in Castel et al. [34] genannten Eigenschaften von Petri-Netzen, deterministisches und stochastisches Auftreten von Ereignissen modellieren zu können. Damit eignen sie sich für die präzise mathematische Modellierung, um die Synchronisierung und die sequenzielle Abfolge von Ereignissen darstellen zu können. Ebenfalls mit Hilfe von Petri-Netzen lässt sich die Historie von Zuständen abbilden. Durch den zusätzlichen Einsatz des RETE-Algorithmus kann zudem die Detektionsgeschwindigkeit von Ereignismustern [28, 70] im Vergleich zu den Ansätzen mit endlichen Zustandsautomaten und Petri-Netzen verbessert werden. Auch der von Gatziu et al. vorgestellte Ansatz SAMOS [79] gründet auf einem (gefärbten) Petri-Netz. Aus dem Forschungsbereich der objektorientierten Datenbanken stammt der von Gehani et al. vorgestellte Ansatz Ode [80]. In diesem werden komplexe Ereignismuster mittels einer Anfragesprache, die ähnlich zu regulären Ausdrücken ist, spezifiziert. Die interne Umsetzung der Detektion der Ereignismuster erfolgt dagegen mit Hilfe von endlichen Zustandsautomaten. Verbreitet sind auch baumbasierte Detektionstechniken, welche ebenfalls unter der Bezeichnung *Matching Trees* bekannt sind. Diese Technik sieht vor, Teilbeschreibungen der komplexen Ereignismuster abzuspalten und separat zu erkennen. Die Blätter des Baums repräsentieren die primitiven Ereignisse, während die Elternknoten die komplexen Muster, also die Verbindungen der jeweiligen Kindknoten mit den entsprechenden Anforderungen darstellen. Die Verbindung der entstandenen Teilereignismuster mittels Matching Trees nutzen sowohl SNOOP [37], als auch GEM [140] und das in [114] vorgestellte GenTrie. Beim graphenbasierten Ansatz wird ein komplexes Ereignismuster mit Hilfe eines gerichteten, azyklischen Graphen in Teilkomponenten zerlegt, wobei Knoten die jeweiligen Ereignisse und Kanten die Ereigniskomposition darstellen [102].

Der technische Aspekt der Detektorrealisierung steht in dieser Arbeit nicht im Vordergrund. Es wird angenommen, dass die Detektoren selbst in der Lage sind, sich anhand der durch die Algebra vorgegebenen Sollbruchstellen in Teildetektoren aufzuspalten. Die Aufspaltung und Zusammenführung erfolgt selbstorganisiert und mit den durch die Spezifikation geforderten und vom Anwendungsentwickler umgesetzten Mechanismen. Die Detektoren, welche als Anwendungen implementiert sind, setzen intern die Basisoperationen Dekomposition, Rekombination und die notwendigen Tätigkeiten im Zuge der Migration und Replikation um. Dazu gehören auch die Aufteilung der Daten- und sonstigen Zustände.

5.2.2 Probleme bisheriger Ansätze

Ansätze, die in Publish/Subscribe-basierten Kommunikationsumgebungen einerseits die automatische Anwendungsverteilung von Anwendungen und andererseits auch ihre weitere, adaptive Verteilung betrachten, sind bisher nicht publiziert worden. Es existieren zwar allgemein formulierte Ansätze, bei denen Komponenten migriert werden können, allerdings geschieht dies außerhalb der Publish/Subscribe-Umgebung und vor allem ohne die Möglichkeit, in Richtung einer

spezifizierten Optimierungsfunktion (Lastausgleich, Ressourceneffizienz etc.) zu verschieben. Allgemeine Ansätze zur Migration von Komponenten sind jedoch bekannt und werden auch bspw. innerhalb von Java unterstützt. Dazu gehören die bereits in *JEE 6.0* integrierten Techniken und solche Ansätze, die Java-basierte Anwendungen oder Prozesse migrieren können [24, 49, 145].

Automatisches Deployment von Anwendungen und Detektoren

Aus dem Anwendungsbereich der Detektion komplexer Ereignismuster in heterogenen Sensornetzen stammt der Ansatz von Kamiya et al. [120]. Die Sensornetze bestehen aus heterogenen Teilnetzen, die durch Gateway-Knoten miteinander verbunden sind. Diese sind auch für die Verwaltung von Subskriptionen in ihrem Teilnetz verantwortlich, besitzen also globales Wissen über einen lokal abgeschlossenen Bereich. So werden die von den Klienten abgegebenen Subskriptionen zunächst an den nächstgelegenen Gateway-Knoten geleitet, dieser analysiert die Subskription und leitet die gesamte Anfrage bzw. Teilanfragen an die Gateway-Knoten weiter, welche die jeweiligen Teilanfragen bearbeiten können. Jeder Gateway-Knoten verwaltet in dem ihm zugehörigen Teilnetz alle Ereignisdetektionsbäume. Ziel ist es, neu ankommende Subskriptionen durch bereits vorhandene Teilsubskriptionen auszuführen und so vorhandene Teilmuster für neue Mustererkennungen wiederzuverwenden. Besonderes Augenmerk legen die Autoren des Papiers zwar auf die initiale Platzierung der Detektoren für komplexe Ereignismuster, allerdings zeigen sie nicht, wie sich das System auf sich ändernde Umgebungen adaptiert.

Eine Mischung aus statischen und dynamischen Platzierungsmethoden für Detektoren in verteilten, ereignisorientierten Umgebungen präsentiert bspw. der Autor in [165], wobei an dieser Stelle zunächst die statischen, und somit für die initialen Verteilungen sinnvollen Methoden betrachtet werden. Ausgangspunkt ist ein Ansatz, der basierend auf einem Overlaynetzwerk und inhaltsbasiertem Routing einen Kostenraum aufspannt, in welchem anschließend Detektoren platziert werden. Dabei wird hier ein Verfahren für die Platzierung von Detektoren für komplexe Ereignismuster in *Hermes* [131, 176] realisiert. Hermes ist, wie auch REBECA, eine ereignisbasierte Middleware, bei der die Kommunikationspartner mittels Publish/Subscribe miteinander interagieren. Ebenso wie REBECA ist auch Hermes für die Kommunikation und Interaktion von Partnern in verteilten Umgebungen geeignet. Wie auch REBECA nutzt Hermes ein Overlay-Netzwerk. Der Einsatz eines Overlays bietet bspw. den Vorteil, dass Fehlertoleranzmechanismen auf Ebene des Overlays transparent eingesetzt werden können, um das logische Brokernetz zu stabilisieren und den Verbindungsauf- und -abbau zwischen den Brokern und dem Brokernetz zu vereinfachen. Hermes bedient sich zudem der Ereignisverteilungsbaumstrukturen [176], die mittels Rendezvous-Brokern aufgebaut werden. Womit auch schon ein bedeutender Unterschied zwischen Hermes und REBECA genannt ist, nämlich das explizite Zurückgreifen auf Rendezvous-Broker.

Vor der initialen Platzierung eines Detektors wird das komplexe Ereignismuster zunächst vorverarbeitet, und zwar indem aus diesem ein Anfragegraph erstellt wird. An dieser Stelle sind die Parallelen zu strombasierten Ansätzen nicht von der Hand zu weisen. Diese einzelnen (inneren) Knoten, welche die Detektoren widerspiegeln, werden anschließend im Brokernetzwerk verteilt, d.h auf Broker der Kommunikationsinfrastruktur abgebildet. Diese initiale Abbildung von Detektoren auf Broker ist auf unterschiedliche Arten möglich, erstens durch Platzierung von Detektoren auf Rendezvous-Brokern. Anschließend kann mit einzelnen Migrationsschritten die Platzierung angepasst werden. Dieser Ansatz ist jedoch u.U. problematisch, da die Anpassung der Platzierung im Verhältnis zu lokalen Anpassungen lange dauert und somit wieder Ressourcen verbraucht werden, bis eine vorteilhafte oder gar optimale Platzierung gefunden wird. Beim zweiten Ansatz werden mit Hilfe gesammelter Daten über Publisher primitiver Ereignisse und bereits existierender Detektoren Platzierungsentscheidungen zentral getroffen. In diesem Fall ist die zentrale Entscheidung wiederum problematisch.

Beim ersten Ansatz werden Rendezvous-Broker für die jeweiligen Detektoren benötigt. Ausgehend von diesen Rendezvous-Brokern erfolgt die graduelle Anpassung der Detektorplatzierung, welche auf der Grundlage von beobachteten, tatsächlichen Datenströmen und der Nähe zu benachbarten Detektoren des Anfragegraphen besteht. Bei einer rein zufälligen initialen Platzierung der Rendezvous-Broker kann die Platzierungsanpassung einige Zeit in Anspruch nehmen.

Der zweite durch O'Keeffe [165] vorgestellte Ansatz implementiert dagegen einen verbesserten Platzierungsalgorithmus, welcher auf einem virtuellen Latenzraum mit virtuellen Koordinaten der Broker im Overlaynetzwerk basiert. Die Koordinaten der Broker innerhalb des Latenzraums werden dabei mittels eines zentralisierten Algorithmus auf der Grundlage der Relaxierung von Federn (engl *spring relaxation*) durchgeführt. Die Berechnung der Positionen von Detektoren im Latenzraum wird nach dem *Vivaldi*-Algorithmus [48] durchgeführt. Vivaldi ist ein skalierbarer verteilter Algorithmus auf der Grundlage von *Spring Relaxation*. Die Kraft bzw. Spannung der Federn basiert dabei auf dem Potential zwischen der tatsächlichen Latenz und der durch den Latenzraum vorhergesagten Latenz. Die Position im Latenzraum bestimmt dabei jeder Knoten selbst, indem er periodisch den Abstand zu benachbarten oder anderen ausgewählten Brokern ermittelt. Diese Erkenntnisse werden anschließend mit den vorhergesagten Abständen verglichen. Dieser Prozess wird iterativ wiederholt, bis die tatsächliche und vorhergesagte Position möglichst nahe beieinander liegen und eine minimale Spannung der Federn vorliegt. Dies minimiert gleichzeitig die Vorhersagefehler soweit wie möglich. Durch den Vivaldi-Algorithmus und eine eindeutige Identifizierung besitzt jeder Broker eine virtuelle Koordinate im Latenzraum. Virtuelle Koordinaten ermöglichen einem Broker außerdem die Bestimmung der Latenz zwischen ihm und jedem anderen Punkt im Latenzraum.

Auch bei diesem zweiten Verfahren zur initialen Platzierung von Detektoren werden diese zunächst in Anfragegraphen (gerichteter, azyklischer Graph mit Quellblattknoten, Zielblattknoten und inneren Knoten als Teildetektoren) um-

geformt, wobei die Quellblattknoten die Broker der primitiven Ereignisquellen und die Zielblattknoten die Broker der Subskriptionen widerspiegeln. Bis hierher ist der Ausgangsschritt der initialen Platzierung gleich dem ersten von O'Keeffe [165] vorgestellten initialen Platzierungsverfahren. Bei der Platzierung von Detektoren werden zunächst die Koordinaten von primitiven Ereignisproduzenten und vorhandenen Detektoren in einem System bestimmt, dies erfolgt durch Abbildung der Knoten des internen Anfragegraphen auf die vorhandenen Broker. Daraufhin erfolgt die Bestimmung von Koordinaten für die noch zu verteilenden Detektoren, deren Berechnung zentral erfolgt. Die Platzierung innerhalb des Systems erfolgt in einem virtuellen Raum, der durch Latenzen zwischen den jeweiligen Brokern bestimmt wird. Doch zunächst wird nach bereits vorhandenen Detektoren durch eine zentrale Koordinierungsstelle gesucht, dies geschieht mit Hilfe eines Look-Ups zum entsprechenden Rendezvous-Broker des obersten komplexen Detektors bzw. dessen Repräsentation im Anfragegraphen. Über den Rendezvous-Broker wird der Look-Up zum Detektor weitergeleitet. Existiert dieser, muss nicht nach weiteren Unter- bzw. Teildetektoren gesucht werden. Durch den Look-Up wird ein Advertisement auf dem Rückweg zum Rendezvous-Broker erzeugt und an den Subscriber geleitet, womit die Platzierung beendet ist. Wird hingegen kein Advertisement erzeugt, existiert kein Detektor und es muss mittels weiterer Look-Ups nach Unterdetektoren, d.h. nach Teilereignismustern des Detektors gesucht werden. In diesem Fall erhält der Subscriber des komplexen Ereignismusters eine entsprechende Nachricht über den fehlenden Detektor, die Suche nach vorhandenen (Teil-) Detektoren beginnt von neuem, diesmal jedoch auf einer tieferen Ereignismusterebene. Dazu werden wieder Look Up-Nachrichten erzeugt, welche wiederum zu den jeweiligen Rendezvous-Brokern und von dort zu möglichen Detektoren weitergeleitet werden. Falls erneut keine Detektoren gefunden werden, wird das Prinzip der Look-Ups bis zur Ebene der primitiven Ereignisquellen weitergeführt. Schließlich wird dem zentralen Koordinator entweder die Adresse des Detektors bzw. der primitiven Ereignisquelle im virtuellen Latenzraum weitergeleitet oder die Adresse des Rendezvous-Brokers selbst. Während ersteres bei einmaligem Vorhandensein von Detektoren/primitiven Ereignisquellen sinnvoll ist, ist die zweite Variante vor allem dann interessant, sobald mehrere gleichartige Detektoren oder Ereignisquellen existieren.

Die zentrale Koordinierungsstelle erhält damit die Koordinaten der vorhandenen Detektoren bzw. der primitiven Ereignisquellen. Für alle noch nicht platzierten Detektoren ermittelt nun die zentrale Koordinierungsstelle die noch benötigten Positionen. Dazu wird auf einen zentralen Algorithmus zur Federkraftrelaxierung zurückgegriffen. Hier werden zunächst die Positionen der Detektoren im virtuellen Latenzraum berechnet. Ist eine Position im virtuellen Raum gefunden, erfolgt die Abbildung der Position auf das tatsächliche Netzwerk. Dazu werden laut O'Keeffe [165] Techniken wie Raumfüllkurven [175] mittels Relaxierung bzw. das „Scaled-θ-Routing" [89, 131] genutzt. Dazu verweist O'Keeffe vor allem auf Arbeiten, die im Umfeld von Hermes erstellt wurden. Nachdem die Detektoren platziert und deployt wurden, werden entsprechende Subskriptionen

und Advertisements über bzw. an die jeweiligen Rendezvous-Broker abgegeben.

Während das erste Platzierungsverfahren durch die zufällige Verteilung von Rendezvous-Brokern und damit möglicherweise weit weg von den eigentlichen Detektoren enorme Nachjustierungen der Detektorplatzierungen nach sich ziehen kann, ist die mehrstufige, initiale Platzierung von Detektoren bezüglich des Nachjustierungsaufwands effizienter gelöst. Außerdem wird eine unnötige Doppelplatzierung von Detektoren verhindert. Allerdings ist der vorgestellte Ansatz anfällig gegenüber den Problemen des rendezvousbasierten Routings, hängt doch die Effizienz der initialen Platzierung maßgeblich von der Verteilung der Rendezvous-Broker und vor allem auch von deren Granularität ab, da Detektoren für bestimmte Ereignismuster spezifiziert und damit auf einen Ereignistyp festgelegt sind, aber andere, wie bspw. räumliche Aspekte, außen vor lassen.

Automatische Platzierungsanpassung von Anwendungen und Detektoren

Gegenüber der Platzierung von Detektoren mittels der gegenseitigen Abbildung von virtuellen Räumen auf reale Netzwerke und der Nutzung von Rendezvous-Brokern existieren weitere Konzepte, welche die Verteilung einzelner Dienste ohne die beiden genannten Techniken mittels Spring Relaxation ermöglichen. So beschreibt Herrmann [97] die Verteilung einzelner Dienste, betrachtet aber das Deployment ganzer Anwendungen nicht. Er liefert jedoch die Grundidee, wie Dienste innerhalb eines Gerätenetzwerks migriert werden können. Werden Anwendungen als mobile Klienten betrachtet, kommen weitere Veröffentlichungen zur Auswahl verwandter Arbeiten hinzu. So beschreiben Zeidler und Fiege [213] bspw. wie sich mobile Klienten in einem Publish/Subscribe-basierten Umfeld inklusive der Anpassung ihrer Abhängigkeiten migrieren lassen. Doch ähnlich wie bei Herrmann [97] findet eine Betrachtung des Deployments von Komponenten nicht statt.

Strombasierte Verarbeitung. Ähnlich zur Arbeit von O'Keeffe [165] zur Verteilung von Detektoren in einem Publish/Subscribe-basierten Umfeld präsentieren Pietzuch et al. [178] Platzierungsstrategien für Detektoren komplexer Ereignismuster im Umfeld strombasierter Verarbeitung. Dabei nutzen Pietzuch et al. ebenfalls eine Schichtenarchitektur bestehend aus den drei folgenden Schichten: Netzwerkschicht, Zwischenschicht und Anfragedefinitionsschicht. Außerdem verfolgt der vorgestellte Ansatz, wie der von O'Keeffe [165], eine Mischung zwischen dezentralisierter Bestimmung von Detektorplatzierungen und zentralisierter initialer Detektorplatzierung. Alle Netzwerkknoten werden in einen Kostenraum abgebildet, welcher der Zwischenschicht bekannt ist. In dieser Zwischenschicht werden die optimalen Platzierungen der Detektoren ermittelt und die Positionen zurück auf die physische Topologie abgebildet und die Detektoren dort deployt. Anschließend wird die Platzierung mittels lokalem Wissen verbessert. Zu häufigen Umplatzierungen von Detektoren wird jedoch entgegengewirkt, damit keine Oszillationen entstehen. Dies geschieht, wie auch bei dem in der Arbeit

vorgestellten Ansatz, durch die Definition eines Schwellwertes, der das Minimum einer zu erreichenden Einsparung darstellt. Darüber hinaus betonen die Autoren des Papiers, dass die Idee des virtuellen Kostenraums neben der Bestimmung eines optimalen Ausführungsortes für Detektoren auch zum leichteren Wiederfinden von bereits deployten Detektoren geeignet ist. Prinzipiell sind sich die Ansätze von O'Keeffe [165] und Pietzuch et al. [178] ähnlich, ermöglichen doch beide optimale Platzierungsentscheidungen mit Hilfe von Ankündigungen und durch die Berechnung einer Platzierung in einem virtuellen Kosten- bzw. Latenzraum. Dieses Vorgehen wird in Kombination zentral und dezentral ausgeführt. Der in der Arbeit vorgestellte Ansatz dagegen legt weniger Wert auf eine optimale initiale Platzierung, sondern verlangt als Ausgangspunkt lediglich eine gültige, initiale Platzierung für die folgende, schrittweise Verbesserung. Auf diese Weise werden das notwendige Vorhalten einer Sammelstelle für globales Wissen und die gegenseitige Transformation von Informationen aus dem realen Netzwerk in den virtuellen Kostenraum eingespart. Trotzdem wird die Möglichkeit erhalten, eine vorteilhafte Platzierung auf der alleinigen Grundlage lokalen Wissens zu erreichen, globales Wissen und global gültige Auswertungen werden nicht benötigt.

Eine weitere Idee für die Platzierung von Detektoren bieten Ahmad und Cetintemel [2]. Die Autoren setzen hier auf eine Heuristik auf der Grundlage eines paarweisen Vergleichs von Latenzen benachbarter Broker. Sie erweitern diese Idee um „gut platzierte" Broker, welche im ursprünglichen Informationsverarbeitungsprozess nicht vorkommen. Im Grunde arbeitet der Ansatz auf der Grundlage eines Overlay-Netzes, Anfragen sind baumartig strukturiert, Broker sind physisch über TCP verbunden, logisch sind sie jedoch in verteilten Hash-Tabellen (*DHTs*) organisiert. Die Routingtabellen der Overlayknoten enthalten neben den Weiterleitungsinformationen auch Statusinformationen über die Netzwerkverbindungen. Zusätzlich existiert zu jedem Abarbeitungsbaum ein korrespondierender Kontrollflussbaum. Dies verbessert laut den Autoren die Skalierbarkeit des jeweiligen Abarbeitungsbaums. Der Abarbeitungsbaum ist in Zonen unterteilt und zu jeder Zone existiert ein Koordinator. Dieser ist für die korrekte Abarbeitung der Aufgabenschritte innerhalb der Zone verantwortlich. Während die Knoten physisch mittels TCP verbunden sind, erfolgt die logische Adressierung über verteilte Hash-Tabellen. Darüber hinaus enthalten die Routingtabellen des Overlaynetzes Informationen über die jeweiligen Verbindungen. Zu jedem Abarbeitungsbaum von Detektoren existiert zur Verbesserung der Skalierbarkeit und Parallelisierung ein entsprechender Kontrollstrukturbaum. Das Optimierungsziel ist auch hier die Minimierung der Anzahl von Nachrichten. Dies geschieht mittels der Nutzung von zentralen Optimierungsansätzen (Edge, Edge+ und In-Network-Placement), welche in einer dezentralen Variante genutzt werden. Dazu werden allerdings weitere Informationen benötigt (Metadaten) und bspw. zum Austausch von Teilen des Abarbeitungsbaums verwendet. Das Ziel des beschriebenen Optimierungsansatzes ist die Minimierung der Netzwerklast. Dies wird durch drei verschiedene Platzierungsstrategien erreicht, wobei diese zunächst zentralisiert agieren. Auf deren Grundlage entwickeln die Autoren

dezentrale Varianten der drei zentralisierten Ansätze, wobei sie dies vor allem durch zusätzlichen Informationsaustausch (Austausch von Teilen der Abarbeitungsbäume und von Metadaten) erreichen. Die Autoren zeigen mit ihrer Arbeit vor allem, wie zentralisierte Ansätze dezentral ausgeführt werden können, allerdings wird dadurch ein erhöhter Kommunikationsaufwand erzeugt, gleichzeitig erreichen die Ergebnisse nicht das Niveau zentraler Vorgehensweisen.

Sensornetze. Wie auch bei strombasierten Datenbanken wird die dynamische Platzierung von Anwendungen und Anwendungskomponenten genutzt, um die Platzierung einem ebenso dynamischen Umfeld anzupassen. Sensornetzwerke weisen eine Ähnlichkeit zu ereignisbasierten Middlewarearchitekturen auf. Auch im Umfeld von Sensornetzwerken existieren Ansätze zur dynamischen Platzierung von Anwendungen.

So präsentieren Madden et al. [139] einen Optimierungsansatz für Geräte mit kleinem Energiespeicher, so wie auch bei mobilen Geräten der begrenzte Energievorrat einen limitierenden Faktor darstellt. Das dort vorgestellte Verfahren basiert auf dem Mitlesen von Nachrichten, bei dem Broker die durchgeleiteten Nachrichten mitlesen und für sie interessante Nachrichten herausfiltern. Wenn eigene und mitgelesene Informationen keine Überschneidungen aufweisen, werden sie wie sonst üblich weitergeleitet. Sind Überschneidungen detektiert worden, wird lokal entschieden, ob die weiterzuleitenden Informationen verändert werden. Einsparungen in dieser Hinsicht werden jedoch auch durch Routingverfahren beim inhaltsbasierten Routing erreicht (Stichwort überdeckungsbasiertes Routing). Die Optimierung bezüglich des Energieverbrauchs erfolgt darüber hinaus durch die Verlagerung von der Definition der Operatorbäume auf ausreichend gut ausgestattete Geräte. Die Detektion komplexer Informationsmuster erfolgt innerhalb des Netzwerks, indem diese schrittweise verlagert werden. Die Schritte ergeben sich aus dem Anfragegraphen. Neben dem Verwerfen eigener Informationen soll Energie dadurch eingespart werden, dass die Definition von Anfragen und ihre Abbildung auf Baumstrukturen durch eine energiereiche Basisstation durchgeführt werden und diese Aufgaben nicht den schwächeren Sensorknoten zugeordnet werden. Wie die Abbildung erfolgt, bleibt jedoch unbeantwortet. Zusammenfassend lässt sich sagen, dass das Mitlesen von Nachrichten sinnvoll ist, solange lokale Informationen keinen absehbaren Einfluss auf die bereits verfügbaren Informationen haben.

Der Ansatz von Kamiya et al. [120] erlaubt der Anwendung, sich für komplexe Ereignisse zu subskribieren. Das Netzwerk ist heterogen und besteht aus unterschiedlichen Teilnetzen. Diese sind durch Gateway-Knoten miteinander verbunden. Jeder Knoten ist dabei verantwortlich für einen Teilbereich des Netzwerks, er kennt damit auch alle im Teilnetz vorhandenen Subskriptionen, die wiederum baumartig gegliedert sind. Eine abgegebene Subskription wird an einen dem Nutzer nächsten Gateway-Knoten gesendet, welcher die Subskription auflöst bzw. dekomponiert und an die Gateways weiterleitet, die für einzelne identifizierte Teilbereiche des Gesamtnetzwerks zuständig sind. Es wird vor allem Wert

auf Wiederverwendbarkeit von Teilmustern gelegt, sodass die Gateway-Knoten versuchen, neu hinzukommende Subskriptionen durch vorhandene Kompositionsbäume umzusetzen. In ihrem Beitrag beschreiben die Autoren neben der initialen Dekomposition von Ereignismustern auch die Delegation der Detektionsaufgaben. Was allerdings nicht beschrieben wird, ist die Frage, ob das System bei einer Änderung der Umgebung bzw. der Netzwerkstruktur reagiert.

Ein Konzept zur Optimierung auf der Grundlage der Selektivität von Operatoren bieten Srivastava et al. [197]. Optimierungsgrundlage ist die Minimierung der gesamten durch Anfragen und ihre Verarbeitung entstehenden Kosten im Netzwerk, bei denen zwischen Weiterleitungskosten und Verarbeitungskosten unterschieden wird. Der Beginn der Optimierung besteht in einer „gierigen“ Kombination von Weiterleitungspfaden. Darauf aufbauend nutzen die Autoren Verfahren, die zwischen den erwarteten Weiterleitungskosten und der Filterung von Nachrichten balancieren. Allerdings werden nur lokale Filteroperationen in die Kostenbetrachtungen einbezogen, Einsparungen durch Verschiebung in höhere Verarbeitungsstufen spielen bei der Kostenbetrachtung keine Rolle, da nur direkt betroffene Filter bei der Optimierung betrachtet werden. Eine Erweiterung bindet die gesamte Verarbeitungskette in die Optimierung mit ein, und zwar indem die einzelnen Verbindungen als Filter betrachtet werden und ein durchgehender Verarbeitungsstrom entsteht. Unter bestimmten Umständen ist mit Hilfe dieses Ansatzes nach Angaben der Autoren auch eine optimale Lösung möglich. Der Ansatz ist jedoch auf eine Anfrage mit jeweils einer Ereignisquelle beschränkt, eine Erweiterung auf Nachrichtenströme mit mehrfachen Quellen, wie bei der ereignisbasierten Kommunikation üblich, wird von den Autoren jedoch nur diskutiert, aber nicht umgesetzt.

Ying et al. [210] präsentieren einen Algorithmus, der Detektoren komplexer Ereignismuster adaptiv positioniert. Die Autoren des Beitrags heben hervor, dass die Platzierung von Detektoren möglichst nahe bei den Ereignisquellen zwar intuitiv sinnvoll, aber nicht in jedem Fall die beste Platzierungsentscheidung darstellt, da diese Geräte möglicherweise nicht über notwendige Ressourcen verfügen. Sie sind daher nicht in der Lage, umfangreiche Anfragen zu bewältigen. Die Autoren begründen vielmehr, dass Netzwerkoptimierungen bestrebt sind, alle Kosten für Kommunikation, Speicher und Verarbeitung in die Optimierung einzubeziehen. Sie beschreiben zunächst unterschiedliche Optimierungsvorgehen. Sind bspw. bei fehlender Filterung innerhalb des Netzwerks keine Zwischenspeicherungen notwendig, bezieht sich die Optimierung in diesem Fall auf die Minimierung von Weiterleitungs- und Speicherkosten. Die durch die Autoren beschriebene Optimierungsstrategie basiert auf einer Bellman-Ford-Heuristik. Die beteiligten Knoten verfügen über Wissen über ihre Zustände und ihre Kostensituation inkl. der Kosten für Weiterleitungsschemata von Ereignissen, die Weiterleitungskosten von Ereignissen zu Nachbarknoten und Kosteninformationen über die Generierung von Ereignissen bei Nachbarbrokern. Die eigenen Kosteninformationen werden zudem regelmäßig mit den Nachbarknoten ausgetauscht. Der Optimierungsansatz adaptiert die Detektorplatzierung auf der Grundlage lokal verfügbarer Informationen über eigene und die gesammelten Statusinfor-

mationen der Broker in der Nachbarschaft. Ist bei einer Verschiebung eines Detektors zu einem Nachbarbroker zu erwarten, dass seine Ausführung kostengünstiger durchgeführt werden kann, erfolgt die Verschiebung. Detektoren und dazugehörige Caches werden jedoch als Ganzes verschoben, das mögliche Verbesserungspotential durch Dekomposition und Rekombination von Detektoren und Caches wird nicht ausgenutzt.

Einen anderen Ansatz zur optimalen Platzierung von Detektoren verfolgen Abadi et al. [1]. Sie ermöglichen keine freie Platzierung von Detektoren, sondern setzen auf vordefinierte, mögliche Ausführungsorte und eine beschränkte Auswahl von Operatortypen. Durch die Einschränkung möglicher Ausführungsorte werden sowohl die Komplexität des Platzierungsproblems, als auch das Optimierungspotential eingeschränkt. Das Optimierungsziel ist, wie so häufig, die Lastbalancierung. Zur Zielerreichung werden unterschiedliche Ebenen definiert, nämlich eine lokale Umgebung und eine globale, in denen jeweils mit beschränkten Operatortypen Verbesserungen erreicht werden.

Auf der Ebene der aktiven Datenbanken stellen Paton und Díaz [174] ein Konzept auf der Grundlage der *CORBA*-Architektur dar. Die Autoren beschreiben zwar die Implementierung und Verwaltung von Detektionsmechanismen mit der Verwaltung von Regeln und Services über verschiedene Ereigniskanäle mittels „push-“ und „pull-“ Mechanismen, allerdings wird nicht diskutiert, wie Platzierungen von Detektoren verändert und in den Ansatz integriert werden können.

Das von Yao und Gehrke [208] vorgeschlagene System *Cougar* arbeitet im Bereich von Sensornetzen. Die Netze selbst bestehen aus Teilnetzen, die mittels Gatewayknoten miteinander verbunden sind. Die Gatewayknoten verfügen über das Wissen aller im Teilnetz verfügbaren Ereignisse und Ereignismuster. Auf Basis der Zerlegung einer Anfrage in ihre Bestandteile werden, wie im Bereich von Datenbanken, die Anfrage aufgegliedert und Teilanfragen auf die Gatewayknoten verteilt, die über die entsprechenden Ereignismuster verfügen. Dabei können Teilanfragen frühzeitig bearbeitet werden, sodass Ereignisse bzw. Ereignismuster zu einem frühen Zeitpunkt herausgefiltert werden. Auf diese Weise lässt sich Energie, die in Sensornetzwerken ein stark limitierenden Faktor ist, einsparen.

Eine Weiterentwicklung dieses Konzepts findet sich bei Bonfils und Bonnet [21]. Bei diesem Ansatz wird die Platzierung von Operatoren zwar ebenfalls laufend überprüft und angepasst, dies wird jedoch durch Exploration des Nachbarschaftswissens erreicht. Der präsentierte Ansatz arbeitet auf der Grundlage eines Sensornetzes. Das ist ähnlich einer Datenbank aufgebaut [22] und basiert auf hierarchisch organisierten Knoten, die in ihrer Nachbarschaft Kosteninformationen sammeln und austauschen. Grundlage der Platzierung ist ein Kostenmodell, bei dem Transferkosten, Datenraten und Routingkosten proportional zueinander wachsen bzw. schrumpfen. Zwecks der Platzierungsoptimierung werden zusätzliche Kostennachrichten zwischen den Knoten versandt. Lokale Kosteninformationen bzw. Kostenzustände der jeweiligen Broker werden dabei an die jeweiligen Elternknoten weitergeleitet. Zudem werden Elternknoten in bestimmten Mengen zusammengefasst, welche mögliche Platzierungsalternativen

für in der Nachbarschaft bereits deployte Anwendungen darstellen. Alle in diesen „Kandidatenmengen“ enthaltenen Knoten erhalten die ermittelten Kosteninformationen und kalkulieren auf dieser Basis und ihrer eigenen Kostensituation die fiktiven Kosten, die bei Anwendungsausführung anfallen. Durch Vergleich der eigenen fiktiven Kostensituation mit der mitgelieferten Kostensituation des momentan ausführenden Knotens ermitteln die fiktiven Ausführungsknoten eigenständig, ob durch eine Platzierungsänderung der Anwendung eine günstigere Kostensituation erreicht werden kann. Ist dies der Fall, wird die Anwendung neu platziert und der Informationsstrom umgeleitet. Der Ansatz basiert auf regional verfügbaren Informationen, die aktuelle Kostensituation wird nur einer begrenzten Kandidatenmenge mitgeteilt. Auch die Koordination der Kandidaten mit der ausführenden Stelle erfolgt lokal begrenzt. Allerdings ist die Einführung zusätzlicher Kostennachrichten gegensätzlich zum Prinzip der Vermeidung zusätzlicher Nachrichten. Darüber hinaus ist bei mehreren möglichen Ausführungsorten ein gewisser Grad an Koordination nötig. Außerdem hängt der Erfolg des Verfahrens auch mit der Auswahl und vor allem der Größe der Kandidatenmenge zusammen, wobei deren Größe mit der benötigten Anzahl von Nachrichten korreliert. Des Weiteren sind die Bestimmung der Kandidatenmenge und der alternativen Weiterleitungspfade mitverantwortlich für die möglichen Einsparungen und beeinflussen gleichzeitig den zusätzlichen Overhead.

Publish/Subscribe Middleware. Im Umfeld Publish/Subscribe-basierter Systeme existieren ebenfalls Ansätze zur Platzierungsoptimierung. So präsentieren Carzaniga et al. [31] einen Ansatz auf der Grundidee des Matchings von Ankündigungen und Subskriptionen komplexer Ereignismuster in der Nachbarschaft. Dazu halten die Broker neben den eigentlichen Routinginformationen zusätzlich Informationen (Tabellen) über die an den Nachbarbrokern verfügbaren Ereignismuster. Jeder Broker hält eine Tabelle mit Ereignismustern, welche den Broker erreichen, sowie den bereits verarbeiteten Ereignismustern. Zusätzlich wird in der Tabelle gespeichert, aus welcher Richtung bzw. von welchem der Nachbarbroker, der Broker die Ereignismuster erhalten hat. Erreicht den Broker eine komplexe Ereignissubskription, sucht er nach passenden Einträgen in seiner Tabelle. Findet sich lokal kein passender Eintrag, wird die Subskription in kleinere Muster aufgeteilt und an die Nachbarbroker zur Verarbeitung weitergeleitet. Der Ansatz von Carzaniga et al. [31] ermöglicht das Aufsplitten und die Verteilung von Detektoren für Teilmuster, wobei allerdings nur eine vergleichsweise kleine Menge möglicher Muster unterstützt wird. Leider wird nicht erläutert, welche Vorbedingungen zu welchen Platzierungsentscheidungen führen. Wie bereits im Abschnitt 3.2.1 erwähnt wurde, beschreibt auch O'Keeffe [165] sowohl einen Ansatz für die initiale, als auch für die laufende Platzierung. Kernpunkte bei der Platzierungsanpassung sind die Platzierung von Komponenten in einem virtuellen Kostenraum und deren Abbildung auf das reale Netzwerk sowie die Wiederverwendung von bereits vorhandenen Komponenten.

Der Ansatz von O'Keeffe [165] nutzt unterschiedliche Schichten in einer Rendezvous-basierten Middleware, hier den Ansatz von *Hermes* von Pietzuch et

al. [178], einer Middleware die ursprünglich für die strombasierte Verarbeitung entwickelt wurde. Die Grundidee der Platzierung besagt, dass jeder Broker eine virtuelle Position in einem Kosten- bzw. Latenzraum inne hat, also Broker über Netzwerkkoordinaten verfügen. Diese werden mit Hilfe des Vivaldi-Algorithmus berechnet. Einige Ausführungen zu Netzwerkkoordinaten und dem Vivaldi-Algorithmus finden sich bereits in Abschnitt 5.2.2. Jede Komponente kennt ihre Position im virtuellen Latenzraum, die modellierte Kraft zwischen zwei Brokern wird bestimmt durch die berechnete und die tatsächlich ermittelte Latenz. Durch eine dezentrale Verschiebung von Komponenten wird die resultierende Kraft zwischen den betroffenen Komponenten abgebaut und die Latenzen nähern sich einander an.

Nach erfolgter initialer Platzierung wird die Optimierung durchgeführt. Subscriber und Publisher primitiver Ereignisse haben im Modell einen festen Standort und die Detektoren wurden bereits im Netzwerk platziert. Jeder Detektor kennt sowohl die Koordinaten seines eigenen Brokers, als auch die Koordinaten der Broker, die jeweils die über- bzw. untergeordneten Detektoren des Anfragegraphen tragen. O'Keeffe [165] schlägt das Mitführen dieser Informationen in den Notifikationen entlang des Ereignispfades oder die Erweiterung von Subskriptionen um die geforderten Informationen vor. Anhand der Unterschiede zwischen vorhergesagten und tatsächlichen Latenzen bestimmen sich die auf den Detektor einwirkenden Kräfte, durch Migration der Komponenten werden die Kräfte abgebaut. Die resultierende Kraft bestimmt jeder Detektor lokal und berechnet mit Hilfe dieser Kraft seine Position im (globalen) Kostenraum. Findet sich im zweiten Schritt, der Abbildung auf das physische Netzwerk, ein Broker in der Nähe der ermittelten Position, erfolgt die Migration des Detektors auf den ermittelten Broker. Der Ansatz nutzt eine Mischung aus zentralen und dezentralen Elementen zur Detektorplatzierung mit lokaler Berechnung einer Position eines Detektors in einem globalen Kostenraum. Eine Anfrage wird von vornherein in einzelne Komponenten zerlegt und initial, auch mit Hilfe von Rendezvous-Brokern, verteilt.

Abschließend zur Betrachtung der unterschiedlichen Ansätze zur verteilten Detektion komplexer Ereignismuster in unterschiedlichen Anwendungsgebieten lässt sich feststellen, dass eine Vielzahl von Ideen für die Detektorplatzierung existiert, allerdings ist die Verteilung von Platzierungsentscheidungen nicht einheitlich. Die Arbeit von O'Keeffe [165] präsentiert dabei einen sehr guten Überblick über bestehende Arbeiten. Die besprochenen Ansätze unterscheiden sich neben ihren Hauptanwendungsgebieten vor allem durch die zugrunde liegenden Kostenmodelle und Optimierungsziele. Viele Arbeiten nutzen das Potential nicht aus, welches sich aus der Dekomposition und Rekombination von Ereignismusterdetektoren heraus entwickelt. Darüber hinaus wird von der Möglichkeit, vorhandene Detektoren wiederzuverwenden, oft nicht Gebrauch gemacht.

5.2.3 Selbstorganisierte und verteilte Detektion

Das Problem, Detektoren für komplexe Ereignisse in einem Publish/Subscribe-basierten Kommunikationsnetzwerk zu positionieren, ist ein Anwendungsbeispiel zur Verteilung von allgemeinen Anwendungskomponenten, worauf in Kapitel 3 dieser Arbeit ausführlich eingegangen wurde. Es bleibt, wenn auch vereinfacht, das Problem der Zuweisung von Aufgaben zu den sie ausführenden Instanzen, das sogenannte *Task Assigning Problem*, welches selbst in statischen Umgebungen NP-schwer [20] ist. Die inhärente Dynamik der Umgebung und der Anwendungen und deren Folgen für die Anwendungsverteilung durch Veränderungen der Subskriptionen und Änderungen von Produktions- und Detektionsraten wurden ebenfalls ausführlich beschrieben. Aufgrund der Dynamik des Systems ist die Adaption der Anwendungsplatzierung notwendig, sollen doch in einer Umgebung mit mobilen Geräten und beschränkten Ressourcen möglichst die Beanspruchung von Kommunikationsverbindungen und damit der Energieverbrauch minimiert werden. In dem in diesem Kapitel vorgestellten Anwendungsfall wird gemäß des vorgestellten Konzepts auf eine Lösung mittels globalem Wissen verzichtet, da die Sammlung der benötigten Informationen die verfügbaren Ressourcen belasten und eine Zentralisierung möglicherweise zu einem Flaschenhals führen würde.

Daher wird der Einsatz eines dezentralisierten, selbst-organisierten Verfahrens zur adaptiven Platzierung von Detektoren für komplexe Ereignismuster vorgeschlagen, dessen Konzept in Kapitel 3 dieser Arbeit vorgestellt wurde. Die Platzierungsentscheidung wird auf der Grundlage einer Heuristik getroffen, welche mit dem Ziel, Kosten einzusparen, ausgeführt wird. Der Ansatz bedient sich der vier grundlegenden Operationen auf den Anwendungen, nämlich der *Dekomposition*, *Replikation*, *Migration* und *Rekombination*. Diese wurden in Abschnitt 3.5.2 dieser Arbeit vorgestellt. Durch die unterschiedliche sequenzielle Ausführung von Basisoperationen wird die Detektorplatzierung schrittweise verbessert und zu einem lokalen Optimum geführt. Obwohl die Basisoperatoren bereits grundlegend erläutert wurden, wird ihr Verhalten im Folgenden noch einmal mit Bezug auf die Detektoren komplexer Ereignismuster beschrieben.

Detektordekomposition. Bei dieser Operation wird ein Detektor eines komplexen Ereignismusters in Bestandteile zerlegt, wobei jeder Teildetektor für eine jeweilige Teilmenge von Ereignissen zuständig ist. Der Teildetektor erfasst Ereignismuster in Form partieller Ereignismuster. Die detektierten Ereignisse werden anschließend an den in der Detektorhierarchie auf nächsthöherer Verarbeitungsstufe stehenden (Teil-) Detektor zur weiteren Bearbeitung weitergeleitet. Dort werden die Teilmuster wiederum verarbeitet und weitergeleitet, bis die jeweiligen Teilereignismuster den Detektor der höchsten Stufe erreicht haben.

Die Aufteilung der Detektoren erfolgt nach deren interner Gliederung, die primär durch den Anwendungsentwickler bestimmt wird. Im Wesentlichen wird in dieser Arbeit auf die Algebra von [35] zurückgegriffen, welche Ereigniskompositionsoperatoren und Konsumptionsmodi nutzt (siehe Abschnitt 5.2.1). Wichtige,

d.h. im Kontext der Arbeit, genutzte Operatoren sind Disjunktion, Konjunktion, Sequenz und der Beliebig-Operator zusammen mit Angaben zur zeitlichen Gültigkeit und zum Umfang des Ereignismusters.

Detektorreplikation. Die Detektorreplikation teilt den Ereignisraum horizontal in disjunkte, nicht überlappende Teilbereiche auf. Jeder resultierende Detektor ist exklusiv für einen Teil des Ereignisraums zuständig. Sollen die Replikate zum Erreichen eines Optimierungsziels beitragen, so ist die reine Replikation der Detektoren regelmäßig nicht ausreichend, stattdessen muss die Replikation mit einer Verschiebung (Migration) des/der replizierten Detektoren kombiniert werden. Sind bspw. Übertragungsressourcen einzusparen, werden die betreffenden Detektoren i.d.R. räumlich in der Nähe oder in der Schnittmenge der betreffenden Teilereignisräume angesiedelt.

Detektormigration. Die Migration von Detektoren stellt die elementare Operation zur Verbesserung der Kostensituation dar. Sie dient der nahtlosen Migration der Detektoren. Mit dieser Aktion werden Detektoren von einem Ausführungsort zu einem anderen verschoben. In REBECA sind die Ausführungsorte die Broker der Middleware. Nach der Verschiebung der Detektoren zum neuen Ausführungsort werden sie dort nahtlos weitergeführt. Lassen sich nicht benötigte Ereignisse frühzeitig und selektiv herausfiltern, kann die Verschiebung von Detektoren insbesondere entlang von Ereignisströmen (vor allem in Richtung der Ereignisquellen) die Netzwerkressourcen in beträchtlichem Maße entlasten und damit (Weiterleitungs-) Kosten einsparen.

Detektorrekombination. Die Detektorrekombination fügt vormals dekomponierte bzw. auch replizierte Detektoren oder Detektoren mit gleichem Filter zu einem Detektor zusammen. Sinnvoll ist die Operation immer dann, sobald bei zwei Ereignisquellen gleichen Typs die Ereignisproduktionsrate eines Ereignisstroms signifikant abgenommen hat oder ein Ereignisproduzent gänzlich ausfällt. Statt zwei Detektoren mit ihren jeweiligen Zuständen und den dazugehörigen Ereignisflüssen und Subskriptionen zu verwalten, werden zwei oder mehrere Detektoren an einem Ort zusammengefasst. Die Platzierung erfolgt an einem neuen, gemeinsam genutzten Standort oder die Detektoren werden an einem bereits bestehenden Standort zusammengefasst. Die Rekombination umfasst außer dem Auflösen eines Detektors auch die Zusammenführung von Detektorzuständen, vor allem die Zusammenführung von zwischengespeicherten, partiellen Ereignissen. Nach der Rekombination wird ein längerer und damit kostenintensiverer Weg von Ereignissen zum dann zuständigen Detektor in Kauf genommen. Diese im Vergleich höheren Kosten werden bei einer Detektorrekombination jedoch durch die Vorteile, die sich durch die Vermeidung von doppeltem Verwaltungs- und Speicheraufwand ergeben, wieder aufgewogen.

5.2.4 Selbstorganisierende Detektorplatzierung

Die selbstorganisierende Detektorplatzierung als das Kernthema dieser Arbeit steht auch im Mittelpunkt der Evaluation. Die Grundprinzipien der selbstorganisierenden Platzierung werden in diesem Abschnitt zunächst wiederholt und anschließend bewertet.

Wiederverwendung. Ein wichtiges Grundprinzip ist, wie bei verwandten Arbeiten, die Wiederverwendung von Detektoren. Die Wiederverwendung wird im Kontext der Verschiebung von Detektoren betrachtet, d.h. unter dem Aspekt des nahtlosen Weiterführens von Detektoren oder Anwendungen im Allgemeinen. Die von den Detektoren erzeugten Ereignisse sind auch nach ihrer Verschiebung von Subskribenten sichtbar und werden vom Notifikationsdienst so behandelt wie zuvor. Die Detektoren werden nicht mehrfach ausgeführt, womit auch ihre Zustände nicht redundant gehalten werden müssen. Dies spart sowohl Verwaltungs- und Ausführungsaufwand, als auch die Kosten für die Kommunikation zwischen redundanten Detektoren. Um Detektoren nahtlos weiterzuführen, werden sie in ihrer Funktion unterbrochen, migriert und anschließend an gleicher Zustandsstelle weitergeführt. Sie müssen so erkennbar und in ihrer Zustandshaltung beschaffen sein, dass eine Rekombination nach einer Dekomposition oder Replikation möglich ist. Dies lässt sich über eine bedarfsweise durchgeführte Analyse der eingehenden und ausgehenden Ereignisströme und über die angewendeten Ereignisfilter und die Komposition von Ereignissen herausfinden. Im Sinne der schnelleren Durchführbarkeit wird in dieser Arbeit ein anderer Ansatz vorgeschlagen. Dieser beruht auf einer Kennzeichnung der Detektoren und der von ihnen verarbeiteten und produzierten Ereignisse (Subskriptionen, Ankündigungen) sowie der genutzten Filter.

Kostenmodell. Während des Betriebs von Detektoren bzw. von Anwendungen lässt sich eine Vielzahl unterschiedlicher Kosten identifizieren. Im Kostenmodell des Anwendungsfalls sind dies insgesamt vier unterschiedliche Kostenarten. Zunächst lassen sich durch die Ausführung von Anwendungen die (1) *Ausführungskosten* beschreiben. Sie entstehen durch die Bewertung des Ressourcenverbrauchs am sie ausführenden Gerät. Ein weiterer Kostenblock sind die durch den Aufwand zur Speicherung von Ereignissen bzw. Teilereignismustern verursachten Kosten, die sog. (2) *Speicherkosten*. Die Kosten werden hervorgerufen durch bereits detektierte, partielle Ereignismuster, die durch die Broker selbst gespeichert und verwaltet werden. Bei der Zusammensetzung der Speicherkosten kann von einer Grundlast durch das Vorhalten von Strukturen zur Hinterlegung der Ereignisse/Ereignismuster und der eigentlichen Speicherung mit Belegung der Speicherzellen ausgegangen werden. Dieser zweite, flexible Kostenanteil fließt in das Kostenmodell ein, während der fixe Anteil, hervorgerufen durch die Bereitstellung der Verwaltungsmöglichkeit, nicht in die Kostenbetrachtung eingeht, da er unabhängig von der Anwendung entsteht und durch eine Änderung der Anwendungsplatzierung nicht beeinflusst wird. (3) *Weiterleitungskosten* spiegeln die

Kosten, die durch Weiterleitung von Notifikationen entstehen, aber auch die Weiterleitung und Verwaltung von Subskriptionen und Ankündigungen, wider. Die Weiterleitung von Ankündigungen lässt sich durch eine veränderte Platzierung der Detektoren im Netzwerk nicht bzw. nur wenig beeinflussen, da die Verteilung und Verwaltung der Ankündigungen maßgeblich von der Platzierung der Ereignisproduzenten und dem genutzten Verteilungsalgorithmus abhängig sind. Ebenso hängt der Weiterleitungs- und Verwaltungsaufwand für Subskriptionen von dem verwendeten Algorithmus und der Platzierung der Detektoren ab. Unter der Annahme, dass der Aufwand für die Verwaltung und Weiterleitung von Ankündigungen und Subskriptionen hauptsächlich vom genutzten Weiterleitungsalgorithmus abhängt und durch die Platzierung von Detektoren weder ein eindeutig positiver oder negativer Effekt auf die resultierenden Kosten zu beobachten ist, sind bei der Kostenbetrachtung lediglich die Weiterleitungskosten von Interesse, die direkt der Kommunikation zwischen den Detektoren zugeordnet werden können. Als vierter Kostenfaktor werden die (4) *Platzierungskosten* betrachtet. Hierbei werden nur die Kosten in das Kostenmodell aufgenommen, die durch tatsächliche Ausführung der Basisoperationen anfallen. Platzierungskosten werden durch den Aufwand zur initialen Platzierung sowie durch die Ausführung der vier Basisoperationen (inkl. der damit verbundenen Tätigkeiten wie Serialisierung/Deserialisierung, Versand der serialisierten Komponenten über Netzwerk) und der (Wieder-/Neu-) Platzierung der Detektoren hervorgerufen.

Primäres Ziel der Verschiebung von Detektoren ist das Einsparen von Weiterleitungskosten, da diese direkt die Netzwerk- und Brokerauslastung widerspiegeln, welche wiederum den Energieverbrauch direkt und entscheidend beeinflussen. Im betrachteten Szenario ist die Energie der entscheidende Faktor. Dabei steht außer Frage, dass eine Neupositionierung von Detektoren zunächst Energie verbraucht und damit Kosten erzeugt, während mögliche Einsparungen sich erst anschließend bemerkbar machen. Die Verbesserung der Platzierung erfolgt auf der Grundlage lokalen Wissens, unter Zuhilfenahme des Kräftemodells, durch Ausführung einer Sequenz der Basisoperationen.

Aus den Weiterleitungskosten, Speicherkosten und Platzierungskosten leiten sich die auf die Platzierung einwirkenden Kräfte ab. Die Kräfte verhalten sich proportional zu den ausgelösten Kosten, welche in unserem Modell wiederum direkt von den Ereignissen abhängen, die (1) einen Detektor erreichen/verlassen bzw. (2) von ihm gespeichert werden. Die (3) Platzierungskosten orientieren sich dagegen am Aufwand der Serialisierung, Verschiebung und Deserialisierung von Detektoren, welche weitgehend unabhängig vom hinterlegten Zustand des Detektors sind. Die jeweiligen Kräfte wirken auf die Platzierung eines Detektors ein, wobei die ermittelten Kräfte die möglichen Einsparungen repräsentieren, die durch eine Platzierungsänderung realisiert werden können.

Ereignisnotifikationsflüsse erreichen einen Detektor aus unterschiedlichen Richtungen (Nachbarbrokern), ebenso fließen Notifikationen in Richtung der unterschiedlichen Nachbarbroker ab. Für jede Richtung wird zunächst die Nettosumme aller exklusiv ein- und ausgehenden Notifikationen bestimmt. Exklusiv

bedeutet, dass eingehende Notifikationen nur gezählt werden, wenn sie nicht in Richtung anderer Nachbarbroker weitergeleitet werden, also exklusiv verbraucht werden. Exklusiv sind auch ausgehende Notifikationen immer dann, wenn diese durch auf dem Broker beheimatete Detektoren erzeugt werden (und ebenfalls nicht lediglich weitergeleitet werden). Im Zusammenhang mit den auf einen Detektor einwirkenden Kräften, werden die eingehenden Notifikationen auch als „konstituierende", „partielle" Ereignisse oder Ereignismuster bezeichnet, während die „detektierten" Ereignismuster als Synonym für die ausgehenden Notifikationen bekannt sind. Überwiegt die Anzahl der eingehenden Notifikationen die der ausgehenden, wirkt eine Kraft, welche den Detektor in Richtung des Nachbarbrokers zieht, überwiegt hingegen die Anzahl der ausgehenden Notifikationen, wird der Detektor an seinem derzeitigen Ausführungsort festgehalten. Die entstehende Kraft ist prinzipiell proportional zum Betrag der Summe der resultierenden Notifikationen.

Nachdem die resultierende Kraft je Richtung aus allen ein- und ausgehenden exklusiven Notifikationen je Detektor und je Richtung berechnet wurde, wird im zweiten Schritt ermittelt, ob eine Kraft, d.h. die resultierende Kraft, aus einer Richtung alle anderen Kräfte der anderen Richtungen überwiegt, oder ob am Ende mehrere Kräfte gleichmäßig stark auf einen Detektor einwirken. Im in der Arbeit genutzten Kräftemodell ist das Einwirken der Kräfte wie ein „Ziehen" des Detektors in die Richtung der resultierenden Kraft zu verstehen. Dominiert eine Richtung alle anderen, ist eine Migration angebracht, bei zwei oder mehr dominanten Richtungen eine Replikation mit anschließender Migration.

Neben der Weiterleitung von Ereignisnotifikationen kann auch die Speicherung von Notifikationen einen erheblichen Beitrag zu den Gesamtkosten leisten. Sind bspw. durch eine doppelte Ausführung von Komponenten Kosten für die Speicherung von Notifikationen enstanden, lassen sich durch eine Rekombination die Kosten der doppelten Speicherhaltung reduzieren. Die mögliche Reduktion wirkt entgegen der möglichen Einsparung der Weiterleitungskosten und zieht ehemals miteinander verbundene Detektoren wieder zusammen.

Eine Migration bzw. Replikation mit anschließender Migration oder eine Rekombination werden durchgeführt, wenn das Kräftemodell potentielle Einsparungen bei der Weiterleitung bzw. Speicherung von Ereignisnotifikationen in Aussicht stellt. Möglicherweise sind diese in Aussicht gestellten Einsparungen jedoch nicht groß genug, um eine dauerhafte Platzierungsänderung zu begründen. Dies könnte in einer Folge von sich gegenseitig aufhebenden Platzierungsänderungen münden. Um solche alternierenden Platzierungsänderungen zu verhindern, wurde eine Kraft eingeführt, die proportional zu den Platzierungskosten wächst. Diese spiegelt die Ausführungskosten der Basisoperationen als „Reibung" wider und wirkt der Ausführung der Basisoperationen entgegen. Vor der Ausführung einer der Basisoperationen muss die resultierende Kraft (Einsparungen von Notifikationsflüssen bzw. Einsparung von Zwischenspeicher) die Reibungskraft als Widerstand übersteigen. Die Reibungskraft ermöglicht außerdem die Responsibilitätssteuerung des Systems.

Initiale und laufende Platzierung. Auf der Grundlage der vier Basisoperationen Dekomposition, Migration, Rekombination und Replikation sowie den vorgestellten Grundzügen der selbst-organisierenden Platzierungsstrategie, wird im Folgenden der in dieser Arbeit vorgestellte Ansatz am Beispiel komplexer Ereignisdetektoren beschrieben.

Initiale Platzierung - Detektoren in peripheren Geräten. Ausgangspunkt ist eine Detektorplatzierung, bei der die Detektoren zunächst auf den Geräten positioniert sind, die sich für bestimmte Ereignismuster interessieren. Sie befinden sich also auf dem Gerät, welches die Subskriptionen der Anwendungen zuerst ins Netzwerk publiziert hat. Ziel der Optimierung ist bei diesem Anwendungsfall, dass im Netzwerk zur Laufzeit der Detektoren möglichst wenige Kosten anfallen. Mit Hilfe einer Sequenz der Basisoperatoren (siehe Abschnitt 5.2.3) sollen die Detektoren so im Netzwerk verteilt werden, dass ein möglichst vorteilhafter Zustand erreicht wird. Der Verbund der Detektoren muss trotz veränderter Platzierung die selben, korrekten Ergebnisse liefern wie im ursprünglichen Setup.

5.3 Simulation und Simulationsumgebung

Der Nachweis, dass der präsentierte Ansatz die Kosten der Anwendungsausführung im Gesamtnetzwerk signifikant senkt, kann durch Simulation nachgewiesen werden. Der Einsatz von Simulationen bringt nicht nur Vorteile mit sich. Sie überwiegen aber gegenüber den Nachteilen, die die Evaluation eines realen Systems in realer Umgebung mit sich bringt. Der vorgestellte Ansatz wurde in die bestehende Publish/Subscribe-Middleware REBECA integriert. Die in dieser Middleware verschickten Ereignisnachrichten werden auch im Umfeld intelligenter Umgebungen genutzt, um Zustände bzw. Zustandsänderungen von Systemen zu repräsentieren. Die Simulation bettet die Middleware in ein ereignisorientiertes Gerüst ein und versorgt sie von außen mit Ereignissen, die anschließend durch die Middleware verarbeitet werden. REBECA ist konzeptionell ein diskret-ereignisorientiertes System, womit auch die eingesetzte Simulation eine diskret-ereignisorientierte Arbeitsweise unterstützen muss [69]. Bei der Modellierung und Simulation stellt sich zunächst die Frage, welche Aspekte mit welcher Detailtreue modelliert und simuliert werden sollen. Wird die Detektion von Ereignismustern als zusammenhängende Folge von Abarbeitungsschritten als Modellierungs- und Simulationsziel betrachtet, lässt sich dieser Prozess als das Emittieren von Ereignissen, deren Weiterleitung und die Abarbeitung der Detektion beschreiben. Abgebildet auf die Arbeitsweise von Publish/Subscribe heißt dies, dass das Absenden und Routing einer Notifikation, die Abgabe einer Subskription und deren Routing, die Verarbeitung einer empfangenen Subskription, deren Verarbeitung und die Abgabe einer neuen Notifikation umgesetzt werden müssen. Eine Möglichkeit der Umsetzung ist es, zunächst die umgesetzte Middleware in einem Modell abzubilden und zu simulieren. Den Grundprinzipien der Modellbildung folgend bedeutet dies, die Kernaspekte des Systems zu

analysieren und durch ein mathematisches Modell darzustellen. Mit Hilfe eines Simulators und der entsprechenden Instrumentierung lassen sich anschließend die gewünschten Experimente durchführen. Ein anderer Ansatz ist die Einbindung der implementierten Middleware durch einen Simulator. Da die Middleware implementiert vorhanden ist und für Simulationen genutzt werden kann, ist keine Abstraktion von ihr nötig.

Die Evaluation und der Nachweis, dass das vorgestellte Platzierungsverfahren Kosten einspart, erfolgt mittels Integration der adaptiven Anwendungsverteilung in die Middleware REBECA, wie es bereits bei weiteren Konzepten in der Vergangenheit realisiert wurde. Die hinzugefügten, funktionalen Middlewarekomponenten wurden bereits in Kapitel 4 erläutert. Zusätzlich zur Funktionalität wurden auch die Simulation der Erzeugung, Verarbeitung und Weiterleitung der Ereignisse implementiert. Sie umfasst die zusätzlichen Java-Klassen zur Spezialisierung der *BasicBrokerEngine* und der Schnittstelle *BrokerInterface*, die Spezialisierung der *Event*-Klasse sowie die Erweiterungen der Schnittstellen *Component* und *Serializable* durch die neuen Schnittstellen *Replicable* und die daraus abgeleitete Schnittstelle *Migratable* sowie die abgeleitete Klasse *MigrationQueue*. Die mit den funktionalen Klassen und den Simulationsklassen angereicherte Middleware REBECA wird durch den Simulator PEERSIM gespeist.

5.3.1 Simulator

PEERSIM [146] ist ein Simulator, der neben einem zyklenbasierten Simulator auch einen diskret-ereignisorierten Simulator bietet. Letzterer ist umfangreicher und wurde für weitläufige Netzwerke mit direkter (Peer-to-Peer) Kommunikation entwickelt. PEERSIM abstrahiert von Abläufen tieferer, vor allem physischer Netzwerkschichten, sodass der Simulator für den Einsatz von Simulationen von Overlay-Netzwerken weite Verbreitung findet.

Neben PEERSIM gibt es noch weitere diskret-ereignisorientierte Simulatoren für Netzwerke auf der Grundlage ereignisorientierter Kommunikation, bspw. die Simulatoren der *ns-* (*network simulator*) Reihe, wobei *ns-3* [185] den aktuellen Stand der Entwicklung darstellt. Der Simulator ns-3 hat mit seinen Vorgängern gebrochen und wurde neu entwickelt [93]. Weitere geläufige Simulatoren für Netzwerke sind OMNet++ [202] sowie OPnet [38]. Sowohl ns-2 und ns-3 aber auch OMNet++ sind komponentenbasierte Netzwerksimulatoren, die auch eine grafische Nutzeroberfläche bieten. Findet ns-3 vor allem im akademischen Bereich seine Anwendung, wird OMNet++ neben der Forschung auch in der Industrie verwendet. OPnet dagegen wird vorwiegend industriell genutzt. Es ist besonders nutzerfreundlich, bietet eine umfangreiche Komponentenbibliothek und weitere unterschiedliche Simulationsmethoden [167]. Die letztgenannten Simulatoren haben den großen Vorteil, eine hohe Güte der Simulationsergebnisse zu liefern, welche mit Ergebnissen, die durch Beobachtungen einer in einem realen System umgesetzten Middleware erhalten werden könnten, konkurrieren. Um jedoch diese hohe Genauigkeit erreichen zu können, müssen die physische Datenübertragung

genauso wie der gesamte Protokollstapel abgebildet und simuliert werden. Dies ist allerdings aufwendig und damit kostenintensiv. Gleichzeitig wäre die Aussagekraft der ermittelten Ergebnisse nur auf die konkret simulierten Szenarien übertragbar, sodass die simulierten Anwendungsgebiete nur begrenzt auf weitere, reale Szenarien anwendbar sind [169]. Durch den Einsatz von PeerSim und die einhergehenden Abstraktions- und individuellen Vertiefungsmöglichkeiten sowie die umfassenden Anpassungsmöglichkeiten lässt sich PeerSim mit Rebeca besonders gut kombinieren, sodass aussagekräftige Evaluationsergebnisse bei vertretbarem Aufwand erbracht werden können. Allerdings greift PeerSim nicht einfach auf Rebeca zu und versorgt die Middleware mit Ereignissen, sondern ist zunächst für den Aufbau und Betrieb des Netzwerks zuständig, auf dem die Middleware ausgeführt wird. Das im Folgenden als PeerSim-Netzwerk bezeichnete Netzwerk besteht daher aus PeerSim-Brokern, welche miteinander verbunden sind und gleichzeitig als Ausführungscontainer für Protokollinstanzen dienen. Für das Deployment eines PeerSim-Netzwerks werden daher ausführbare Protokolle benötigt, welche durch die PeerSim-Broker im Betrieb in Form von Protokollereignissen ausgeführt und verarbeitet werden. Darüber hinaus muss durch den Entwickler eine Konfigurationsdatei bereitgestellt werden, in der allgemeine Informationen über den Aufbau und Ablauf der Simulation (Netzwerkgröße, Ende der Simulation) sowie Angaben zu Auswahl und Ablauf von auszuführenden Protokollen inklusive der benötigten Parameter und Einstellungen zusammengefasst sind. Der Aufbau des Netzwerks erfolgt schrittweise, indem zunächst ein Musternetzwerkknoten erstellt wird. Dieser enthält alle benötigten Protokolle in ausführbarer Form und dient als Muster bzw. *Prototyp* für alle weiteren Broker. Der prototypische Broker wird anschließend solange geklont, bis die gewünschte Anzahl von Brokern zur Verfügung steht. Diese Vorgehensweise setzt ein bekanntes *Erzeugungsmuster* der Softwareentwicklung (engl. *design pattern*) um, nämlich das des *Prototyps* [75]. Prototypen können auch so gestaltet sein, dass sie sich bei ihrer konkreten Ausgestaltung unterscheiden und so eine gewisse Varianz beim Deployen konkreter Instanzen ermöglichen. Welche und wie viele Ausprägungen des Prototyps konkret aufgebaut werden, wird durch den Inhalt des Instanziierungsprotokolls bestimmt, indem es weitere Steuerungsprotokolle startet, die dann die Knoten ausdifferenzieren. Das Instanziierungsprotokoll wird einmal vor Beginn der eigentlichen Simulation ausgeführt. Demgegenüber lassen sich Steuerungsprotokolle mehrfach und auch periodisch aufrufen. Aus den Ausführungen zu PeerSim ist erkennbar, dass es auf der Definition und der Ausführung von Protokollen basiert. Zur Evaluation des in dieser Arbeit vorgestellten Ansatzes ist seitens von PeerSim nötig, die notwendigen Ausführungsprotokolle, vor allem der Migration, und Simulationsereignisse bereitzustellen, wobei letztere die Publish/Subscribe-basierte Kommunikation umsetzen. Rebeca ermöglicht durch den modularen Aufbau eine vergleichsweise komfortable Einbindung der gewünschten PeerSim-Protokolle, da sich die benötigten Plugins modular integrieren lassen.

Die Simulation des Netzwerks und der Broker bauen auf bewährten Simulationen diskret-ereignisorientierter Kommunikation mit Publish/Subscribe und der Umsetzung mit REBECA und PEERSIM auf [169].

5.3.2 Simulation des Netzwerks

Der Aufbau des Netzwerks erfolgt innerhalb des Initiierungsprotokolls, dies beinhaltet die Zuweisung des initialen Zufallswertes (*Seed*). Bei jedem Experiment werden 100 Netzwerkknoten erzeugt, welche anschließend zufällig miteinander verbunden werden, so dass ein zyklenfreier Graph in Baumstruktur entsteht. Aufbauend auf diese Struktur erhält jeder Knoten einen Broker. Daraufhin werden die Anwendungskomponenten an die Broker verteilt. Je nach initialer Verteilung werden einem Broker keine, eine oder mehrere Komponenten zugewiesen. Die Anzahl der Knoten und der möglichen Ausführungsorte ist bis auf Weiteres auf 100 begrenzt. Mit dieser Konfiguration ist ein ausreichend großes Testumfeld sichergestellt und das Umfeld intelligenter Umgebungen, wie bspw. einer Haussteuerung, wird wiedergegeben. Mit dem Aufbau der Netzwerkbroker, ihren Kommunikationspfaden sowie der Zuweisung der Broker und Anwendungen ist die unterste Ebene der Netzwerkschicht modelliert, auf den darauf aufbauenden Schichten finden das Routing (Nachrichtenweiterleitungsschicht) und die Weiterleitung von Nachrichten (Nachrichtentransportschicht) statt. Die Broker und ihre Verbindungen untereinander werden als M/M/1-Warteschlangen mittels dem First In - First Out (FiFo) - Prinzip mit einer zufälligen Verzögerung implementiert. Die Kontrolle des Nachrichtenaustauschs zwischen den Brokern und die Verwaltung der Verbindungen obliegt der Nachrichtentransportschicht als oberster Netzwerkschicht und wird über Transportprotokolle abgedeckt.

5.3.3 Simulation der Middleware und der Platzierungsmechanismen

Die Publish/Subscribe-Funktionalität wird in der Middleware implementiert. Sie übernimmt vielfältige Aufgaben wie bspw. das Routing und Einspeisen von Nachrichten und Ankündigungen. Mit der Vielzahl ihrer Aufgaben ist ebenfalls eine Vielzahl von PEERSIM-Protokollen für die Simulation nötig. Kernstück ist das *Brokering*-Protokoll (an dieser Stelle wird das *SimpleBrokering*-Protokoll genutzt), welches alle weiteren Protokolle des Brokers steuert und die Funktionalitäten der jeweiligen Broker-Plugins umsetzt. Für die Simulation werden primär vier Middlewarefunktionalitäten verfügbar sein. Das sind zum einen (1) die Weiterleitung und Verarbeitung empfangener Nachrichten, (2) die Ausführung der eigentlichen Publish/Subscribe-Funktionalität wie das Propagieren, Routing und Matching von Advertisements, Subscriptions und Notifications, (3) die Bereitstellung von Funktionalität für Anwendungskomponenten sowie (4) die Sammlung von Daten zur Bereitstellung von Statistiken [169].

Die Middlewarefunktionalitäten und die Nachrichtenweiterleitung werden durch eine Vielzahl von Protokollen implementiert. Diese umfassen *Event Processing* in PeerSim (zur schrittweisen Verarbeitung von Notifikationen, Subskriptionen und Ankündigungen) mit den Funktionalitäten von Rebeca: *Event Matching* (Matching von Subskriptionen und Notifikationen), *Event Routing* (Weiterleitung von Notifikationen, Subskriptionen und Ankündigung) und *Event Advertising* (Ankündigung). An dieser Stelle sei angemerkt, dass die in dieser Arbeit vorgestellten Experimente auf der Grundlage des identitätsbasierten Routings durchgeführt wurden. Die Bereitstellung der Statistiken erfolgt durch die Ausführung von *Measuring-* sowie *Statistic*-Protokollen. Die Einbindung der Anwendungskomponenten an die Broker als Bestandteil der Middlewarefunktionalität erfolgt über die *BasicComponentEngine*.

Anwendungsplatzierungsanpassung: Migration und Replikation. Die Umsetzung der laufenden Anwendungsverteilung erfolgt durch das Zusammenspiel der Middlewarekomponenten von Rebeca und der PeerSim-Protokolle. Der Ablauf sieht vor, dass zunächst Daten über den Ereignisfluss von der *Monitoring*-Komponente gesammelt und von der *StatisticEngine* ausgewertet werden. Dies erfolgt innerhalb von Rebeca im Zusammenhang mit dem *SimpleMeasuring*-Protokoll als PeerSim-Komponente. Innerhalb der Statistik der Middleware wird die Entscheidung getroffen, ob eine Migration oder eine Replikation mit einer anschließenden Migration ausgeführt wird. Damit sind durch die *Monitoring-* und *Statistic*-Komponenenten und *-engines* die Voraussetzungen für die Ausführung der adaptiven Anwendungsplatzierung gegeben.

Die Migrationsausführung erfolgt in Rebeca mit der Klasse *BasicMigrationEngine* und die Replikationsausführung durch die zusätzliche Klasse *BasicReplicationEngine*. Neben den eigentlichen Engines aus Rebeca sind für die Durchführung von Simulationen mit PeerSim weitere Protokolle notwendig. Eine Übersicht über die zur Durchführung der Simulation notwendigen Klassen findet sich im linken Teil der Abbildung 5.1. Die Abbildung wurde bereits im Kapitel 4 als Abbildung 4.5 gezeigt und wird an dieser Stelle erneut platziert. Die vier zusätzlichen Protokolle zur Simulation lauten *ProtocolEngine*, *SimpleMigrating*, *SimpleReplicating* und *SimpleStatistic*. Das „Simple“ bedeutet, dass hier die grundlegenden Eigenschaften des vorgestellten Ansatzes implementiert wurden und spätere Erweiterungen in Zusammenhang mit den in Rebeca umgesetzten Arbeitsweisen möglich sind. Die hier genannten vier Protokolle simulieren das Verhalten einer tatsächlich vorhandenen Middleware als ein Peer-to-Peer-Netzwerk. Die jeweiligen Aufgaben des Protokolls ergeben sich aus der Bezeichnung der jeweiligen Klassen. So sind die *ProtocolEngine* für die simulative Ausführung des Netzwerkbrokers, *SimpleMigrating* für die Ausführung der Migration, *SimpleReplicating* für die Ausführung der Replikation und *SimpleStatistic* für die Sammlung der Daten für eine spätere Replikations- oder Migrationsentscheidung verantwortlich.

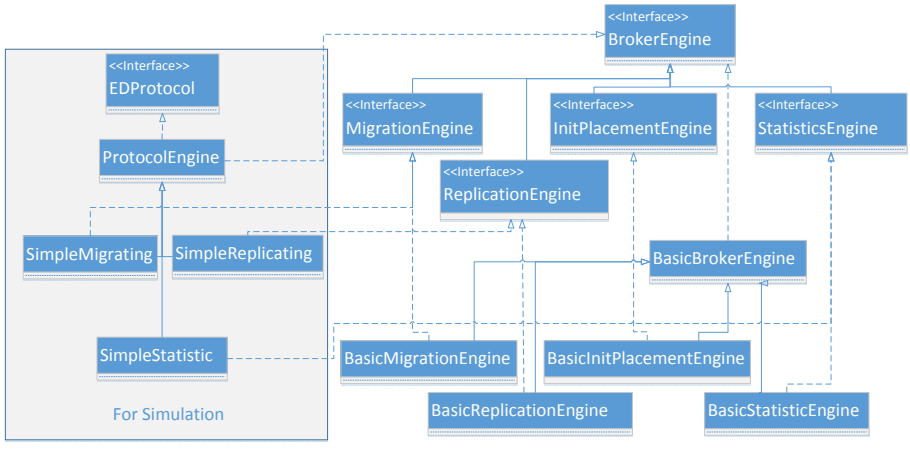


Abbildung 5.1: Implementierung der Engines

5.4 Experimente

Mit den im Folgenden beschriebenen Experimenten wird nachgewiesen, dass durch die adaptive Verteilung von Anwendungskomponenten zur verteilten Detektion komplexer Ereignismuster, bei der Verringerung von Kommunikationsaufwand signifikant Kosten und Energie eingespart werden. Dies ist notwendig, da im Umfeld intelligenter Umgebungen mit mobilen Geräten Energieressourcen limitierende Faktoren sind und Kommunikation maßgeblichen Einfluss auf die Energiekonsumption hat. Die Optimierung erfolgt gemäß der in Abschnitt 3.5.4 beschriebenen Zielfunktion unter der Voraussetzung, dass bei einer Platzierungsanpassung die zusätzlichen Kosten für die Platzierungsanpassung selbst geringer sind als die durch die Anpassung erzielten Einsparungen bei der Anwendungsausführung. Die Neuplatzierung erfolgt immer unter der Maßgabe, dass die durch den Anwender und Entwickler gestellten Anforderungen auch nach einer Platzierungsänderung eingehalten werden. Die Experimente bauen auf eine erfolgreiche initiale Anwendungsplatzierung in eingeschwungenem Zustand auf. Der Nachweis der Vorteilhaftigkeit des Ansatzes wird als Ergebnis der laufenden Platzierung betrachtet. Die initiale Platzierung dient als Ausgangspunkt der laufenden Optimierung und wird je nach Experimentziel variiert. Durch diese Variationen werden unterschiedliche initiale Anwendungsverteilungen simuliert.

Die Middleware wird auf einem Netzwerk von azyklisch miteinander verbundenen Brokern ausgeführt. Die Broker kommunizieren via Kommunikationskanälen, die mit ihrem Verhalten nicht im Mittelpunkt der Simulation stehen. Die Latenzen und Kapazitäten sind aus diesem Grund einheitlich gestaltet.

Bei den Experimenten wird auf die Basisoperationen Migration und Replikation (mit anschließender Migration) zurückgegriffen, auf die explizite Ausführung von Dekomposition und Rekombination wird dagegen verzichtet. Stattdessen werden

die Anwendungen bereits von Anfang an komponentenweise ausgeführt. Damit sind die Experimente nicht abhängig von den Vorstellungen des Anwendungsentwicklers und stellen die maximalen Optimierungsmöglichkeiten des implementierten Verteilungsansatzes in der jeweiligen Umgebung dar.

Die Experimente bauen auf der Middleware REBECA und dem PEERSIM-Simulator auf, wobei die Middleware und der Simulator direkt miteinander interagieren und die Implementierung zur Anwendungsverteilungsanpassung direkt genutzt wird. Demzufolge greifen die Experimente auf die gesamte Middleware und deren Dienste zurück. Die Middleware besteht aus Brokern, die auf bestimmte Eigenschaften bezüglich ihrer verfügbaren Ressourcen zugreifen. Dieses Ausführungspotential unterscheidet sich je nach Experiment. Auch die Anwendungen sind bezüglich ihrer Ansprüche an den Ausführungsort verschieden. Die unterschiedlichen Ressourcenanforderungen werden ebenfalls in den Experimenten abgebildet. Die Entscheidungsfindung, ob, wann bzw. welche Basisoperationen durchgeführt werden, erfolgt lokal. In den Experimenten wird ein vereinfachtes Kosten- und Kräftemodell verwendet, bei dem lediglich die Weiterleitungskosten und die Platzierungskosten die auf die Anwendungsplatzierung einwirkenden Kräfte formen. Für die Ermittlung der Weiterleitungskosten genügt es, die Anzahl der ein- und ausgehenden Nachrichten zu zählen. Dies ermöglicht eine Auswertung mittels lokal geführten Statistiken über die ein- und ausgehenden Nachrichten. Die Statistiken werden durch die Broker geführt und regelmäßig ausgewertet. An die Auswertung gekoppelt ist eine Ausführungsstrategie, die bestimmt, wann welche Basisoperation ausgeführt wird.

Die Experimente werden mit einer unterschiedlichen Anordnung von Anwendungskomponenten, sprich Ereignisdetektoren, durchgeführt. Die Detektoren erzeugen neue Ereignisse bzw. filtern Ereignisse heraus, sodass der Ereignisstrom entweder verringert oder verstärkt wird. Diese Verringerung bzw. Verstärkung wird im Folgenden als *Amplification* bezeichnet. Die Abarbeitung von den Ereignisproduzenten (*Producer*) hin zu den Ereigniskonsumenten (*Consumer*) erfolgt entlang der Verarbeiter (*Worker*). Während die Consumer und Producer an einer festen Stelle im Netzwerk deployt werden und während der Laufzeit nicht verschoben werden können, kann die Platzierung der Worker zur Laufzeit angepasst werden. Während die Platzierung von Consumern und Producern am Rand des Netzwerks angesiedelt ist, finden sich die Worker im inneren Bereich des Netzwerks. Wie bzw. nach welcher Verteilung die Worker initial organisiert sind, wird individuell für jedes Experiment festgelegt. Die Worker sind in Ketten (*Chains*) organisiert, womit die Abarbeitungsschritte der Bearbeitung komplexer Ereignismuster nachgebildet werden. Die Ketten haben unterschiedliche Muster, bspw. 1-1-1 oder 3-1-3. Das bedeutet, dass im ersten Fall die Ereignisströme nacheinander von einem Producer zu einem Worker weitergeleitet, dort verarbeitet und an einen Consumer gesendet werden. Im zweiten Fall existieren drei Producer und erzeugen Ereignisse. Diese werden an einen Worker weitergeleitet und verarbeitet, der die Ereignisse anschließend an drei Consumer sendet. Bei den Experimenten sind, wenn nicht anders angegeben, stets insgesamt 50 Chains bei einer Netzwerkgröße von 100 Knoten zu verteilen.

Kennzahlen. Zur Evaluierung des in dieser Arbeit vorgestellten Ansatzes werden im Folgenden zahlreiche Experimente gezeigt, die sowohl die Sinnhaftigkeit des Ansatzes im Allgemeinen, als auch das Verhalten bei unterschiedlichen Umgebungsbedingungen beleuchten. Die Evaluierung des Ansatzes erfolgt anhand definierter Kennzahlen.

Gemäß dem Ziel, den Gesamtkommunikationsaufwand zu minimieren, werden die Nachrichten innerhalb des gesamten Netzwerks gezählt. Zu diesen Nachrichten gehören sowohl die Ereignisnotifikationen, also die Nutzdaten/Nutzlast, als auch die Steuerungs- und Managementnachrichten, welche damit den Overhead repräsentieren. Diese Nachrichtengruppen lassen sich separat erfassen.

Experimentbeschreibungen. Es finden insgesamt vier verschiedene Gruppen von Experimenten statt. Die Ergebnisse werden gruppenweise zusammengefasst und dargestellt. Je Gruppe werden die einzelnen Experimentläufe vielfach wiederholt. In den folgenden Beschreibungen zu den jeweiligen Experimenten werden die zugehörigen Experimentziele, die Kriterien der Messung, die getroffenen Annahmen, der Aufbau und der Ablauf des Experiments beschrieben. Weiterhin gehören zur Experimentbeschreibung die Erwartungen und die tatsächlichen Ergebnisse inklusive ihrer Interpretation.

5.4.1 Initiale Platzierung

Die initiale Platzierung von Anwendungskomponenten beeinflusst die Ressourcenauslastung eines Netzwerks signifikant in der Hinsicht, dass eine ungünstige Platzierung von Anwendungskomponenten zwischen Produzenten und Konsumenten von Ereignissen zu höherem Verbrauch der betrachteten Ressourcen führt als bei einer verbesserten Platzierung. Grob gesagt: Liegen die Konsumenten und Produzenten auf der linken Seite eines Netzwerks und sämtliche Anwendungskomponenten auf der rechten Seite, werden während der Ausführung zunächst alle Nachrichten von links nach rechts und anschließend wieder auf die linke Seite gesendet. Dies verbraucht selbstverständlich mehr Weiterleitungsressourcen, als wenn die Anwendungen ebenfalls auf der linken Seite oder zumindest in der Nähe der Konsumenten oder Produzenten positioniert sind. Für die initiale Platzierung existieren, wie in dieser Arbeit beschrieben wurde, verschiedene Techniken, um bereits mit der initialen Platzierung gute Ergebnisse zu erreichen. Die initiale Platzierung ist nicht Ziel der Betrachtung der Experimente, sondern dient lediglich dazu, eine gültige Platzierung zu finden, auf die die laufende Platzierung aufsetzt. Daher wird die initiale Platzierung als Ausgangspunkt für Verbesserungen simuliert. Dies geschieht anhand der gewählten Verteilung von Anwendungskomponenten im Netzwerk, wobei das Netzwerk in seinem Grundaufbau gleich bleibt. Bei der Variation der Platzierung stehen zwei unterschiedliche Varianten zur Auswahl, die Gleich- und die Normalverteilung. Welche dieser statistischen Verteilungstechniken bei den Experimenten genutzt wird, ist der jeweiligen Experimentbeschreibung zu entnehmen.

5.4.2 Laufende Optimierung

Ziel. Das Experiment zur laufenden Optimierung dient dem Nachweis, dass der Einsatz von Migration und Replikation von Anwendungskomponenten Nachrichtenvolumen einsparen und damit global Kosten senken kann.

Annahmen. In diesem grundlegenden Fall wird von Komponenten, d.h. Brokern und Ausführungscontainern ausgegangen, welche zwar begrenzte, aber dennoch stets ausreichende Ressourcen zur Verfügung stellen. Die Anwendungen liegen bereits in Komponenten gegliedert vor und können unabhängig von Dekomposition im Netzwerk migriert und repliziert werden. Auch Rekombinationen sind möglich, wenn gleich arbeitende Detektoren an einem Broker zusammengefasst werden und eine Komponente entsteht, deren Ereignisfilter beide vorherigen Filter überdeckt.

Ablauf. Zunächst wird ein azyklisches Netzwerk aufgebaut, wobei auf jedem Knoten ein Middlewarebroker platziert wird und auf jedem Knoten ein Ausführungscontainer mit gleicher, ausreichender Kapazität bereitsteht. Nach dem Aufbau des Brokernetzwerks und der Middleware erfolgt die Verteilung der Anwendungskomponenten, also der Producer, Consumer und Worker zufällig anhand einer Gleichverteilung. Anschließend werden die Anwendungskomponenten ausgeführt, die Worker verarbeiten die Ereignisse.

Variation. Die Ausführung des Experiments erfolgt in unterschiedlichen Varianten. Diese unterscheiden sich in der Anzahl und der Anordnung von Producern und Consumern, wobei sich die Worker stets frei im Netzwerk bewegen können. Die Chains variieren zwischen den Experimenten. Es werden Experimente mit den Chains 1-1-1, 1-1-2, 1-1-3, 2-1-1, 2-1-2, 3-1-1 und 3-1-3 durchgeführt. Die jeweiligen Experimente werden für jede Chain unter Variation der Verstärkungen der Worker ausgeführt. Die Verstärkungen reichen von 0,3 bis 1000.

Erwartung. Die Erwartung bei der Durchführung und Auswertung der Experimente besteht darin, dass im Vergleich zur nicht-optimierten Variante bei der optimierten Variante weniger Nachrichten im Netzwerk vorhanden sind. Zusätzlich ist zu erwarten, dass durch den Einsatz der Replikation zusätzlich Nachrichten eingespart werden. Allerdings ist auch davon auszugehen, dass sich die Anzahl der Nachrichten von Migration und Replikation bei steigender Verstärkung wieder angleichen. Die Worker können ab einem gewissen Punkt keine weiteren Verbesserungen mehr erreichen, weil sie nicht mehr weiter in Richtung der Consumer migriert werden können. Dies gilt genauso für das andere Extrem, bei sehr geringer Verstärkung, also bei starkem Filtern. In diesem Fall können die Worker nicht mehr weiter als bis zum Producer wandern. Am Ende der Experimentlaufzeit halten sich die Worker in Abhängigkeit von der Verstärkung entweder so weit wie möglich in der Nähe der Producer oder der Consumer auf.

Ergebnisse und Diskussion. Die Ergebniscarts dieses Experiments entsprechen einem einheitlichen Muster. Das Diagramm in Abbildung 5.2 zeigt die Anzahl der Nachrichten (Name/Wert-Ereignisse) an der Ordinatenachse und die unterschiedlichen Verstärkungen des Ereignisstroms durch die Worker an der Abszissenachse. Die Werte der Abszissenachse sind als Werte x je 100 zu interpretieren, somit entspricht der Wert der Verstärkung mit 10^2 einer Verstärkung von 1. Links davon werden die Ereignisströme negativ verstärkt, damit wird das Herausfiltern von nicht benötigten Ereignissen simuliert. Rechts des Abzissenwerts von 10^2 wird die Anzahl der Ereignisse nach der Bearbeitung durch einen Worker erhöht.

Die abgebildeten Kurven zeigen die jeweilige Zusammenfassung einzelner Experimente und entsprechen damit dem Durchschnitt der Ergebnisreihen. Bei den Ergebnislinien spiegelt die schwarze Linie die Messergebnisse ohne den Einsatz von Migration und Replikation wider, während die blaue Linie die gemittelten Ergebnisse unter dem Einsatz Migration zeigt. Die rote Linie fasst die Ergebnisse der Migration und Replikation zusammen.

Chain: 1-1-1. Bei diesem Experiment befindet sich eine Vielzahl von Chains im Aufbau 1-1-1 im Netzwerk. Das bedeutet, dass ein Consumer und ein Producer über genau eine Workerchain miteinander verbunden sind. Die Worker verstärken die Ereignisströme dabei mit unterschiedlicher Intensität. Wie in

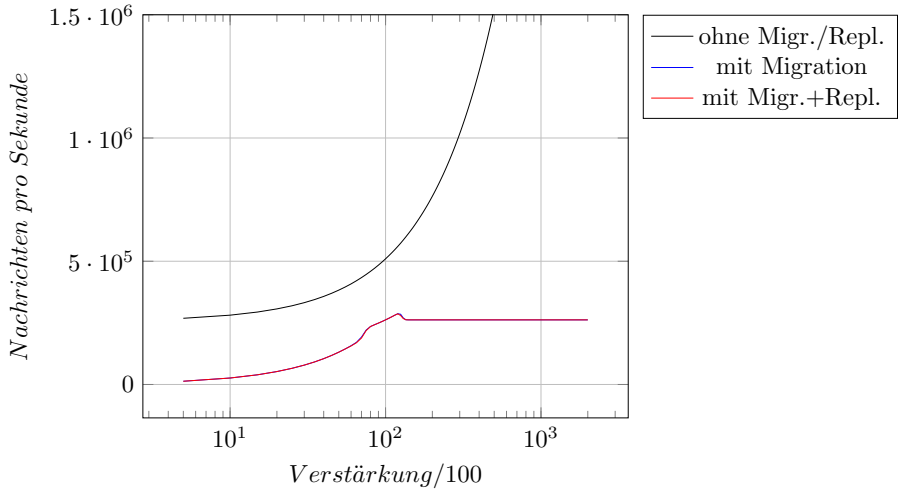


Abbildung 5.2: Optimierung einer 1-1-1 Chain

Abbildung 5.2 zu sehen ist, steigt die Anzahl der Nachrichten ohne Migration und Replikation mit wachsender Verstärkung. Die Kurve zeigt ein exponentielles Wachstumsverhalten. Dies ist durch die Kombination von festem Verstärkungsfaktor und der Vielzahl an zu bewältigenden Schritten vom Worker bis zum Con-

sumer durch das Netzwerk zu begründen. Die Kurven für die optimierte Variante mit Migration sowie für Migration und Replikation liegen direkt übereinander. Dies liegt daran, dass bei einer 1-1-1 Chain keine Replikationen notwendig bzw. sinnvoll sind und lediglich Migrationen ausgeführt werden. So bewegen sich die Worker bei einer Verstärkung < 1 in Richtung der Consumer und bei einer Verstärkung > 1 in Richtung der Producer. Sind diese Anpassungen einmal vollzogen, die Worker also entsprechend so weit wie möglich in Richtung der Producer bzw. Consumer gewandert, ist keine weitere Platzierungsveränderung möglich. Da eine Verstärkung der Nachrichtenströme durch die Worker erfolgt, findet keine weitere Verstärkung mehr statt, sobald die Worker auf die Positionen der Broker gewandert sind, an dem sich auch die Producer befinden. Daher findet ab einer Verstärkung von mehr als 1 keine weitere Steigerung der Anzahl von Nachrichten im Gesamtsystem mehr statt. Des Weiteren ist zu erkennen, dass die optimierten Varianten zwar deutliche Kosteneinsparungen bewirken, jedoch bis zu einer Verstärkung von 1 in etwa parallel zur nicht-optimierten Kurve verlaufen. Dies ist damit zu erklären, dass die Worker bei einer Verstärkung < 1 in Richtung der Producer migriert werden, sowie bei einer Verstärkung > 1 in Richtung der Consumer. Damit werden, je nach Verstärkungssituation, die Ereignisströme nicht weiter verringert (Verstärkung < 1) bzw. weiter erhöht. Bei einer Verstärkung von > 1 befinden sich die Worker auf dem Broker, auf dem sich auch der Consumer befindet, daher steigt die Anzahl der Nachrichten in diesem Fall sowie bei einer weiteren Verstärkung nicht weiter an.

Wie in der Abbildung 5.2 jedoch zu erkennen ist, decken sich die Kurve von Migration sowie die Kurve von Migration mit Replikation rund um die Verstärkung von 1 nicht vollständig. Dieses Verhalten lässt sich damit erklären, dass für Migration und Replikation unterschiedliche Schwellwerte gelten und mehr Platzierungsentscheidungen zu treffen sind. Die Anpassung erfolgt im Endeffekt jedoch auf das gleiche Niveau. Bis zur Verschiebung der Worker auf die Position der Consumer verlaufen die optimierten Kurven in etwa parallel zur nicht-optimierten Variante. Nach der Verstärkung nahe 1 schwingt sich die Anzahl der Nachrichten nach erfolgter Anpassung und vorübergehend höherer Nachrichtenanzahl auf einem stabilen Niveau ein. Der stärkere Anstieg rund um den Wert 1 liegt daran, dass die Anpassung bei bestimmten Schwellwerten anspringt und rund um diese Phase der Algorithmus die Worker nicht immer eindeutig einem bestimmten Ausführungsort zuordnet.

Chain: 1-1-2. Bei diesem Experiment wird mit einer 1-1-2 Chain gearbeitet, also einem Producer, einer Workerchain und zwei Consumern. Dies spiegelt ein Ereignismuster wider, welches von zwei Konsumenten angefordert wird.

Zunächst einmal liegen die Kurven der Ergebnisse von Migration sowie Migration und Replikation genau übereinander. Die Darstellungen der nicht-optimierten Variante und der optimierten Varianten verlaufen dabei in etwa parallelen Kurven, die nicht-optimierte Variante verläuft allerdings insgesamt auf einem deutlich höheren Niveau. Es werden Nachrichten und damit Kosten in erheblichem

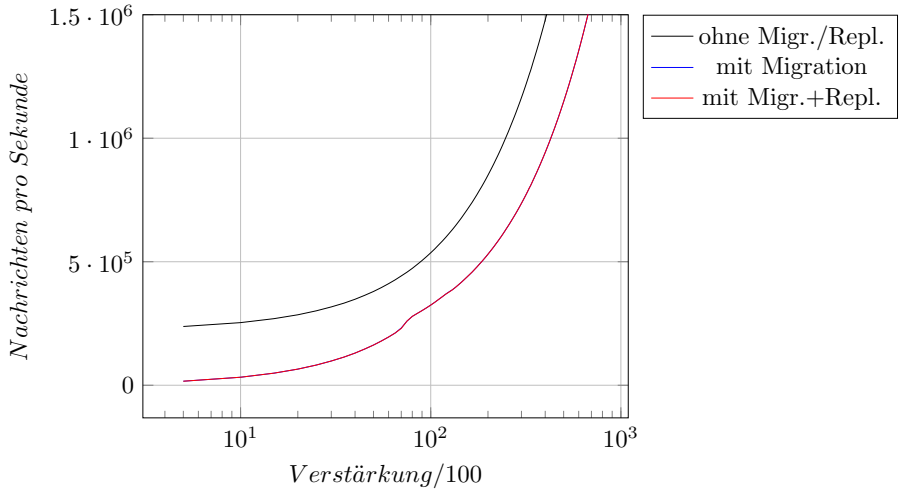


Abbildung 5.3: Optimierung einer 1-1-2 Chain

Umfang eingespart. Bei diesem Experiment, im Gegensatz zum vorigen Teilexperiment mit der 1-1-1 Chain (siehe Abschnitt 5.4.2), verlaufen die optimierten Kurven über den gesamten Zeitverlauf in etwa parallel zur nicht-optimierten Kurve. Der Kurvenverlauf gleicht bis zur Verstärkung von 1 dem des vorherigen Experiments. In diesem Teil migrieren die Worker in Richtung der Producer. Die Anzahl der Nachrichten verringert sich entsprechend und sinkt auf ein niedrigeres Niveau. Bei einer Verstärkung > 1 ist jedoch ein anderes Verhalten als zuvor bei der 1-1-1 Chain zu beobachten. Zwar liegt die Anzahl der Nachrichten bei beiden Optimierungsvarianten zu jeder Zeit unter der nicht-optimierten Kurve, allerdings verharren sie nicht auf einem Plateau. Dies liegt darin begründet, dass die Worker nicht komplett zu den Consumern migriert werden, sondern an der Gabelung, an dem sich die Ereignisströme in Richtung der Consumer verzweigen, verbleiben. Eine weitere Migration kann an dieser Stelle nicht stattfinden, da kein (Ausgangs-) Strom eindeutig überwiegt. Allerdings wird auch keine Replikation und anschließende Migration vorgenommen. Eine Replikation der Worker in Richtung der Consumer würde bedeuten, die benötigten Eingangsströme des Brokers in unterschiedliche Richtungen weiterzuleiten. Allerdings werden bei einem Publish/Subscribe-System grundsätzlich die publizierten (und ggf. angekündigten) Notifikationen an alle (potentiell) interessierten Empfänger im gesamten Netzwerk verbreitet. Damit würden, trotz Aufspaltung, beide Consumer die Notifikationen auch von dem Worker erhalten, der auf dem Broker mit dem anderen Consumer ausgeführt wird. Statt einer Einsparung würden noch mehr Notifikationen im Netzwerk verschickt. Eine mögliche Ersparnis würde ins Gegenteil umschlagen. Eine Möglichkeit zur Lösung dieses Problems besteht darin, den Ergebnisereignisraum aufzuteilen. Dazu müssten jedoch während der Replikation in Richtung der Consumer sowohl die Ergeb-

nisbeschreibung der Worker (Publikationen und Advertisements), als auch die Subskriptionen der Consumer angepasst werden. Dieser schwere Eingriff in das Publish/Subscribe-Kommunikationsparadigma ist jedoch zu unterbinden. Es ist zu zeigen, dass das hier vorgestellte Optimierungsverfahren ohne Anpassungen des Publish/Subscribe-Kommunikationsparadigma möglich ist. Durch die fehlende Aufspaltung in Richtung der Consumer verbleibt der Worker an der Gabelung in Richtung beider Consumer und verstärkt entsprechend den Notifikationsstrom. Durch die Weiterleitung über weitere Broker steigt die Anzahl der Nachrichten entsprechend an.

Aufgrund der fehlenden Replikation in Richtung der Consumer verhalten sich die Kurven beider Optimierungsvarianten gleich, da lediglich eine Migration in Richtung der Producer stattfindet. Es existieren allenfalls leichte Abweichungen bei einer Verstärkung von $0,8$. Dies lässt sich wiederum durch den erhöhten Auswertungsaufwand für die Migration im Verbund mit der Replikation und statistische Effekte erklären.

Chain: 2-1-1. Bei diesem Experiment wird mit einer 2-1-1 Chain gearbeitet, demnach mit zwei Producern, einer Workerchain und einem Consumer. Diese Art Experiment ermöglicht erstmals die Ausführung von Migrationen in Kombination mit der Replikation.

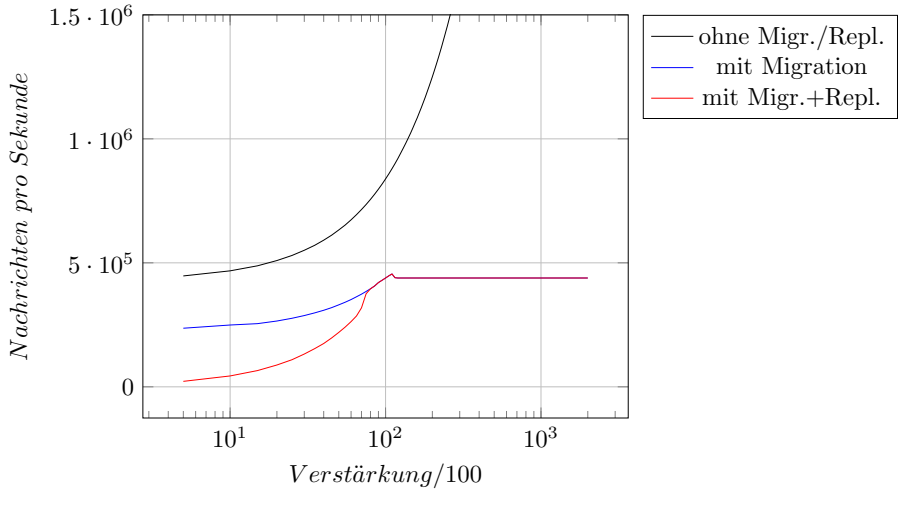


Abbildung 5.4: Optimierung einer 2-1-1 Chain

Wie Abbildung 5.4 zeigt, lässt die Auswertung wiederum die Unterteilung der Ergebnisse in eine Phase mit der Verstärkung < 1 und > 1 zu. Wie schon beim Teilexperiment mit der Chain 1-1-1 (siehe Abschnitt 5.4.2) wird bei einer Verstärkung > 1 ein Plateau erreicht, bei dem die Anzahl der Nachrichten im System nicht mehr zunimmt, dies gilt sowohl für die Migration als auch für die

Migration im Zusammenspiel mit der Rekombination. Allerdings wird das Plateau erst mit einer gewissen Verzögerung nach der Verstärkung von 1 erreicht, dieses Verhalten lässt sich durch die Verzögerung der Anpassung aufgrund unterschiedlicher Schwellwerte herleiten. Ab einer Verzögerung von 1 und mehr migrieren die Worker in Richtung der Consumer, befinden sich die Worker auf den gleichen Brokern wie die Consumer, findet keine das Netzwerk belastende Verstärkung mehr statt, was in der Abbildung verdeutlicht wird.

Ein bisher nicht beobachtetes Verhalten ist in der Phase bei einer Verstärkung von < 1 zu sehen. Sowohl die Migration, als auch die Migration im Zusammenhang mit der Replikation sparen in beträchtlichem Maß Notifikationen ein. Allerdings nähern sich die Kurven bei zunehmender Verstärkung an und liegen bei einer Verstärkung von etwa 0,8 auf gleichem Niveau. Je kleiner dabei die Verstärkung ist, desto mehr Ereignisnotifikationen werden eingespart, sodass die höchste Einsparung bei geringer Verstärkung durch die Migration und Replikation erreicht wird. Bei diesem Verfahren wird der Worker zunächst solange in Richtung der Producer verschoben, bis beide Notifikationen gemeinsam weitergeleitet werden. An dieser Nahtstelle erfolgt die Replikation der Worker und eine weitere Migration, so nah wie möglich in Richtung der Producer. Je früher der Notifikationsstrom verringert wird, desto weniger Nachrichten müssen anschließend durch das Netzwerk geleitet werden, daraus resultiert die vergleichsweise große Differenz zwischen beiden Optimierungsvarianten. Je mehr anschließend die Verstärkung (bis zu einem Wert von 1) wächst, desto weniger Auswirkungen hat die frühzeitige Reduktion von Ereignisströmen. Schon die Migration ist bei höherer Verstärkung nicht mehr geeignet, die Nachrichtenanzahl zu senken. Auch eine Replikation bringt im Zusammenhang mit der Migration nur noch eine kleine Verbesserung. Im Unterschied zur 1-1-2 Chain (siehe vorheriger Abschnitt 5.4.2) wird bei der Replikation in Richtung der Producer der Ereignisraum der eingehenden Ereignisse jeweils beschränkt. Dies geschieht bei der Replikation der Worker, indem beim Aufspalten der Worker zunächst die Eingangsströme und der Ereignisraum analysiert werden. Bei der Aufspaltung subskribieren sich die neu gebildeten Worker lediglich für die ihnen zugeordneten Bereiche, nachgeordnete Worker, Verarbeiter bzw. Interessenten der Nachrichten müssen nicht angepasst werden.

Chain: 2-1-2. Bei diesem Experiment wird mit einer 2-1-2 Chain gearbeitet, demnach mit zwei Producern, einer Workerchain und zwei Consumern. Diese Art Experiment ermöglicht sowohl die Ausführung von Migrationen, als auch die Kombination aus Migration und Replikation.

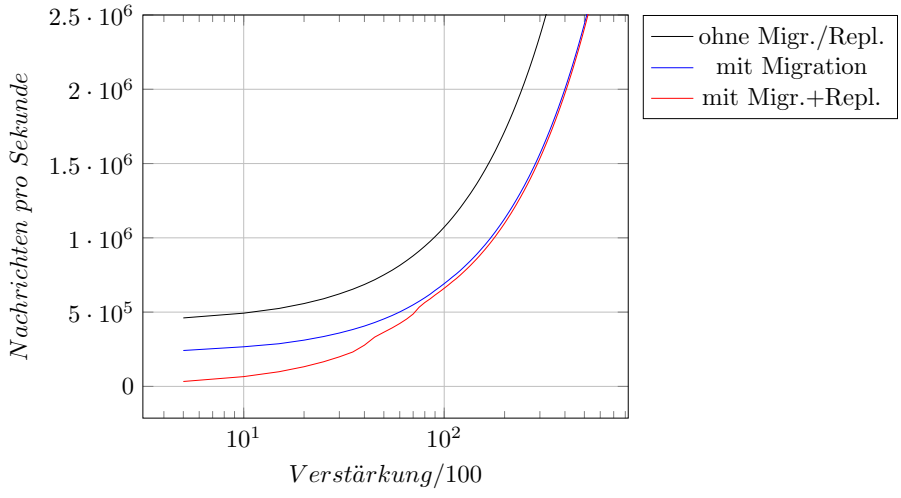


Abbildung 5.5: Optimierung einer 2-1-2 Chain

Abbildung 5.5 gleicht einer Mischung der beiden vorangegangenen Abbildungen. Während der erste Teil der Darstellung bis zu einer Verstärkung von 1 der Abbildung 5.4 gleicht, ähnelt der rechte Teil dieser Abbildung der Abbildung 5.3. Allerdings verlaufen die Kurven im rechten Teil von Abbildung 5.5 auf einem vergleichsweise höheren Niveau. Auch bei dieser Abbildung sind im ersten Teil (links der Verstärkung von 1) die Migration und Migration mit Replikation der nicht-optimierten Kurve in Bezug auf die Netzwerklast überlegen. Auch hier spielt die Migration mit Replikation durch die Replikation der Worker zu den Consumern bei einer Verstärkung < 1 ihre Vorteile aus. Mit zunehmender Verstärkung verhalten sich die beiden optimierten Varianten zunehmend ähnlich und liegen auch hier bei einer Verstärkung von etwa 0,8 auf annähernd gleichem Niveau. Bei einer Verstärkung > 1 hingegen laufen beide Optimierungskurven gemeinsam und unterhalb der nicht-optimierten Verteilung, allerdings mit gleicher Steigung. Die Platzierung in diesem Teil ist wieder gleich der Platzierung in Abbildung 5.3, bei der keine Replikation der Worker in Richtung der Consumer möglich ist und die Worker nur bis zur Verzweigung der beiden Consumer migriert werden können.

Chain: 1-1-3. Bei diesem Experiment wird mit einer 1-1-3 Chain operiert, also jeweils einem Producer, einer Workerchain und drei Consumern.

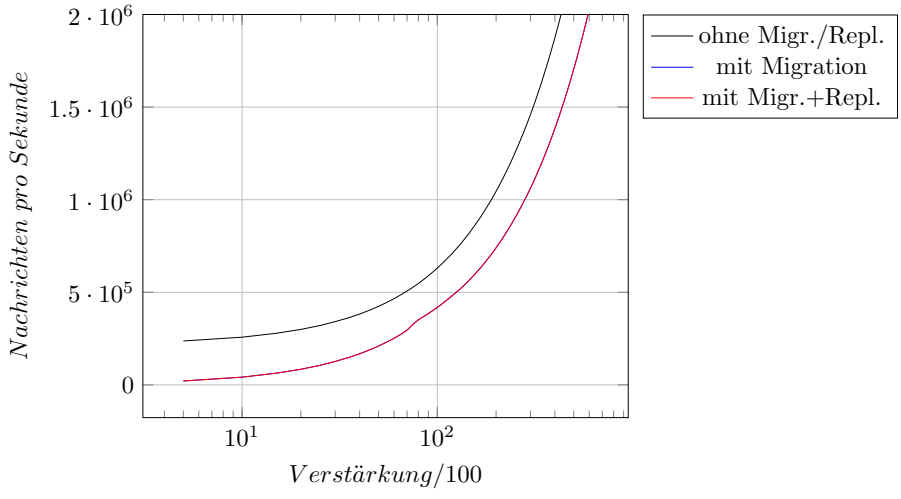


Abbildung 5.6: Optimierung einer 1-1-3 Chain

Die Ergebnisse entsprechen bis zu einer Verstärkung von 1 denen des Experiments mit einer 1-1-1 Chain aus Abschnitt 5.4.2. Beide optimierten Varianten verhalten sich konsistent. Bei einer Verstärkung von < 1 werden die Worker in Richtung der Producer migriert, so wird die jeweils größtmögliche Abschwächung der Anzahl von Nachrichten erreicht. Bei einer Verstärkung von > 1 migrieren die Worker lediglich bis zum Gabelungspunkt der Consumer. Eine zusätzliche Replikation in Richtung der Consumer findet nicht statt. Der zunächst lineare Anstieg der Verstärkung des Nachrichtenstroms an den Workern bewirkt mit der noch zu überwindenden Distanz zwischen Workern und den drei Consumern einen deutlich steileren als linearen Anstieg der Nachrichtenanzahl. Beide optimierten Kurven verlaufen parallel mit etwa gleichem Abstand zur Kurve ohne Optimierung. Der „Knick“ der optimierten Platzierung bei einer Verstärkung von ca. 0,8 resultiert wiederum aus dem Zusammenspiel der Gegengewichtskraft, welche die Platzierungskosten widerspiegelt.

Chain: 3-1-1. Dieses Optimierungsexperiment nutzt eine Chain in der Form 3-1-1, also drei Producer und jeweils einen Worker und einen Consumer.

Die Ergebnisse dieses Experiments in Abbildung 5.7 decken sich mit den Erwartungen, welche die Ergebnisse des Experimentsetups der Chain 2-1-1 vermuten lassen (siehe Abschnitt 5.4.2 und Abbildung 5.4). Auch bei der nun vorliegenden Abbildung ist das Ergebnis zweigeteilt. Sind bis zu einer Verstärkung von etwa 0,8 beide Optimierungsvarianten unterschiedlich vorteilhaft, nähern sie sich von links kommend einander an, wobei die zusätzliche Replikation höhere Ein-

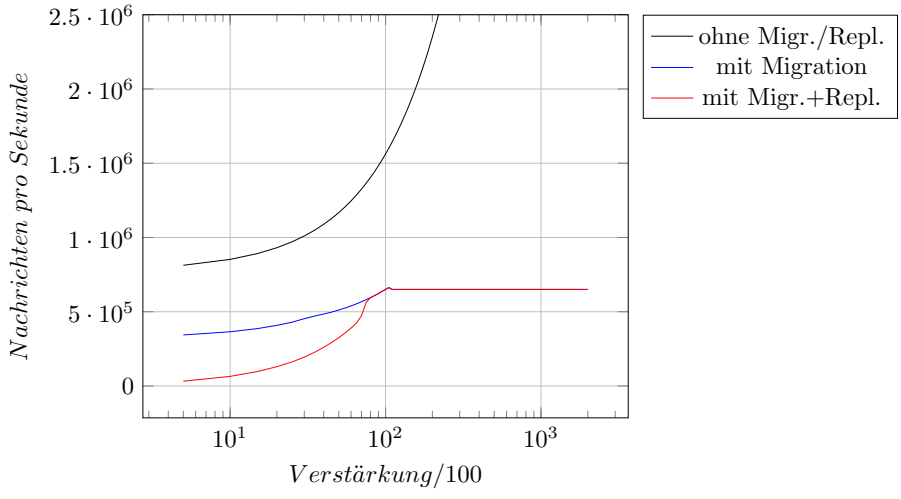


Abbildung 5.7: Optimierung einer 3-1-1 Chain

sparungen ermöglicht. Durch die Replikation und die anschließende Migration migrieren die Worker so nah wie möglich an die Producer. Die Einsparungen sind daher umso größer, je geringer die Verstärkung ist, da diese frühzeitig auf die Summe der Ereignisse einwirken kann. Bei einer Verstärkung ab ca. 0,8 und besonders > 1 sind keine Verbesserungen mehr möglich, da die Worker bereits soweit wie möglich in Richtung der Consumer migriert wurden und die Verstärkung nicht mehr auf die Summe der Gesamtereignisse einwirkt. Replikationen bei einer Verstärkung > 1 sind laut des Platzierungsmodells nicht vorgesehen.

5.4.3 Adaptivität der Anwendungsplatzierung

Ziel. Ziel des zweiten Experiments ist der Nachweis, dass selbstorganisiertes Verhalten erfolgreich in den vorgestellten Ansatz integriert wurde. Das bedeutet, dass ein stabiles System nach einer induzierten Änderung wieder in einen Zustand gebracht wird, der stabil und möglichst optimal ist. Dabei werden verschiedene Experimentparameter verändert. Zunächst wird das Verhalten des Optimierungsansatzes bei der Variation der Anzahl der Producer unter Beibehaltung der anderen Experimentparameter untersucht. Mit diesem Experiment wird verdeutlicht, welche unterschiedlichen Optimierungspotentiale die Migration und die Migration mit Replikation aufweisen. Das zweite Experiment veranschaulicht, welches Verhalten der vorgestellte Optimierungsansatz bei Änderungen der Verstärkungen der Worker zeigt. Die anderen Experimentparameter bleiben dagegen unverändert. Dieses Experiment verdeutlicht, dass der vorgestellte Optimierungsansatz adaptiv ist und auf laufende Änderungen reagiert sowie auch bei veränderten Situationen optimale Anwendungsplatzierungen anstrebt. Das dritte und letzte Experiment dieser Experimentgruppe demonstriert die unterschiedlichen Verzögerungen, die bei der Platzierungsanpassung existieren. Unter ausschließlicher Variation des Schwellwertes der Migration werden sowohl die Anzahl der Migrationen, als auch deren zeitlicher Ablauf dargestellt.

Annahmen. Auch bei diesem Experiment wird von Komponenten (Broker und Ausführungscontainern) ausgegangen, welche über begrenzte, aber dennoch stets ausreichende Ressourcen verfügen. Die zu verteilenden Anwendungen liegen bereits in Form von einzelnen Komponenten (Worker) vor und können eigenständig, ohne vorherige Dekomposition, im Netzwerk verschoben werden.

Ablauf. Das Experiment läuft folgendermaßen ab: Zunächst wird ein azyklisches Netzwerk aufgebaut und auf jedem Knoten werden ein Middlewarebroker und ein Ausführungscontainer platziert. Auf jedem Container steht die gleiche ausreichende Anwendungsausführungskapazität bereit. Nach dem Aufbau des Brokernetzwerks und der Middleware erfolgt die Verteilung der Worker, Consumer und Producer, wobei die Worker in Chains organisiert sind. Die Verteilung der Consumer und Producer erfolgt zufällig anhand einer Gleichverteilung.

Beim ersten Experiment wird die Anzahl der Producer variiert, sie liegt zwischen 1 und 20 bei einer gleichbleibenden Verstärkung von etwa 0,3. Zu jedem ganzzahligen Wert für die Produceranzahl wird die Anzahl der Nachrichten im Netzwerk bestimmt. Die Messung erfolgt bei eingeschwungenem Systemzustand.

Beim zweiten Experiment wird das System mit einer 3-1-1 Chain optimiert. Zu bestimmten, festgelegten Zeitpunkten werden Änderungen der Verstärkung induziert und die Anzahl der Nachrichten im System vor, während und nach der Optimierung bestimmt.

Das dritte Experiment betrachtet das Verhalten des Systems bei Änderung des Schwellwerts der Migration bei einheitlicher Chain und Verstärkung. Es zeigt

den Zusammenhang zwischen der Anzahl und der zeitlichen Verteilung der Migrationen zum Schwellwert. Im laufenden Betrieb wird der Schwellwert geändert und gemessen, wann zuletzt eine Migration stattfand und wieviele Migrationen bei welchen Schwellwerten auftreten.

Variation. Es finden drei unterschiedliche Ausprägungen von Variationen statt. Zunächst wird die Anzahl der Producer bei einer gleichbleibenden Verstärkung und ähnlichen Chains (x-1-1) variiert. Bei der zweiten Variation werden bei einer 3-1-1 Chain im laufenden Betrieb die Verstärkungen der Worker verändert. Im dritten Experiment wird der Schwellwert der Migration bei ansonsten gleichen Experimentparametern verändert. Die Variationen sind bei jedem Teilexperiment in den folgenden Teilabschnitten genauer beschrieben.

Erwartung. Bei Veränderung der Anzahl der Producer ist zu erwarten, dass das System nach der induzierten Veränderung eine erneute Optimierung mit anschließender stabiler Phase durchführt. Dieser Zyklus wird mehrfach ausgeführt. Es ist zu erwarten, dass nach einer Anpassung an die induzierte Änderung zunächst sowohl die Anzahl der Nachrichten insgesamt, als auch die eigentlichen Nutzdaten/Ereignisse ansteigen. Durch die folgenden Optimierungsschritte mittels der Basisoperationen werden die Anzahl von Nachrichten und Ereignissen verringert, wobei das Niveau der Nachrichtenanzahl vor der induzierten Änderung vermutlich nicht mehr erreicht wird. Stattdessen ist zu erwarten, dass sich die Anzahl der Nachrichten auf einem höheren Niveau als zuvor stabilisiert. Zudem wird vermutlich bei der kombinierten Migration mit Replikation eine geringere Nachrichtenanzahl als bei ausschließlicher Migration erreicht.

Bei der Variation der Verstärkung der Worker ist ebenfalls zu erwarten, dass das System nach der induzierten Veränderung eine erneute Optimierung mit anschließender stabiler Phase durchläuft. Auf welchem Niveau sich die Anzahl der Nachrichten nach der Änderung einpendelt, hängt von der veränderten Verstärkung ab. Liegt sie unter der zuvor geltenden, wird vermutlich ein insgesamt geringeres Niveau erreicht, bei größerer Verstärkung liegt das Niveau vermutlich ebenfalls höher als zuvor. Es wird angenommen, dass die Nachrichtensummen direkt nach einer Änderung sprunghaft ansteigen und anschließend im Laufe der Optimierungen absinken.

Der Zusammenhang zwischen der Anzahl der Migrationen und der Höhe des Schwellwerts ihrer Auslösung besteht voraussichtlich darin, dass bei niedrigem Schwellwert viele Migrationen hintereinander stattfinden. Hingegen sollten bei zunehmendem Schwellwert die Migrationen weniger oft ausgeführt werden und mehr Zeit zwischen den jeweiligen Migrationen vergehen, da es länger braucht, bis die jeweiligen Schwellwerte erreicht werden. Entsprechend wird sich auch die Anzahl der Migrationen entwickeln. Es wird zudem von einem Zusammenhang zwischen dem Schwellwert und dem fixen Vergessensfaktor α ausgegangen.

Ergebnisse und Diskussion. Die Ergebnisse der Experimente sind nochmals untergliedert, es wird zunächst das Experiment unter der variierenden Anzahl von Produzenten betrachtet, anschließend erfolgt ein Experiment zur Messung der Anzahl von Nachrichten über den zeitlichen Ablauf der induzierten Anpassung der Verstärkung der Worker und zuletzt die Variation der Schwellwerte bei der Migration.

Variation der Anzahl der Produzenten. Bei diesem Experiment variiert die Anzahl der Produzenten, während die Verstärkung gleich bleibt und bei 0,3 liegt. An der Chain ändern sich, neben der Anzahl von Producern, die Teilnehmer nicht. Es bleibt bei einer x-1-1 Chain. Es werden wieder die Ergebnisse der Migration sowie der Migration und Replikation im Vergleich zu der nicht-optimierten Variante gezeigt.

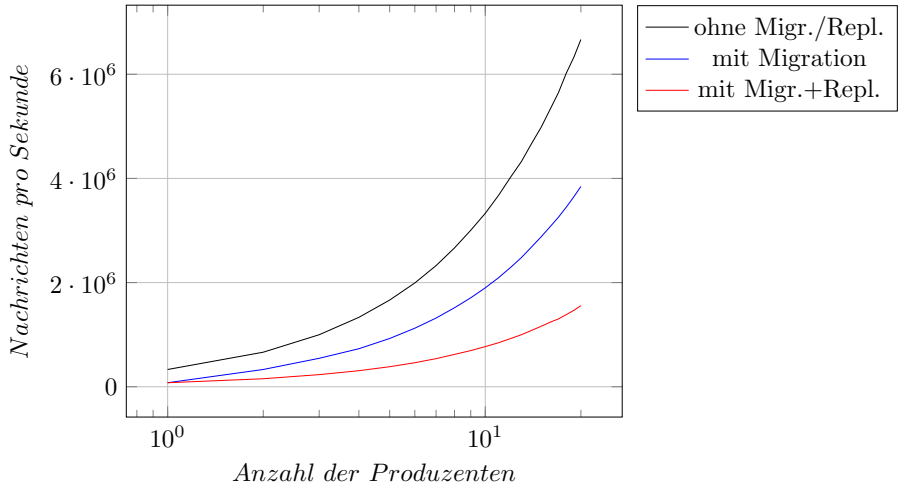


Abbildung 5.8: Anzahl der Nachrichten bei Änderung der Produzentenanzahl

Die Messungen der optimierten Varianten erfolgen, wie auch im vorigen Experiment, im eingeschwungenen Zustand, wobei die Anzahl der Producer zwischen einem und etwa zwölf gemessen wird, siehe Abbildung 5.8. Bei nur einem Producer liegen Migration und Migration mit Replikation auf gleichem Niveau und unterhalb der nicht-optimierten Variante. Zu diesem Zeitpunkt sind die Worker in die direkte Nähe der Producer migriert, eine Replikation mit Migration wird zu diesem Zeitpunkt nicht ausgeführt. Mit zunehmender Anzahl der Produzenten steigt der Abstand zwischen der Kurve mit der Migration und jener mit Migration und Replikation stärker, als zwischen der Kurve mit dem nicht-optimierten Verlauf und dem Verlauf unter Einsatz der Migration. Durch die Replikation und die Migration werden die Worker bei einer Verstärkung < 1 (hier 0,3) so nah wie möglich bzw. möglichst direkt bei den Producern ausgeführt. Damit werden frühzeitig nicht benötigte Nachrichten herausgefiltert. Ist keine Replikation vor-

gesehen (Variante mit ausschließlich Migration), wandern die Worker nur solange in Richtung der Producer, bis diese am Schnittpunkt der Nachrichtenquellen angelangt sind und auf diesem Broker verbleiben. Mit diesem Experiment wird nochmals deutlich, was sich bereits im vorherigen Abschnitt (5.4.2) abgezeichnet hat. Zum einen ist dies die Tatsache, dass bei einer Verstärkung von < 1 die Migration mit der Replikation mehr Nachrichtenvolumen einspart als der Einsatz der Migration allein. Zum anderen vergrößert sich bei zunehmender Anzahl von Producern die Vorteilhaftigkeit der Migration mit der Rekombination gegenüber dem alleinigen Einsatz der Migration. Allerdings wird bereits durch den Einsatz der Migration die Gesamtanzahl von Nachrichten deutlich reduziert, obwohl der Anstieg der Kurve der Migration zwischen der Kurve des nicht-optimierten Laufs und der kombinierten Migration und Replikation liegt. Bleibt als Zusammenfassung dieses Teilexperiments die Erkenntnis, dass der Optimierungsansatz mehr Nachrichten einspart, je mehr Producer an der Chain beteiligt sind.

Anpassung nach Änderung. Das folgende Teilexperiment dient der Veranschaulichung, was bei einer Anpassung der Verstärkung der Worker im laufenden Betrieb zu erwarten ist. Abbildung 5.9 verdeutlicht das Verhalten, wobei die Abszissenachse die abgelaufene Zeit darstellt. Das Setup ist eine 3-1-1 Chain, und damit die Chain, bei der bei einer Verstärkung < 1 die höchsten Einsparungen durch Migration und Replikation zu erwarten sind. Anders als bei den vorherigen Experimenten werden zudem alle Nachrichten des Netzwerks auf der Ordinatenachse zusammengefasst. Die Darstellung zeigt ein System von Beginn an, also vom Anfangszustand über die Einschwingphase bis zur Induzierung mehrerer Änderungen. Die Einschwingphase beginnt beim Startzeitpunkt und ist ab etwa 180 *sec* beendet. Die kleinen Ausschläge bis hierhin und im weiteren Verlauf entstehen durch die internen Nachrichten, welche durch den Austausch von Informationen, z.B. über Migrationen und Replikationen, zusätzlich zur Nutzlast übertragen werden. Zu den Zeitpunkten 300, 600 und 900 *sec* wird die Verstärkung verändert. Sie liegt zunächst bei 0,2, steigt zum Zeitpunkt 300 auf den Wert von 20, fällt zum Zeitpunkt 600 auf 0,5, um zum Zeitpunkt 900 abschließend auf 10 zu steigen. Wie aus den vorherigen Experimenten abzulesen ist, werden bei der Migration die Worker bei einer Verstärkung < 1 solange in Richtung der Producer migriert, bis sie die Nahtstelle zu den Producern erreichen. Durch zusätzliche Replikation werden die Worker bis zum Ausführungsort der Producer migriert. Bei einer Verstärkung von > 1 migrieren die Worker dagegen so weit wie möglich in Richtung der Consumer.

Nachdem sich das System eingeschwungen hat, sind alle Worker in Richtung der Producer migriert und das Minimum an Nachrichten in diesem Zustand ist erreicht. Die Verstärkung der Worker wird anschließend verhundertfacht, die Anzahl der Nachrichten „explodiert“ dann. Durch schrittweise Migration der Broker in Richtung der Consumer wird die Anzahl der Nachrichten kontinuierlich abgebaut, sodass nach etwa 100 *sec* erneut ein Minimum erreicht ist. Dass dieses nun leicht unterhalb des ursprünglichen Minimums liegt, ist der Netzwerkinfrastruktur und der Auswahl von Schwellwerten zur Replikation und Migration

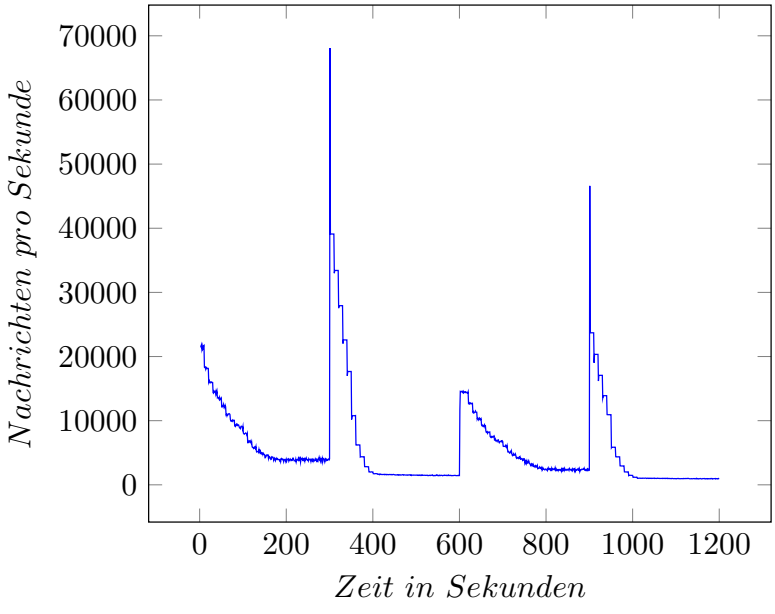


Abbildung 5.9: Änderung der Verstärkung der Worker im laufenden Betrieb: Zum Zeitpunkt 300 von 0,2 auf 20, zum Zeitpunkt 600 von 20 auf 0,5 und zum Zeitpunkt 900 von 0,5 auf 10

geschuldet. Durch die Absenkung der Verstärkung auf nur noch 0,5 zum Zeitpunkt 600 wird eine weitere Migration in Richtung der Producer ausgelöst, so dass sich die Worker wieder auf den Produzenten befinden. Der zunächst vorhandene Sprung der Anzahl der Nachrichten zum Zeitpunkt 600 ist weniger stark als der zum Zeitpunkt 300, allerdings ist die Veränderung bei 600 auch nur um den Faktor 40 und nicht wie zum Zeitpunkt 300 um 100 erfolgt. Die Migration der Worker in Richtung der Producer benötigt zudem länger als im ähnlichen Abschnitt zuvor, da die Komponenten bereits zerlegt vorliegen. Die Erhöhung der Verstärkung zum Zeitpunkt 900 um den Faktor 20 lässt die Anzahl der Gesamtnachrichten wieder sprunghaft ansteigen, allerdings liegt der Anstieg absolut gesehen zwischen denen zu den Zeitpunkten 300 und 600, was auch durch den Änderungsfaktor zu erklären ist. Anschließend migrieren sich die Worker wieder in Richtung der Consumer, der Zeitaufwand entspricht dabei dem der Migration nach der Änderung zum Zeitpunkt 300.

Durch dieses Teilexperiment wird nachgewiesen, dass das System nicht nur auf Änderungen der Chain reagiert, sondern auch mit sich änderndem Verhalten der Worker mithalten kann, wobei annähernd gleiche Ergebnisse erreicht werden und auch die Änderungsgeschwindigkeit je Migrationsrichtung in etwa konstant ist. Somit ist festzustellen, dass der in dieser Arbeit vorgestellte Platzierungsansatz dem System ermöglicht, sich auf ändernde Situationen einzustellen und in kurzer Zeit einen ressourcenschonenden Zustand zu erreichen.

Variation des Schwellwerts bei der Auswertung. Bei diesem Experiment wird die Höhe des Schwellwerts variiert, bei dem eine Migration ausgelöst wird. Das Experiment findet unter dem Einsatz einer einheitlichen Verstärkung der Worker von 0,3 statt, wobei die historischen Werte mit einem Vergessensfaktor von 0,2 in die Berechnung eingehen. Die Struktur der Chain ist 3-1-1 mit drei Producern und jeweils einem Worker und einem Consumer. Im Zusammenspiel der Chain-Struktur mit der Verstärkung befindet sich der Platzierungsansatz, allein bei ausschließlicher Betrachtung der Migration, im maximal möglichen Verbesserungsbereich (siehe Abbildung 5.7).

Abbildung 5.10 zeigt den Zusammenhang zwischen der Höhe des Schwellwerts zur Auswertung der Statistik (Schwellwert) und der Zeitspanne, wann zuletzt eine Migration stattgefunden hat. Dabei wird ab dem Zeitpunkt 0 gemessen und damit auf eine Einschwingphase verzichtet. Bei einem Schwellwert bis etwa 1 schwankt der Zeitraum zwischen den Migrationen auf hohem Niveau und hoher Intensität. In diese Phase fällt das Einschwingen, daher die große Varianz. Anschließend fällt der Zeitraum sehr zügig auf ein niedriges Niveau ab und bleibt dort bis zu einem Schwellwert von etwa 90. Dieser Zeitraum ist geprägt durch die eigentliche Verbesserung der Platzierung, es finden viele Migrationen statt, bis die maximal möglichen Verbesserungen erreicht sind. Rund um den Schwellwert 100 steigt die Zeit zwischen den Migrationen wieder signifikant an. Grund dafür ist, dass erforderliche Migrationen bereits durchgeführt wurden und ein lokales Optimum erreicht werden konnte. Zwischen dem Schwellwert 100 und 200 verläuft die Kurve ähnlich zum Abschnitt zwischen 0 und 100, allerdings auf leicht höherem Niveau als zuvor. Grundsätzlich werden bei niedrigerem Schwellwert mehr Migrationen durchgeführt als bei höherem. Durch die Kombination mit dem Vergessen bei jeder Platzierung kommt es zu dem Effekt, dass rund um den Schwellwert von 100, wo weniger Migrationen ausgeführt werden, auch die Statistikdaten länger aktuell bleiben. Daher wiederholt sich der Trend jeweils nach der Erhöhung des Schwellwerts um 100.

Abbildung 5.11 zeigt die gleichen Ergebnisse wie Abbildung 5.10, wobei die Werte auf der Ordinatenachse nun logarithmisch dargestellt sind und die Darstellung zusätzlich um die Anzahl von Migrationen angereichert ist. Neben dem schon in Abbildung 5.10 beschriebenem Verlauf ist zu erkennen, dass bei niedrigem Schwellwert insgesamt mehr Migrationen durchgeführt werden als bei höherem. Dies ist durchgehend zu erkennen und gilt auch rund um den Schwellwert 100. Durch den Einfluss der Statistik und dem Vergessen steigt die Zeit zwischen den Migrationen rund um den Schwellwert von 100 an.

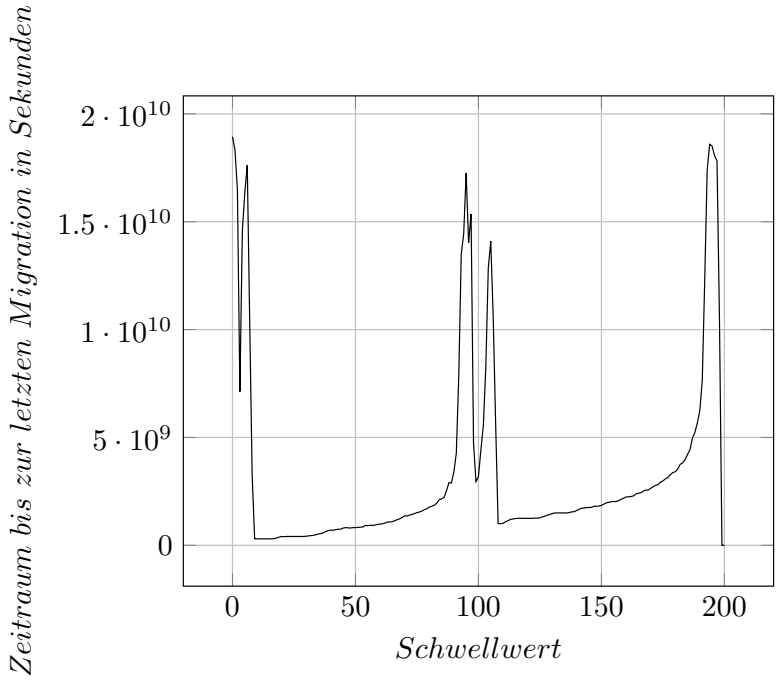


Abbildung 5.10: Änderung des Schwellwertes bei der Migration

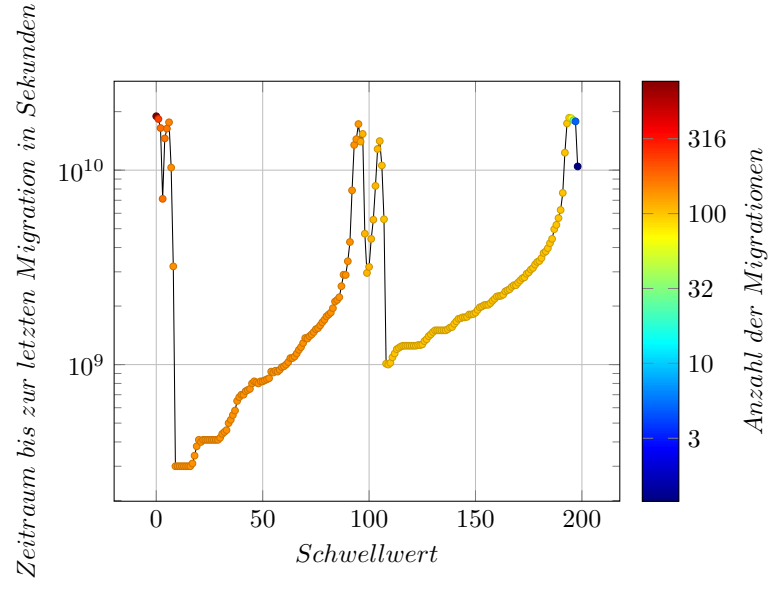


Abbildung 5.11: Änderung des Schwellwertes bei der Migration im Zusammenhang mit der Anzahl von Migrationen im laufenden Betrieb

5.4.4 Optimierung unter beschränkten Ressourcen

Ziel. Bei den bisherigen Experimenten wurde das Verhalten des Systems bzw. der Komponenten unter der Maßgabe betrachtet, dass Broker und Ausführungskomponenten stets über ausreichende Kapazitäten verfügen.

Die folgenden Experimente betrachten den vorgestellten Optimierungsansatz vor allem in Bezug auf die Wirksamkeit und das Verhalten des Ansatzes unter eingeschränkten Ressourcen. Die Ressourcen (verfügbare Ausführungskapazität für Anwendungen) sind, im Gegensatz zu den vorherigen Experimenten, unterschiedlich verteilt. Die zugeordneten Ressourcen bewegen sich in einem festgelegten Rahmen, der im Vorfeld des Experiments bekannt ist. Es wurden Teilexperimente mit unterschiedlicher Ressourcenverteilung durchgeführt. Dazu gehören Experimente mit begrenzten, aber einheitlichen Ressourcen und Experimente mit begrenzten, in einem Wertebereich schwankenden Ressourcen. Die Ressourcenschwankungen unterliegen zusätzlich einer statistischen Verteilung. Bei beschränkten Ressourcen kann es generell zu Verklemmungen, sprich *Deadlocks* kommen. Dies kann immer dann eintreten, wenn die Anwendungskomponente A_i von Broker B_1 in Richtung Broker B_2 wandern möchte, dieser aber keine freien Kapazitäten hat. Selbst wenn diese Migration sinnvoll ist, kommt es zu einer Verklemmung, die erst aufgelöst werden kann, wenn die bei B_1 belegte Kapazität durch Einstellung oder Weggang einer auf B_1 ausgeführten Komponente frei wird. Das Entstehen eines Deadlocks bei beschränkten Ressourcen und dem einfachen Migrationsverfahren ohne gegenseitigem Komponententausch ist zunächst nicht zu verhindern, allerdings ermöglicht der in dieser Arbeit vorgestellte Ansatz genau den gegenseitigen Austausch von Komponenten. So kann Komponente A_i von B_1 nach B_2 migrieren, wenn dafür eine Komponente mit mindestens gleich großem Kapazitätsbedarf von B_2 nach B_1 wechselt. Ein gegenseitiger Tausch von Komponenten ist kein Garant dafür, jederzeit zuverlässig einen Deadlock verhindern zu können. Allerdings würde die Migration dann abbrechen, wenn keine ausreichenden Ressourcen am Zielbroker vorhanden sind und zu einem späteren Zeitpunkt, bei gleichen Ereignisströmen, einen neuen Migrationsversuch starten. Das hier umgesetzte Verfahren zur Platzierungsanpassung gehört zu den optimistischen Verfahren, welches einen Deadlock hinnimmt und hofft, ihn beim nächsten Versuch umgehen zu können. Neben der Migration ohne Komponententausch wird im Folgenden auch das Verhalten mit Komponententausch in den Experimenten berücksichtigt.

Annahmen. Broker und Ausführungscontainer besitzen im Vorfeld des Experiments festgelegte Ausführungsressourcen. Diese sind entweder fix oder schwanken in einem bestimmten Wertebereich unter Normalverteilung. Die zu verteilenden Anwendungen liegen bereits in Komponenten gegliedert vor und können ohne vorherige Dekomposition verschoben werden. Eine Rekombination ist nur nach vorheriger Replikation möglich, d.h. es müssen gleiche Filter vorhanden sein, welche sich wieder vereinen lassen. Die Betrachtung bzw. das Experiment

erfolgt als Chain in der Form 1-1-1, es bilden demnach ein Producer, ein Worker und ein Consumer eine Verarbeitungskette.

Ablauf. Zunächst wird, wie in den vorherigen Experimenten auch, ein azyklisches Netzwerk aufgebaut, wobei auf jedem Knoten ein REBECA-Broker und ein Anwendungscontainer platziert werden. Die Kapazitäten der Broker sind entweder fix oder variieren zwischen bestimmten Werten unter Gleichverteilung. Nach dem Aufbau des Brokernetzes werden den Anwendungscontainern einheitlich fixe oder unterschiedliche Ressourcen zugewiesen. Danach erfolgt die zufällige, gleichverteilte Platzierung der 1-1-1-Chains (Producer, Consumer und Worker). Nach dem Deployment erfolgt die Optimierung der initialen Platzierung, wobei nur die Migration betrachtet wird. Dies ist ausreichend, da auch die Replikation auf eine folgende Migration zurückgreift und bei einer Chain dieser Form eine Replikation ohnehin nicht durchgeführt wird. Anschließend wird die Verstärkung zwischen einem Wert von etwa 0,3 bis 150 variiert. Die Messungen erfolgen getrennt für jede unterschiedliche Konfiguration der Größe der Ausführungscontainer. Bei jedem der Teilexperimente wird jeweils bei unterschiedlicher Verstärkung der Worker die Gesamtanzahl von Nachrichten im Netzwerk gemessen und in einer gemeinsamen Darstellung zusammengefasst.

Variation. Die Variation besteht darin, dass die Optimierung mit und ohne gegenseitigem Austausch von Komponenten durchgeführt wird. Die einzelnen Experimente werden unter verschiedenen Bandbreiten von Ressourcen wiederholt. Dabei gelten die Ergebnisse aus Experiment 5.4.2 mit der Abbildung 5.2 als Referenz für die obere und untere Schranke (untere Schranke: optimierte Variante, obere Schranke: keine Optimierung). Weitere Variationen sind die Beschränkung der Ausführungskapazität und der Einsatz von gegenseitigem Komponentenaustausch. Während bei gegenseitigem Austausch digital, d.h. mit und ohne variiert wird, variiert die Kapazität in der Form, dass bei den Referenzschranken mit unbegrenzter Kapazität gemessen und ansonsten zwischen jeweils einer festen und einheitlichen (2 und 5) einerseits und zwischen schwankenden, normalverteilten Ausführungsressourcen (2 bis 10 sowie 2 bis 7) andererseits unterschieden wird.

Erwartung. Durch die Beschränkung der Ressourcen ist zu erwarten, dass mittels Migration im Vergleich zum Ausgangszustand signifikante Einsparungen möglich sind. Diese fallen jedoch vermutlich deutlich geringer aus als bei uneingeschränkt verfügbarer Kapazität. Damit werden alle Optimierungen unter den restriktiveren Randbedingungen zwischen der oberen und unteren Schranke liegen. Bei der Optimierung ohne gegenseitigen Tausch ist außerdem zu erwarten, dass sich die Netzwerkkonsumption weniger stark verringert als bei gegenseitigem Austausch, es ohne Austausch zu Deadlocks kommt und weitere Verbesserungen nicht möglich sind. Darüber hinaus wird ohne gegenseitigen Anwendungsaustausch die Anzahl der Nachrichten voraussichtlich geringer ausfallen als mit Tausch. Bei insgesamt geringerer Kapazität der Ausführungsressourcen werden

zudem weniger Migrationen aufgrund der größeren Anzahl von Verklemmungen durchgeführt. Dies bedeutet, dass bei geringeren Ressourcen höhere Nachrichtenmengen zu erwarten sind. Zusätzlich ist zu erwarten, dass es auch bei geringerer Kapazität und dem gegenseitigen Komponentenaustausch zu Deadlocks kommt. Außerdem ist damit zu rechnen, dass die initiale Verteilung der Ressourcen größeren Einfluss auf das Optimierungsergebnis hat, da die initiale Verteilung, besonders bei ressourcenarmen Setups weniger gut aufgelöst werden kann und Komponenten eher an unvorteilhaften Positionen bzw. in deren Umfeld bleiben. Die Auswertungen für die optimierte und nicht-optimierte Variante unter unbegrenzten Ressourcen bilden, aller Voraussicht nach, nicht nur die obere und untere Schranke, sondern dienen auch als Vorbild für die zu erwartenden Ergebniskurven bei beschränkten Kapazitäten.

Ergebnisse und Diskussion. Im Folgenden werden die Ergebnisse der Experimente dargestellt. Es werden zunächst die Ergebnisse der Messungen mit einheitlicher Kapazität und daran anschließend die Messergebnisse mit normalverteilten Ausführungsressourcen abgebildet. Außerdem wird die Migration einmal wie bisher ohne den gegenseitigen Austausch („Handel“) von Komponenten den Ergebnissen gegenübergestellt, die durch zusätzlichen, gegenseitigen Austausch zustande gekommen sind.

Gleichmäßige Kapazität mit und ohne Handel. Das folgende Ergebnis zeigt die Variation der Verstärkung einer 1-1-1 Chain, also mit einem Producer, Worker und Consumer, wobei insgesamt 50 Chains auf 100 Broker verteilt wurden. Als Referenz ist die obere und untere Schranke in der Abbildung 5.12 durch die nicht-optimierte und die durch Migration optimierte Platzierung zu erkennen. Beide Varianten arbeiten dabei mit unbegrenzten Kapazitäten, wie in Abbildung 5.2 gezeigt wird. Weitere Experimente wurden mit den globalen Kapazitäten 2 und 5 durchgeführt, wobei jeweils eine Messreihe mit und ohne gegenseitigen Austausch von Komponenten erstellt wurde.

Gleichmäßige Kapazität der Größe 2
Bei der Beschreibung der Ergebnisse unter Berücksichtigung einer gleichmäßigen Kapazität von 2 in Abbildung 5.12 fällt zunächst auf, dass die Kurve unter Einbeziehung von Platzierungsverbesserungen ohne Austausch von Anwendungskomponenten parallel und leicht unterhalb der Kurve ohne Ressourcenbeschränkung und Platzierungsverbesserung liegt. Der charakteristische Verlauf der nicht-optimierten Kurve resultiert daher, dass die laufende, lineare Erhöhung der Verstärkung am Broker im weiteren Verlauf in Richtung der Consumer an jedem Weiterleitungsschritt eine weitere Erhöhung nach sich zieht. Durch die geringe Kapazität können jedoch nur wenige Migrationen durchgeführt werden, es kommt schnell zu einer Verklemmung. Die Anwendungskomponenten können bei einer Verstärkung < 1 nicht weit in Richtung der Producer und bei einer Verstärkung > 1 nicht weit in Richtung der Consumer migrieren. Bei gegenseitigem Austausch können dagegen mehr Migrationen ausgeführt werden, sodass die Komponenten,

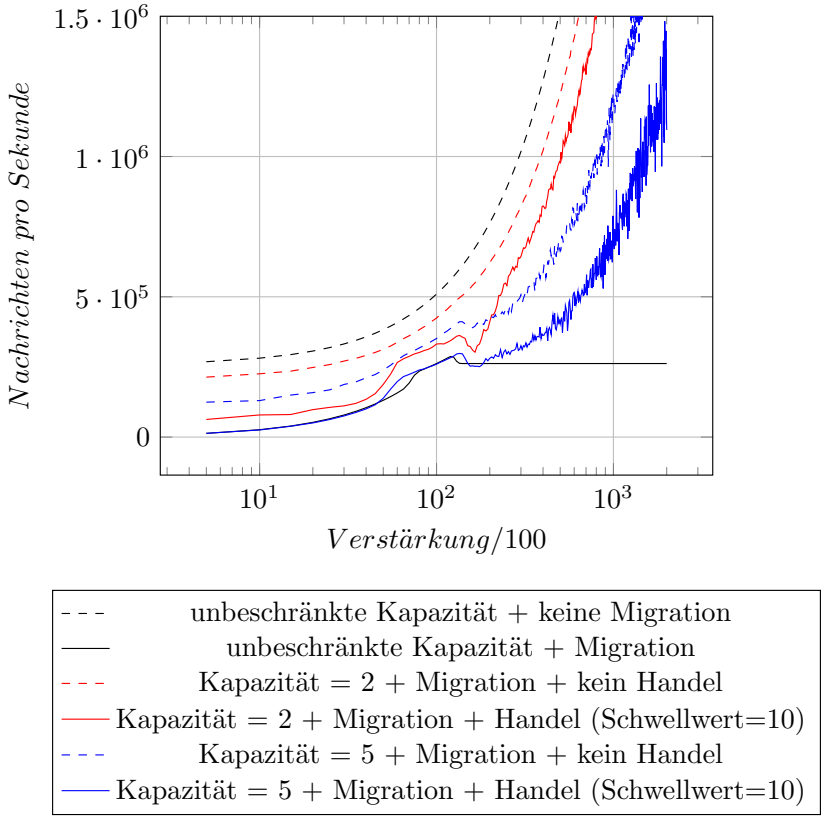


Abbildung 5.12: Anzahl der Nachrichten einer 1-1-1 Chain mit beschränkter, gleicher Kapazität mit und ohne Komponententausch

je nach Verstärkung, in Richtung der Producer oder Consumer wandern. Die Kurve orientiert sich an der optimierten, nicht ressourcenbeschränkten Variante. Sie verläuft allerdings deutlich oberhalb dieser und unterhalb der Messung mit der Kapazität von 2 ohne Komponententausch. Die Kurve verläuft bis zu einer Verstärkung von $0,8$ nicht exakt parallel zur optimierten Variante, verbessert sich jedoch bis $0,8$ stärker als die Ausgangskurve, während sie bei einer Verstärkung zwischen $0,8$ bis $1,2$ deutlich darüber, aber parallel zur optimierten Kurve verläuft. Das liegt wie bei der optimierten Variante daran, dass die Verzögerung der Entscheidung, ob eine Migration bei entsprechenden Schwellwerten sinnvoll ist, länger abgewartet wird. Bei einer Verstärkung von 1,2 sinkt die Kurve unter Ressourcenbeschränkungen stärker als bei der optimierten Variante. Die Gründe dafür sind zunächst, dass Entscheidungen eindeutiger getroffen werden und weiterhin, dass durch den Tausch von Komponenten ein vergleichsweise niedriges Niveau an Nachrichten erreicht wird. Das Niveau der optimierten Variante ohne Ressourcenbeschränkungen wird jedoch nicht erreicht. Bei höherer

Verstärkung sind keine weiteren Verbesserungen möglich, sodass der Effekt der steigenden Verstärkungen und längeren Wege stärker ins Gewicht fällt, als die Platzierung der Komponenten in der Nähe der Consumer. An der relativen Lage der Kurve ändert sich jedoch nichts. An dieser Stelle ist auffällig, dass die Kurve, welche die Anzahl der Nachrichten mit gegenseitigem Austausch präsentiert, weniger glatt verläuft als bei den vorher betrachteten Experimenten. Dies hat wiederum statistische Gründe, da die Auswirkungen der zufälligen, initialen Platzierung umso bedeutender sind, je weniger freie Migrationen ausgeführt werden können. In diesem Fall trifft das zu, da durch die begrenzten Ressourcen und ungünstige Platzierung von Anwendungskomponenten ein gegenseitiger Tausch ebendieser nicht möglich ist und es trotz der Tauschmöglichkeit doch zu Verklemmungen kommt.

Gleichmäßige Kapazität der Größe 5

Die Kurven des Experiments mit der gleichen Kapazität von 5 Ressourceneinheiten verlaufen ebenfalls zwischen der oberen und unteren Schranke des Experiments ohne Ressourcenbeschränkungen und jeweils unter den Kurven der Experimente mit 2 Ressourceneinheiten. Bei beiden Kurven fallen sofort die Ausschläge auf, sodass auch hier die statistischen Effekte der initialen Patzierung, trotz gleich gebliebener Anzahl von Wiederholungen, einen signifikanten Einfluss auf das Ergebnis haben. Ohne Austausch von Komponenten sind die Schwankungen weniger stark als mit dem gegenseitigen Tausch, dies entspricht dem Verhalten bei einer Kapazität von 2. Die Kurve ohne Komponentenaustausch verläuft bis zu einer Verstärkung von ca. 1,2 in etwa parallel zur optimierten, nicht ressourcenbeschränkten Variante. Bis zur Verstärkung von 1,2 liegt die Kurve zwischen den Kurven mit einer Kapazität von 2 (mit und ohne Komponententausch), während sie ab einer Verstärkung von 1,2 unterhalb aber parallel zur Kurve mit der Kapazitätsbeschränkung von 2 ohne Komponententausch liegt. Bei einer Verstärkung bis zu 1,2 können die Komponenten relativ frei in Richtung der Producer migrieren, es sind zumindest in der Nähe der Producer ausreichend freie Kapazitäten vorhanden. Bei zunehmender Verstärkung migrieren die Worker zwar auch in Richtung der Consumer, allerdings kommt es vorher zu Verklemmungen, welche nicht gelöst werden können. Die Steigerung der Verstärkung führt im Zusammenspiel mit der Verlängerung des Weges zur stärkeren Erhöhung der Gesamtanzahl der Nachrichten. Dieses Verhalten wurde bereits in vorangegangenen Experimenten beobachtet. Der leichte „Knick“ bei einer Verstärkung von 1,2 ist weniger stark ausgeprägt als bei der optimalen Variante und der Messung mit der Kapazität von 2 mit gegenseitigem Austausch. Die Gründe für das Verhalten bei einer Verstärkung von 1,2 sind auch hier statistische Effekte in Zusammenhang mit dem Schwellwert. Die Variante mit einer Ausführungskapazität von 5 schwankt im letzten Drittel (Verstärkung $> 1,2$) noch stärker als die anderen Varianten, allerdings kommt sie bei Verstärkungen von $< 1,2$ größtenteils auf die gleichen Werte wie die optimierte Variante ohne Ressourcenbeschränkungen. In diesem Bereich wird die Güte der unlimitierten Kapazität des Experiments erreicht. Es zeigt sich, dass trotz Ressourcenbeschränkungen durch den Austausch von Anwendungskomponenten gute Ergeb-

nisse erzielt werden können. Rund um die Verstärkung von 1 sind die Nachrichtensummen lediglich leicht erhöht, da einerseits statistische Effekte während der Auswertung und andererseits der Aufwand für den Austausch berücksichtigt werden. Bei einer Verstärkung von 1,2 erreicht in diesem Experiment die Summe der Nachrichten nochmals das Niveau der optimierten, nicht kapazitätsbeschränkten Variante bzw. unterbietet es leicht. Bei zunehmender Verstärkung werden die Worker in Richtung der Consumer migriert, durch die Konzentration der Worker in Richtung der Consumer migrieren diese nicht unmittelbar bis zu den Consumern. Die höheren Verstärkungen bewirken im Zusammenhang mit den zurückzulegenden Verbindungen zwischen Workern und Consumern eine stärkere Erhöhung der Gesamtsumme von Nachrichten als durch die alleinige Erhöhung der Verstärkung. Dieses Verhalten wurde bereits mehrfach beobachtet. Auffällig an dieser Kurve sind die angesprochenen starken Schwankungen im letzten Drittel ab einer Verstärkung von 1,2 und der Bereich nahe 1,2. Hier ist zu beobachten, dass die Anzahl der Gesamtnachrichten leicht unter dem Niveau des optimierten, nicht kapazitätsbeschränkten Experiments liegt. Auch dieses Ergebnis kommt durch statistische Effekte zustande, da bei dieser Verstärkung im Vergleich zu der Situation mit der Verstärkung von rund 1,2 weniger zuvor eindeutige Platzierungsentscheidungen getroffen wurden und durch statistische Effekte während der initialen Platzierung sowie den Tausch von Komponenten sogar kurzzeitig bessere Lösungen als bei der Optimierung ohne Beschränkungen erreicht werden.

Normalverteilte Kapazität mit und ohne Handel. Das folgende Ergebnis zeigt die Variation der Verstärkung einer 1-1-1 Chain mit je einem Producer, Worker und Consumer. Es wurden insgesamt 50 Chains auf 100 Broker verteilt. Als Referenz ist die obere und untere Schranke in der Abbildung 5.13 durch die nicht-optimierte und die durch Migration optimierte Platzierung dargestellt. Beide Varianten arbeiten mit unbegrenzten Kapazitäten, wie in Abbildung 5.2 gezeigt wurde. Weitere Messungen wurden durch Experimente mit normalverteilten Kapazitäten a) zwischen 2 und 7, sowie b) zwischen 2 und 10 vorgenommen. Zusätzlich wurden beide Experimente mit und ohne Austausch von Komponenten durchgeführt und die Anzahl der Nachrichten im System ermittelt.

Normalverteilte Kapazität der Größe 2 bis 7
Bei der Beschreibung der Ergebnisse unter Berücksichtigung einer normalverteilten Kapazität im Bereich von 2 bis 7 in Abbildung 5.13 fällt zunächst auf, dass die Kurve unter Einbeziehung von Migration ohne Austausch von Anwendungskomponenten parallel und deutlich unterhalb der Kurve ohne Ressourcenbeschränkung und Platzierungsverbesserung verläuft. Die nicht-optimierte Kurve verläuft wie abgebildet, da die lineare Erhöhung der Verstärkung des Workers in Richtung der Consumer mit jedem Schritt dorthin eine noch größere Steigerung der Gesamtanzahl der Nachrichten nach sich zieht. Durch die geringe Kapazität können jedoch nur wenige Migrationen durchgeführt werden, es kommt schnell zu einer Verklemmung, in deren Folge Anwendungskomponenten bei einer Verstärkung < 1 nicht weit in Richtung der Producer und bei einer Verstärkung > 1 nicht weit in Richtung der Consumer migrieren können. Allerdings liegt die Kurve im Ver-

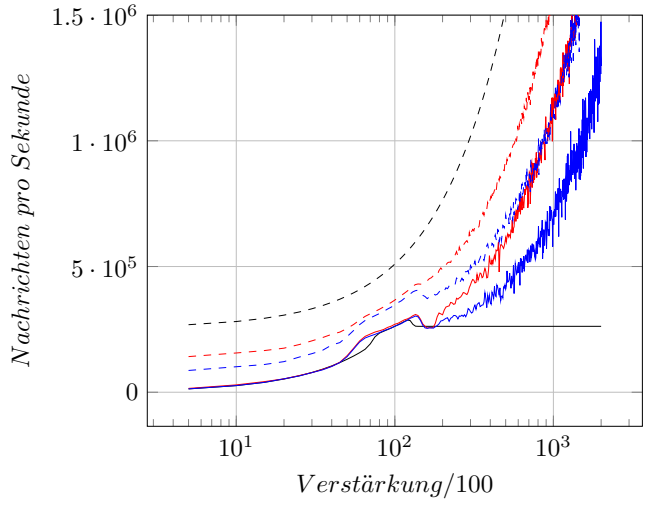


Abbildung 5.13: Anzahl der Nachrichten einer 1-1-1 Chain mit beschränkter, normalverteilter Kapazität mit und ohne Komponententausch

gleich zur Kurve mit der gleichmäßigen Kapazität von 2 ohne Tausch in Abbildung 5.12 deutlich darunter. Dies liegt einerseits an der im Durchschnitt höheren Kapazität bei der Normalverteilung, andererseits existieren mehr Möglichkeiten für Schwankungen, sodass durch eine zufällige, aber möglicherweise geschickte Verteilung der Ressourcen insgesamt eine vorteilhaftere Platzierung möglich ist. Auffällig ist insbesondere, dass diese Kurve im Vergleich zur Kurve mit der gleichmäßigen Kapazität von 2 ohne Tausch bereits deutlich unregelmäßiger erscheint, was wiederum den unterschiedlichen Möglichkeiten der initialen Platzierung geschuldet ist. Wie bereits bei der gleichmäßigen Verteilung mit der Kapazität 2, wird die Summe von Nachrichten durch zusätzlichen Tausch von Komponenten nachhaltig gesenkt. So erreicht bereits die Kurve mit zusätzlichem Komponententausch (Normalverteilung der Kapazitäten von 2 bis 7) bereits bis zur Verstärkung von ca. 1,2 das Niveau der optimierten Version mit unbegrenzten Ressourcen. Allerdings entspricht der Kurvenverlauf zwischen der Verstärkung von 0,8 bis 1,2 nicht exakt dem Verlauf der optimierten Kurve mit unbegrenzten Ressourcen, sondern ist in den Randbereichen leicht erhöht. Dieses Verhalten ist auch hier statistischen Effekten geschuldet. Während bei einer Verstärkung knapp über 1,2 die Kurve bis auf das Niveau der optimierten, nicht ressourcen-

beschränkten Kurve fällt, steigt sie jedoch anschließend wieder, bleibt aber unter dem Niveau der Kurve ohne Komponententausch. Im Bereich ab der Verstärkung von $> 1,2$ sind die Komponenten wiederum nicht in der Lage, sich direkt auf den Brokern mit den Consumern zu positionieren. Durch den Effekt der längeren Wege und der größeren Verstärkung erhöht sich wiederum entsprechend schnell die Gesamtanzahl der Nachrichten im System.

Beim Vergleich beider Kurven ist zu erkennen, dass der gegenseitige Austausch von Komponenten vorteilhaft ist und damit, trotz Ressourcenbeschränkungen, zumindest bei ungleichmäßig verteilten Ressourcen bei einer Verstärkung < 1 ebenso gute Ergebnisse erzielt werden wie mit unbeschränkten Ressourcen. Verklemmungen bzw. das Ausreizen der Kapazitätsgrenzen finden gemäß diesem Experiment im Bereich mit einer Verstärkung $> 1,2$ statt, sodass Komponenten nicht exakt zu den Consumern migriert werden können und damit die Gesamtanzahl von Nachrichten deutlich erhöhen, die Nachrichtensummen jedoch deutlich unter der nicht-optimierten und nicht ressourcenbeschränkten Variante liegen.

Normalverteilte Kapazität der Größe 2 bis 10
Die Kurven des Experiments mit der normalverteilten Kapazität zwischen 2 und 10 verlaufen unterhalb bzw. gleich der Experimente mit normalverteilten Kapazitäten zwischen 2 und 7 und damit zwischen der oberen und unteren Schranke. Die durchschnittliche Kapazität der Broker ist zunächt einmal höher als bei der Variante mit der Normalverteilung von 2 bis 7 und bei der gleichmäßigen Verteilung mit der Kapazität 5. Da letztgenannte Verteilung mit dem Komponententausch bis zur Verstärkung von 1,2 dem optimalen, nicht ressourcenbeschränkten Verlauf entspricht, war zu erwarten, dass zumindest mit dem Komponententausch bei der normalverteilten Kapazität von 2 bis 10 das gleiche Ergebnis erreicht wird. Dies ist auch eingetreten. Ohne Komponenententausch verläuft die Kurve unterhalb der Messung bei der normalverteilten Kapazität von 2 bis 7, jedoch nur leicht unterhalb der Kurve bis zur Verstärkung von 1,2. Danach sackt die Kurve ohne Komponententausch stärker ab und nähert sich der Kurve der normalverteilten Kapazität von 2 bis 10 mit Komponententausch an, allerdings mit geringerer Schwankungsbreite. In diesem Bereich gleicht der Komponentenhandel die geringere Kapzität aus.

Die Experimentergebnisse mit normalverteilten Ressourcen zwischen 2 bis 10 mit Komponententausch liegen im Bereich bis zur Verstärkung von 1,2 exakt auf dem Niveau der Variante mit gegenseitigem Komponententausch und der normalverteilten Kapazität von 2 bis 7 und damit auf dem Niveau der optimierten Variante ohne Kapazitätsbeschränkungen. Die höhere Kapazität von 2 bis 10 hat an dieser Stelle keinen Einfluss auf die Summe der im System vorhandenen Nachrichten, sondern allein der Austausch der Komponenten reicht, um nahezu gleich gute Ergebnisse zu generieren, als wären die Kapazitäten unbeschränkt verfügbar. Dies bestätigen auch die Ergebnisse, welche in Abbildung 5.12 gezeigt werden. Bei einer Verstärkung von mehr als 1,2 ermöglicht der Komponententausch zwar noch bessere Ergebnisse, jedoch ist durch statistische Effekte nochmals eine größere Schwankung zu beobachten und das Niveau der optimierten, ressour-

cenunbeschränkten Variante wird auch hier nicht erreicht. Stattdessen können die Komponenten nicht in die direkte Umgebung der Consumer migriert werden. Trotz größerer Kapazität kommt es noch immer zu Verklemmungen.

Bewertung. Wie in den Experimenten mit zusätzlichem Austausch von Komponenten zu erkennen ist, spielt zunächst die Höhe der im Netzwerk verfügbaren Kapazitäten eine entscheidende Rolle für den Erfolg der Verringerung der Anzahl der Gesamtnachrichten im Netzwerk. Je mehr Kapazitäten verfügbar sind, desto mehr Deadlocks können vermieden werden. Dies ist spätestens nach Auswertung der vorhergehenden Experimente keine Überraschung, sondern zu erwarten gewesen. Bei nur wenigen verfügbaren Ressourcen kommt es rasch zu Verklemmungen, es können nur geringfügige Verbesserungen erreicht werden. Da in den Experimenten lediglich Ausführungsorte simuliert werden und Anwendungen bereits in Form von Komponenten vorliegen, ist lediglich mit minimalen Verbesserungen zu rechnen. Sind die geringen Ressourcen zudem noch gleichmäßig verteilt, lassen sich zusätzlich weniger „Schlupflöcher“ finden als bei ungleichmäßiger Verteilung. Bei ungleichmäßiger Verteilung ist es den Komponenten eher möglich, sich an Brokern mit höherer Kapazität zu sammeln, um von dort gezielt zu anderen Brokern migriert zu werden. Mithin, und durch die leicht höheren Ressourcen, liefern die Experimente unter normalverteilten Ressourcen bessere Ergebnisse als bei einheitlich verteilten Ressourcen.

Allerdings ist erkennbar, dass trotz höherer Ressourcen und höherer Flexibilität durch unterschiedliche Ressourcen, eine annährend optimale Verteilung nicht erreicht wird, da es während der Migrationen noch immer zu Verklemmungen kommt. Da der Optimierungsansatz nur mittels lokalem Wissen arbeitet, um weitreichende, ressourcenaufwendige und zeitraubende Abstimmungen zu vermeiden, ist keine übergeordnete Koordination der Platzierungsanpassungsschritte möglich. Stattdessen wird durch lokales Wissen und nachbarschaftliche Kooperation bis zu einer Verstärkung von 1,2 ein deutlich besseres bzw. mit dem Verhalten bei unbegrenzten Ressourcen gleichwertig gutes Ergebnis erzielt. In Anbetracht der Tatsache, dass in einer realen Umgebung neben den Kommunikationsressourcen auch die Ausführungsressourcen i.d.R. begrenzt sind, bietet der gegenseitige Austausch bzw. der Handel von Komponenten die Möglichkeit, selbst in diesen Fällen eine deutliche Verringerung der Anzahl von Nachrichten im Gesamtsystem und geringere Kosten trotz des zusätzlichen, aber einmaligen Aufwands durch den Komponentenhandel zu erhalten. Damit der Komponentenhandel nur jeweils in eine Richtung vollzogen wird und damit einem dauerhaften und keinem kurzlebigen Trend folgt, spielt das Zusammenspiel mit dem Migrationsschwellwert, wie auch bei der Bestimmung eines „Reibungswerts“ als Platzierungskosten und vierte Kraft des Kräftemodells, eine wichtige Rolle.

5.4.5 Auswirkungen der initialen Platzierung auf das Optimierungsergebnis

Ziel. Bei den bisher durchgeführten Experimenten wurden die Producer, Consumer und Worker initial stets mit gleicher statistischer Methode (Gleichverteilung) verteilt und anschließend unter verschiedenen Randbedingungen optimiert. Ziel dieses Experiments ist die Betrachtung des Verhaltens des Ansatzes bei unterschiedlicher Verteilung der Worker. Dazu werden die Worker nicht mehr wie bisher gleichverteilt, sondern paretoverteilt. Das bedeutet, dass in ihrem Arbeitsablauf zusammenhängende Worker in der Nachbarschaft und damit „klumpenförmig“ zueinander liegen. Das Ziel besteht im Nachweis, dass die laufende Verbesserung bei unterschiedlicher initialer Verteilung möglich ist und gleich gute Ergenisse wie bei der (initialen) Gleichverteilung liefert.

Annahmen. Broker und Ausführungscontainer besitzen im Vorfeld des Experimentlaufs festgelegte, ausreichende Ausführungsressourcen. Die Ressourcen der Broker und Ausführungscontainer sind gleich und schwanken nicht. Die zu verteilenden Anwendungen liegen bereits in Komponenten gegliedert als Worker vor und können ohne vorherige Dekomposition verschoben werden. Eine Rekombination ist nur nach vorheriger Replikation möglich, d.h. es müssen gleiche Filter (Semantik) vorhanden sein, welche sich wieder vereinen lassen.

Ablauf. Zunächst erfolgt der Aufbau eines azyklischen Brokernetzwerks, auf jedem Knoten wird ein REBECA-Broker inklusive eines Anwendungscontainers platziert. Producer und Consumer werden gleichverteilt und finden sich am Rand des Brokernetzwerks. Die Kapazitäten der Broker und der Ausführungscontainer sind gleich und bleiben es auch während der Experimentlaufzeit. Nach dem Aufbau des Brokernetzes werden die Worker zufällig und paretoverteilt platziert. Nach erfolgtem Deployment erfolgt die Optimierung der initialen Platzierung. Zur Bewertung des Ansatzes finden Messungen für den nicht-optimierten Fall bei ausschließlicher Durchführung von Migration und der Kombination von Migration und Rekombination statt. Die Experimente werden für unterschiedliche Verstärkungen durchgeführt.

Variation. Die Variation bei diesem Experiment besteht darin, die Worker nicht mehr gleichverteilt im Netzwerk zu platzieren, sondern „gehäuft“ auf einigen wenigen, benachbarten Brokern. Dazu wird die Verteilung der Worker verändert. Bei der genutzten Paretoverteilung sind die Worker initial zentriert, gleichzeitig können durch die statistische Verteilung ausreichend zufällige, initiale Verteilungen erzeugt werden.

Das Experiment gliedert sich in zwei Teilexperimente. Während zunächst wie in Experiment 5.4.3 die Adaptivität der Platzierungsanpassung untersucht wird,

wird im zweiten Teilexperiment beleuchtet, wie effizient die Anzahl der Nachrichtenströme durch die Anpassung der Workerplatzierung bei zunehmender Verstärkung verringert wird. Damit entspricht dieses Teilexperiment dem Experiment 5.4.2. Bei diesem Experiment wurde allerdings deutlich, dass der Optimierungsansatz mit Replikation nur dann seine Vorteile ausspielen kann, wenn die Worker in Richtung der Producer migriert und verschoben werden können. Dies ist regelmäßig der Fall, wenn mehr Producer als Worker und Consumer in der Chain vorhanden sind. Besonders bei einer Verstärkung von < 1 ist die Replikation mit Migration vorteilhaft. Ab einer Verstärkung von > 1 wird nur noch die Migration als Optimierungsmittel genutzt, die Vorteile der Replikation können nicht mehr ausgespielt werden. Bei dieser Konstellation migrieren die Worker soweit wie möglich in Richtung der Consumer, sodass die Summe der Nachrichten bei Migration immer geringer ausfällt als in der nicht-optimierten Variante, jedoch einer vergleichbar großen Steigung unterliegt. Dies ist für alle bisher und vor allem in Abschnitt 5.4.2 durchgeführten Experimente zu beobachten, sodass nur im Bereich der Verstärkung < 1 ein anderes Verhalten zum parallelen Kurvenverlauf beobachtet werden kann. Aus diesem Grund werden Experimente, bei denen die Anzahl der Consumer > 1 beträgt, in diesem Zusammenhang nicht beleuchtet. Stattdessen werden die Basisvarianten, sprich eine 1-1-1 Chain sowie eine 3-1-1 Chain betrachtet und verglichen.

Erwartung. Es wird erwartet, dass durch eine andere initiale Platzierung ähnliche, d.h. gleichsam vorteilhafte Ergebnisse zu erzielen sind wie bei der gleichverteilten initialen Platzierung. Allerdings ist davon auszugehen, dass bei initialer, paretoverteilter Platzierung am Rand des Netzwerks u.U. mit einer längeren Optimierungszeit zu rechnen ist. Es wird außerdem erwartet, dass bei dem Einsatz von Replikation und Migration höhere Einsparungen an Nachrichtenvolumen erreicht werden können als durch ausschließliche Migration.

Ergebnisse und Diskussion. Die vorliegenden Simulationsergebnisse verdeutlichen, dass die initiale Platzierung nur geringen Einfluss auf die erreichbare Platzierung der Anwendungskomponenten hat.

Chain: 1-1-1. Bei diesem Teilexperiment wird mit einer Vielzahl von Chains im Setup 1-1-1 gearbeitet. Jede Chain besteht aus genau einem Producer, einem Consumer und genau einem Worker. Die Worker verstärken dabei die Ereignisströme mit unterschiedlichem Faktor, dieser liegt zwischen einem Wert von fast 0 und reicht bis zu einem Wert über 10. Wie in Abbildung 5.14 zu sehen ist, steigt die Anzahl der Nachrichten ohne den Einsatz von Migration bzw. Migration und Replikation mit wachsender Verstärkung, wobei beide Optimierungsvarianten wieder nahezu deckungsgleich sind. Wie auch beim Teilexperiment 5.4.2 mit einer 1-1-1 Chain liegen die Kurven der optimierten Varianten zwischen 0 und einer Verstärkung von 1 in etwa parallel zur nicht optimierten Kurve, steigen rund

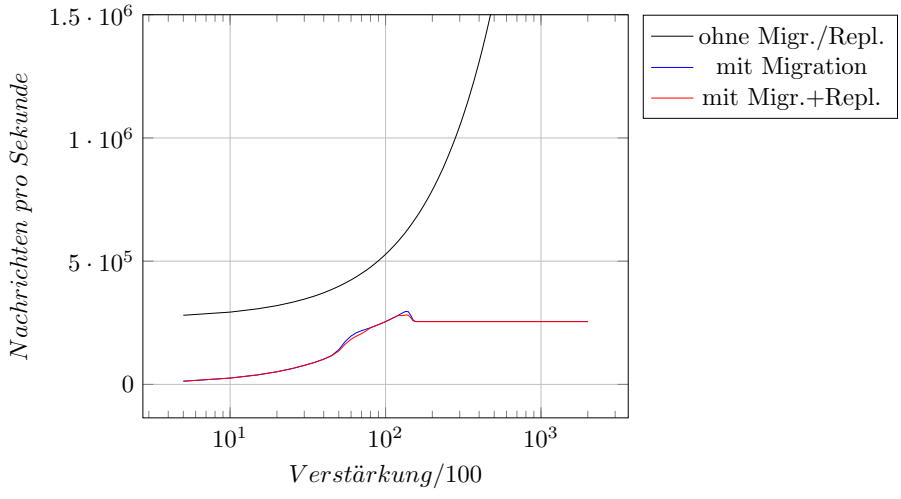


Abbildung 5.14: Optimierung einer 1-1-1 Chain mit paretoverteilten Workern

um die Verstärkung von 1 auf ein leicht höheres Niveau, um danach leicht abzusinken und auf einem gleichbleibenden Niveau zu verharren. Die Optimierungen verlaufen gleich der Variante mit gleichverteilten Workern. Bei einer Verstärkung von < 1 migrieren die Worker in Richtung der Producer, während sie bei einer Verstärkung von > 1 in Richtung der Consumer migrieren. In diesem Fall steigen die optimierten Kurven auch nicht weiter an, da die Worker auf den Consumern liegen und Verstärkungen der Worker keine Auswirkungen auf die Gesamtanzahl der Nachrichten im System haben. Die erhöhte Anzahl von Nachrichten rund um die Verstärkung 1 ist durch die Kombination von Schwellwerten für Migration und Replikation und den versetzten Auswertungen begründet.

Augenscheinlich verlaufen die Optimierungen von Migrationen mit gleich- und paretoverteilter initialer Platzierung sehr ähnlich bzw. genau gleich, die Abbildung 5.15 bestätigt diesen Eindruck. Damit ist erwiesen, dass die initiale Platzierung keinen Einfluss auf das Optimierungsergebnis oder gar den Verlauf der Optimierung selbst hat. Zur Überprüfung des Gesamteindrucks wird zusätzlich das Verhalten der Optimierung unter dem Einsatz von Migration und Replikation überprüft, zur Veranschaulichung der Ergebnisse dient Abbildung 5.16. Auch hier verlaufen beide Kurven gleich, sodass auch hier die initiale Platzierung keine Auswirkungen auf das Optimierungsergebnis hat. Sowohl bei der Gleich- als auch bei der Normalverteilung der Worker im Experiment haben die Broker stets ausreichend Kapazität. Die Worker können im Zuge der Optimierung zu den vorteilhaftesten Brokern migriert und repliziert werden. In den Abbildungen sind ohnehin nur die Ergebnisse nach erfolgter Migration bzw. Migration und Replikation dargestellt. Unterschiede am Anfang der Optimierung, die durch unterschiedliche initiale Verteilungen hervorgerufen werden, sind somit nicht sichtbar.

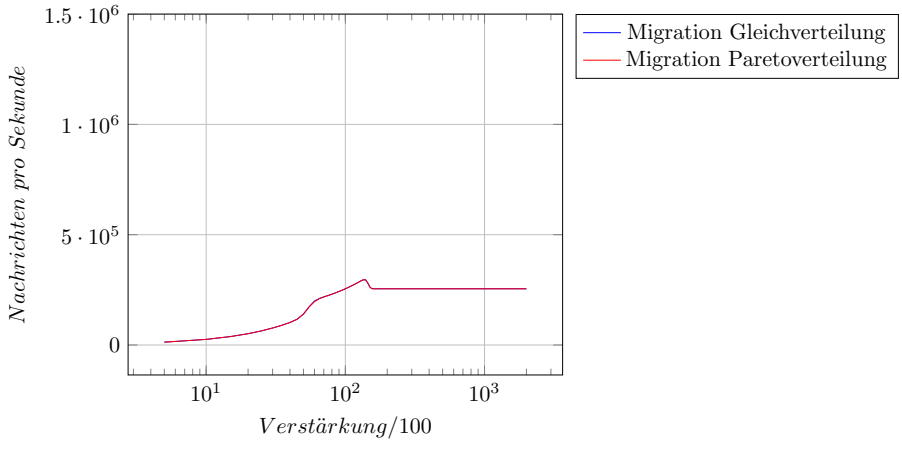


Abbildung 5.15: Vergleich der Migrationsoptimierung einer 1-1-1 Chain mit gleich- und paretoverteilten Workern

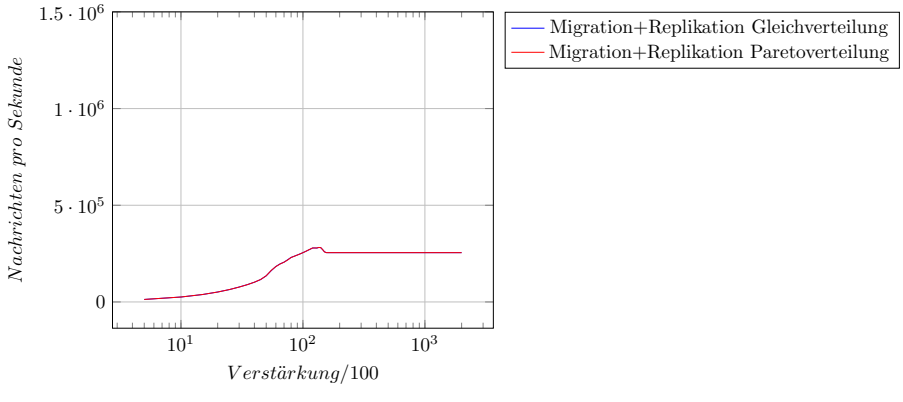


Abbildung 5.16: Vergleich der Optimierung einer 1-1-1 Chain mit Migration und Replikation mit gleich- und paretoverteilten Workern

Chain: 3-1-1. Als weiteres Experiment liegen die Ergebnisse der paretoverteilten Optimierung einer 3-1-1 Chain vor, also einem Setup mit drei Producern und jeweils einem Worker und Consumer. Während des Experiments werden wieder die Verstärkungen der Worker variiert. Diese reichen von einer Stärke zwischen 0 und etwas mehr als 10.

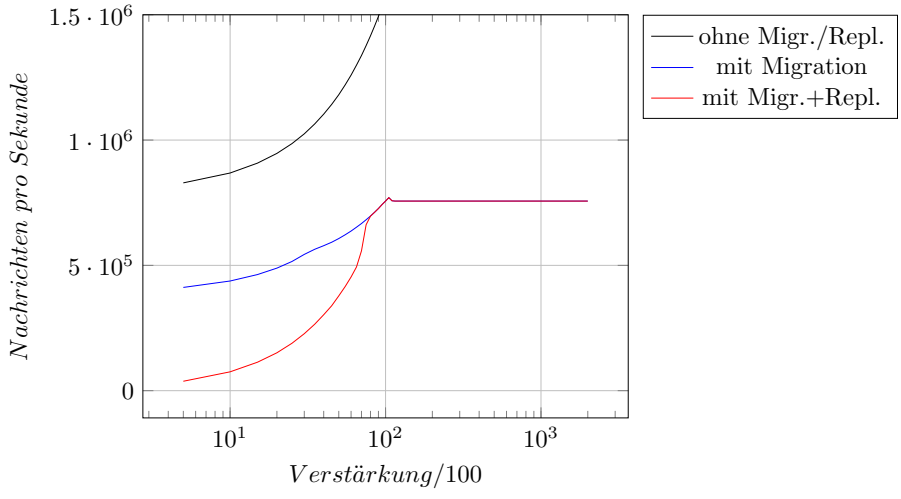


Abbildung 5.17: Optimierung einer 3-1-1 Chain mit paretoverteilten Workern

Wie in Abbildung 5.17 zu sehen ist, steigt die Anzahl der Nachrichten ohne den Einsatz von Migration bzw. Migration und Replikation mit wachsender Verstärkung progressiv. Durch Nutzung von Replikationen zusätzlich zur Migration zeigt die Kurve der Replikation und Migration bei geringer Verstärkung eine deutlich geringere Anzahl von Nachrichten als bei ausschließlicher Migration und bei der nicht-optimierten Variante. Bei einer Verstärkung von leicht unter 1 sind beide Kurven ähnlich und verlaufen anschließend exakt gleich. Während bei einer Verstärkung von < 1 die Worker in Richtung der Producer repliziert und migriert werden, werden Nachrichten so früh wie möglich gering verstärkt bzw. gefiltert. Bei einer Verstärkung von mehr als 1 hingegen migrieren die Worker in Richtung bzw. bis hin zu den Consumern, eine Auswirkung der Verstärkung kommt nicht mehr zum Tragen. Bei einer Verstärkung von 1 bzw. leicht darüber kommt es zu einem weiteren Anstieg. Nach diesem Anstieg fällt die Kurve wieder ab und verläuft anschließend stabil auf gleichem Niveau. Der Grund dafür ist, dass es durch die Bestimmung von Schwellwerten zu leichten Verzögerungen der Platzierungsanpassung, vor allem der Migration, kommt.

Augenscheinlich verlaufen die Optimierungskurven für die Migration und kombinierte Migration und Replikation ähnlich zu dem bereits durchgeführten Experiment in Abschnitt 5.4.2 und der 3-1-1 Chain und gleichverteilten Workern. Dazu werden die optimierte Variante unter ausschließlicher Nutzung von Migration (siehe Abbildung 5.18) und der kombinierten Migration und Replikation

(siehe Abbildung 5.19) miteinander verglichen.

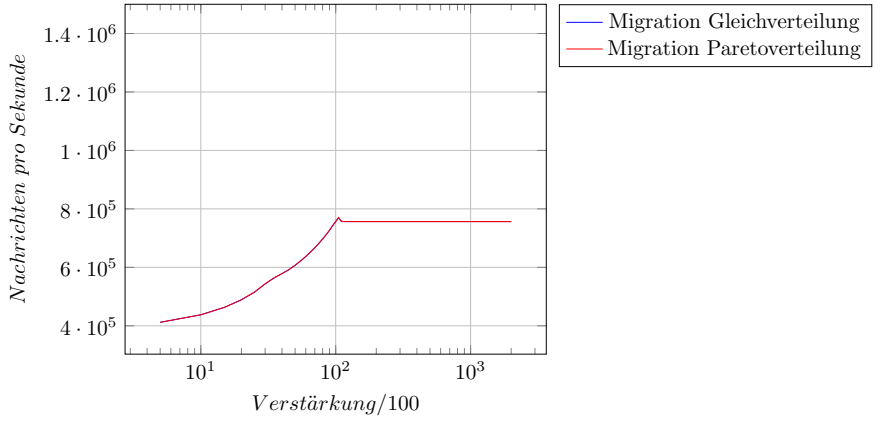


Abbildung 5.18: Vergleich der Migrationsoptimierung einer 3-1-1 Chain mit gleich- und paretoverteilten Workern

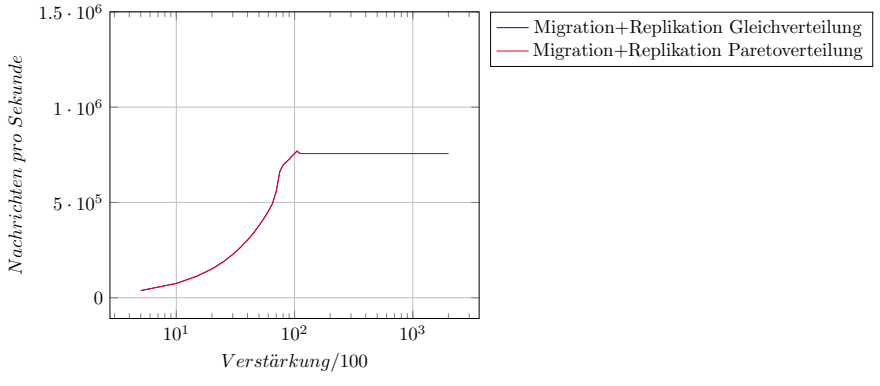


Abbildung 5.19: Vergleich der Optimierung mit Migration und Replikation einer 3-1-1 Chain mit gleich- und paretoverteilten Workern

Da bei beiden Abbildungen die jeweiligen Kurven für Migration und Migration mit Replikation exakt übereinanderliegen, bestätigt dies die Erwartung, dass kein Einfluss von der initialen Platzierung auf das Optimierungsergebnis nachgewiesen werden kann. In den Abbildungen 5.18 und 5.19 werden lediglich die Ergebnisse visualisiert, die nach der Einschwingphase und den durchgeführten Optimierungen erreicht wurden. Sowohl bei gleichverteilten als auch bei paretoverteilten Workern sind in der Versuchsanordnung ausreichend Kapazitäten zur Optimierung vorhanden. Die Worker haben damit ausreichend Möglichkeiten, sich im Netzwerk zu positionieren und gleich gute Ergebnisse zu erreichen.

Optimierung bei Änderung im laufenden Betrieb. Nachdem sich im vorherigen Teilexperiment herausgestellt hat, dass die initiale Platzierung keinen Einfluss auf die Gesamtzahl von Nachrichten hat, untersucht das nun durchgeführte Experiment, ob auch bei laufenden Änderungen mit dem gleichen Optimierungsergebnis zu rechnen ist. Das Experiment orientiert sich an dem in Abschnitt 5.4.3 vorgestellten Experiment zur Änderung der Verstärkung im laufenden Betrieb.

Wie schon in Abbildung 5.9 stellt auch bei den Abbildungen 5.20 und 5.21 die Abszissenachse die abgelaufene Zeit dar und die Ordinatenachse die Anzahl der Nachrichten. Das Setup bildet eine 3-1-1 Chain mit drei Producern und jeweils einem Worker und einem Producer. Diese Anordnung bietet das höchste Einsparpotential im Verhältnis von Migrationen und Migrationen mit Replikationen bei einer Verstärkung von < 1.

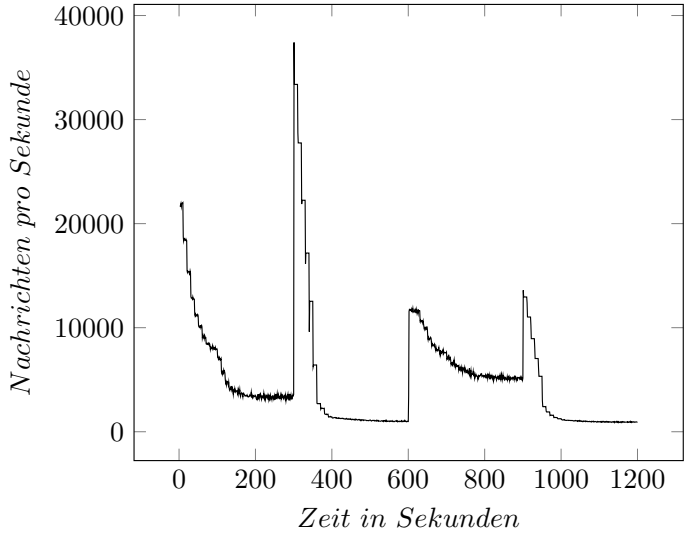


Abbildung 5.20: Änderung der Verstärkung der paretoverteilten Worker im laufenden Betrieb

Zu den Zeitpunkten 300, 600 und 900 *sec* wird die Rate der Verstärkungen durch die Worker verändert. So liegt die Verstärkung zunächst bei 0,2, steigt zum Zeitpunkt 300 auf den Wert von 20, fällt zum Zeitpunkt 600 auf 0,5 und steigt zum Zeitpunkt 900 abschließend auf 10. Zunächst schwingt sich das System ein und die ersten Optimierungen werden durchgeführt. Bis zum Zeitpunkt von 180 *sec* sind beide Systeme eingeschwungen und bei beiden Verteilungen wird ein gemeinsames Minimum erreicht, wobei es bei der Paretoverteilung nicht so gleichmäßig erreicht wird wie bei der Gleichverteilung.

Anschließend wird die Verstärkung zum Zeitpunkt 300 *sec* verhundertfacht, wobei die Anzahl der Nachrichten bei beiden Varianten hochschnellt und einen gleich hohen Spitzenwert erreicht, der danach kontinuierlich abgebaut wird.

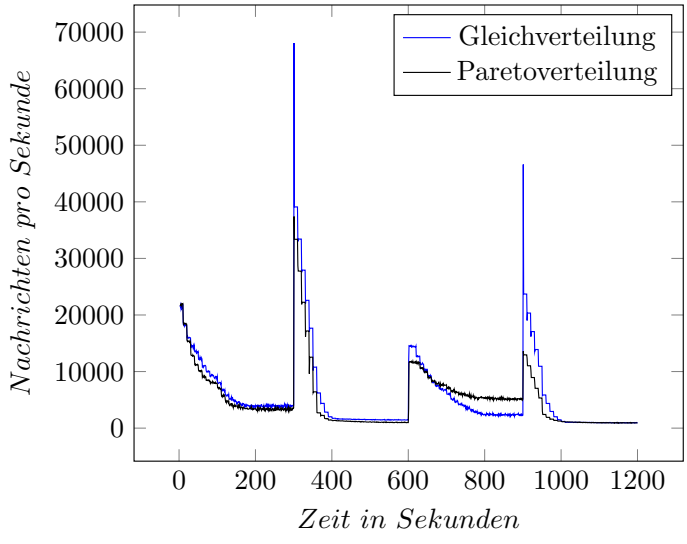


Abbildung 5.21: Änderung der Verstärkung der paretoverteilten Worker im laufenden Betrieb im Vergleich zu gleichverteilten Workern

Auch dieser Vorgang läuft nahezu identisch ab und beide Varianten erreichen ein ähnlich hohes Nachrichtenniveau. Die Absenkung der Verstärkung zum Zeitpunkt 600 *sec* auf nur noch 0,5 löst eine unterschiedlich starke Steigerung der Anzahl von Nachrichten aus, wobei bei der Paretoverteilung ein leicht niedrigerer Spitzenwert gemessen wird. Die anschließende Optimierung erreicht jedoch nicht das Niveau der gleichverteilten Variante. Bei der letzten Änderung zum Zeitpunkt 900 *sec* ist der Ausschlag ebenso weniger stark ausgeprägt als bei der gleichverteilten Variante. Die Optimierung der Paretoverteilung ist jedoch weniger zügig, trotzdem erreichen beide Varianten das gleiche Niveau.

Die unterschiedlich starken Spitzenwerte nach den induzierten Änderungen lassen sich durch die unterschiedlichen Änderungsfaktoren erklären. Die geringeren Ausschläge der Paretoverteilung bei den letzten beiden Änderungen liegen in statistischen Effekten begründet. So migrieren/replizieren die Worker nach der Änderung bei 600 *sec* nicht komplett in Richtung der Producer, so dass die folgende Änderung (Verstärkung) der Nachrichtenanzahl weniger Auswirkungen hat. Insgesamt ist der Optimierungsansatz jedoch bei beiden, unterschiedlichen initialen Platzierungen gleich effektiv.

5.5 Diskussion

Wie an den Ergebnissen der Experimente zu erkennen ist, bietet der Optimierungsansatz bei allen durchgeführten Experimenten Einsparpotential an Nachrichten und Kosten. Die Ergebnisse im Abschnitt 5.4.2 zeigen, dass Optimierungen selbst bei kurzen, für sich genommen übersichtlichen Chains, deutlich die Anzahl von Nachrichten im Gesamtsystem absenken können. Dabei ist die Migration als Basisoperation der Optimierung bei jeder Verstärkung, also bei < 1 und > 1 sinnvoll. Bei einer Verstärkung von < 1 ist zudem die zusätzliche Replikation noch ein wirksames Mittel, um die Anzahl der Nachrichten im Gesamtsystem weiter einzuschränken. Dies wird vor allem bei geringen Verstärkungen deutlich. Bei Verstärkungen von > 1 ist die Replikation nicht hilfreich, da sich stets nur die Ereignismenge der eingehenden Ereignisse einschränken lässt, weil durch die Replikation keine Anwendungskomponenten in Richtung der Ereignissenken verschoben werden können. Der Ereignisraum ist nur in Richtung der Ereignisquellen aufteilbar. Eine Optimierung mit Hilfe der Migration verringert jedoch auch bei Verstärkungen > 1 die Gesamtanzahl von Nachrichten signifikant, auch wenn die Steigerungen ähnlich sind.

Allerdings sei an dieser Stelle angemerkt, dass die erzielten Ergebnisse nur dadurch möglich sind, dass Anwendungen prinzipiell auf jedem Netzwerkknoten ausgeführt werden können. Sind diesbezüglich Einschränkungen vorhanden, verringern sich die möglichen Einsparungen. Die Experimente in Abschnitt 5.4.4 liefern die nötigen Ergebnisse zu dieser Erkenntnis. Anhand der dort gewonnenen Ergebnisse ist zu erkennen, dass eine Verringerung der Kapazität der Ausführungscontainer zu Verklemmungen führt. Eine Verbesserung der Kostensituation durch die Platzierungsänderung von Anwendungskomponenten ist nach wie vor möglich, allerdings fallen die Verbesserungen vergleichsweise gering aus. Größere Einsparungen sind durch eine Erhöhung der Ressourcen möglich, jedoch sind auch größere Verbesserungen erreichbar, wenn einige Knoten im Netzwerk leistungsfähiger als ihre umgebenden Knoten sind und als Sammel- und Verteilungsstelle für Anwendungskomponenten dienen. Dieser Prozess der Verteilung und Steuerung kann auch aktiv und mittels lokalem Wissen vollzogen werden. Verklemmungen lassen sich zu einem großen Teil vermeiden, indem Broker aktiv ihre ausgeführten Anwendungskomponenten verwalten. Durch gegenseitigen Austausch von Komponenten lassen sich mit verhältnismäßig wenig Aufwand weitreichende Verbesserungen erreichen, sodass bei Verstärkungen bis ca. 1,2 und in Abhängigkeit der verfügbaren Ressourcen sogar gleich gute Ergebnisse erzielt werden können, wie dies bei unbegrenzten Kapazitäten möglich ist. Bei Verstärkungen deutlich über 1 können durch Komponententausch ebenfalls deutlich bessere Ergebnisse realisiert werden als ohne diesen Zusatzmechanismus. Allerdings werden die Optimierungsergebnisse mit unbeschränkten Ressourcen in dem hier gewählten Simulationsumfeld nicht erreicht. Darüber hinaus ist an den stärkeren Schwankungen zu erkennen, dass sich die initiale Verteilung von Ressourcen signifikant auf die Ergebnisse der Platzierungsanpassung auswirkt.

Nachdem die Auswirkungen der initialen Platzierung bisher in Randgebieten

beobachtet wurden, bei den Experimenten in Abschnitt 5.4.4 in Erscheinung traten und vor allem für die Größe der Schwankungsbreite von Ergebnissen verantwortlich sind, wurde untersucht, welche Auswirkungen die initiale Platzierung von Anwendungskomponenten auf die Anzahl von Ereignisnachrichten im Gesamtnetzwerk hat. Dazu wurden die im Netzwerk platzierten Worker nicht mehr wie bisher gleich, sondern paretoverteilt. Es kommt zu einer Häufung von Workern rund um bestimmte Knoten im Netzwerk. Es galt herauszufinden, ob der Ansatz in der Lage ist, diese Häufung genauso gut aufzulösen, als wenn die Worker gleichmäßig im Netzwerk verteilt sind. Die Ergebnisse dazu werden in Abschnitt 5.4.5 präsentiert. Es wurden sowohl Verarbeitungsketten mit Migration und Replikation, als auch Verarbeitungsketten ausschließlich mit Migration untersucht und die Ergebnisse mit den paretoverteilten Workern den bisher ermittelten Ergebnissen gegenübergestellt. Die Ergebnisse zeigen, dass die initiale Platzierung der Worker keinen Einfluss auf die Ergebnisse der Optimierung hat und aus dem eingeschwungenen Zustand heraus exakt die gleichen Ergebnisse erzielt werden. Durch das Experiment wurde bestätigt, dass sich der Optimierungsansatz bei paretoverteilten Workern analog der laufenden Platzierungsanpassung verhält wie bei der gleichverteilten initialen Platzierung der Worker. Auch bei laufenden Anpassungen verhält sich der Optimierungsansatz wie bei der initialen Gleichverteilung, lediglich bei einer Anpassung konnte nicht das gleiche (Zwischen-) Ergebnis erreicht werden. Dies ist jedoch durch das Zusammenspiel von Schwellwerten erklärbar. Da bei den Ergebnissen der Platzierung nach der Einschwingphase keine Unterschiede zwischen gleich- und paretoverteilten Workern auszumachen sind, wurde zusätzlich untersucht, wie das System bei Änderung der Verstärkung der paretoverteilten Worker reagiert und mit den Ergebnissen gleichverteilter Worker verglichen. Es zeigt sich, dass die Platzierungsverbesserungen mittels Migration und Replikation bei unterschiedlichen Experimenten gleichsam am effektivsten sind. Die durch Änderungen der Verstärkung ausgelösten Steigerungen der Gesamtzahl von Nachrichten sind allerdings verschieden, was durch die unterschiedlichen Platzierungen der Worker relativ zu den Consumern und Producern und damit in der initialen Verteilung begründet ist. Die Geschwindigkeit der Platzierungsänderung unterscheidet sich sowohl bei Gleich-, als auch bei Paretoverteilung nur geringfügig.

Der in dieser Arbeit vorgestellte Ansatz ist adaptiv und reagiert auf Systemveränderungen. Diese Annahme wurde durch die Experimente in Abschnitt 5.4.3 bestätigt. Zum einen wurde in Abschnitt 5.4.2 deutlich, dass das Optimierungsverfahren in der Lage ist, mit unterschiedlicher Anzahl von Erzeugern Optimierungen durchzuführen. Daneben ist der Optimierungsansatz auch für wechselnde Umgebungsbedingungen geeignet. Die in Abschnitt 5.4.3 gezeigten Ergebnisse belegen, dass der Optimierungsansatz auch mit dynamischen Änderungen umgehen kann. So wurden in Abschnitt 5.4.3 unterschiedliche Produktionsraten der Worker induziert. Dabei steigt das Nachrichtenvolumen im Vergleich zur induzierten Änderung proportional an, wird jedoch anschließend schrittweise optimiert. Dabei erreichten die optimierten Verteilungen grundsätzlich das gleiche, nachrichtensparende Platzierungsniveau.

Kapitel 6

Zusammenfassung und Ausblick

Inhalt

6.1 Überblick über die erlangten Ergebnisse

Intelligente Umgebungen mit einer Vielzahl unterschiedlicher Sensoren und Aktuatoren nehmen einen zunehmend größeren Teil in der persönlichen sowie der IT-Landschaft im Allgemeinen ein. Die in den Umgebungen eingesetzten mobilen Geräte sind dabei grundsätzlich leistungsfähig, allerdings ist die Kommunikation zwischen mobilen Geräten mit einem hohen Maß an Energieaufwand verbunden, wobei der Energievorrat begrenzt ist. Durch die Platzierung von selektiven, nicht benötigte Nachrichten herausfilternde Anwendungskomponenten, lassen sich durch die Nichtweiterleitung von Nachrichten Ressourcen und damit Energie einsparen. Dieses Problemfeld wurde bereits von unterschiedlicher Seite beleuchtet, allerdings fehlen bei den bisher vorgestellten Ansätzen entweder die automatische initiale oder die laufende adaptive Platzierung. Insbesondere die laufende Platzierungsanpassung ist in einem dynamischen Umfeld bedeutsam.

Bei der Art und Weise wie Geräte miteinander kommunizieren, hat sich weitgehend die ereignisorientierte Kommunikation etabliert. Besonders der Einsatz ereignisbasierter Kommunikation mittels Publish/Subscribe ermöglicht die zeitlich, räumlich und in ihrem Arbeitsfluss voneinander entkoppelte Kommunikation und Koordination von Anwendungen untereinander. Damit bildet Publish/Subscribe einen Ausgangspunkt für die Verbesserung der Anwendungsplatzierung, können doch bereits mit dieser Kommunikationsstrategie Geräte und Anwendungen dynamisch dem Netzwerk bei- oder aus diesem austreten.

Die unterschiedlichen Architekturen der an der Kommunikation beteiligten Geräte bedingt für die Kommunikation und flexible Ausführung der einzelnen Anwendungen eine gemeinsame Zwischenschicht. An dieser Stelle kommt die Idee einer Middleware ins Spiel, welche als einheitliche Zwischenplattform sowohl gemeinsam zu nutzende Kommunikationsschnittstellen bereitstellt, als auch als gemeinsame Ausführungsplattform für Anwendungen dient. Bei der Bereitstellung der Ausführungsplattformen müssen jedoch die spezifischen Fähigkeiten der jeweiligen Ausführungsorte beachtet werden, sodass stets die Dienstgüteanforderungen der Anwendungen bzw. der Anwender gewahrt bleiben. Darüber hinaus möchten sich Anwender nicht um die Orchestrierung und Verwaltung der einzelnen Anwendungen kümmern, stattdessen soll die Anwendung wie gewünscht und unterbrechungsfrei arbeiten. Daher muss sowohl die initiale, als auch die laufende Platzierungsanpassung ohne Zutun des Nutzers und damit automatisch erfolgen.

In dieser Arbeit wird ein Konzept entwickelt und präsentiert, wie Anwendungen in einem Umfeld intelligenter Umgebungen kosteneffizient platziert werden können. Dabei ist die Kostenbetrachtung nicht allein auf Weiterleitungskosten ausgerichtet, es werden stattdessen in einem Kostenmodell alle diejenigen Kosten betrachtet, welche die Platzierung von Anwendungskomponenten beeinflussen. Statt Kosten global und mit ebensolchem Wissen zu optimieren, wird auf lokales Wissen zurückgegriffen, welches dann ein lokales Optimum liefert. Die Optimierung selbst erfolgt anhand eines Kräftemodells. Dabei werden Kosten bzw. mögliche Kosteneinsparungen als Kräfte modelliert, welche auf die An-

wendungsplatzierung einwirken. Werden die einwirkenden Kräfte minimiert, so werden auch die auf die Kräfte abgebildeten Kosten minimiert. Mit diesem Ansatz wird schrittweise die Kostensituation verbessert. Zugleich ist es möglich, auf Dynamiken des Kommunikationssystems zu reagieren und die Anwendungsplatzierung laufend anzupassen. Die Platzierungsanpassung ist damit adaptiv und durch ihre Arbeitsweise selbstorganisierend. Bei der Platzierungsanpassung werden die Dienstgüteanforderungen genauso betrachtet wie die jeweils verfügbaren Ressourcen der Ausführungsorte. Dabei werden sowohl die initiale, als auch die laufende Anwendungsplatzierung automatisch ausführt, wobei der Fokus der initialen Platzierung darauf ausgerichtet ist, zunächst eine gültige Platzierung zu finden, sodass dem Nutzer die gewünschte Anwendung in erwarteter Qualität so schnell wie möglich zur Verfügung steht. Darauf aufbauend finden schrittweise Verbesserungen statt. Mit diesem Verfahren wird der zunehmenden Komplexität intelligenter Umgebungen entgegengewirkt und dynamische Veränderungen der Ausführungslandschaft werden mit einbezogen.

In der Arbeit wurden zunächst die Grundlagen der ereignisbasierten Kommunikation mittels Publish/Subscribe vorgestellt. Dazu gehören die Arbeitsweise von Publish/Subscribe und der Aufbau klassischer Middlewarearchitekturen. Anschließend wurde beleuchtet, wie Anwendungsdeployment und Anwendungsmigration ablaufen und wie das kontinuierliche Deployment von Anwendungen ausgeführt wird. Dazu wurde in allen genannten Bereichen auf bestehende Publikationen eingegangen und das in dieser Arbeit entwickelte Konzept von den bestehenden Ansätzen abgegrenzt. Weiterhin wurden im Grundlagenkapitel die Grundzüge selbstorganisierender Systeme beleuchtet.

Im Anschluss wurde das Konzept der initial gültigen und schrittweise verbesserten Platzierung von Anwendungskomponenten entwickelt und eingeordnet. An dieser Stelle wurden zunächst wieder bekannte Ansätze beleuchtet und deren Schwierigkeiten im Zusammenhang mit der Anwendungsumgebung beschrieben. Anschließend erfolgte die Vorstellung des dieser Arbeit zu Grunde liegenden Anwendungsmodells und die Entwicklung des Konzepts zur initialen Platzierung und der laufenden Anwendungsplatzierung. Während die initiale Platzierung weitgehend mit vorhandenen Operationen von Publish/Subscribe umsetzbar ist, war für die laufende Platzierung die Definition eines Kosten- und Platzierungsmodells notwendig, auf dessen Grundlage Platzierungsentscheidungen getroffen werden. Die im Modell definierten Platzierungsziele werden anhand von nur vier Platzierungsoperationen erreicht, welche eine Vielzahl unterschiedlicher Platzierungskonfigurationen ermöglichen.

Das daran anschließende Kapitel zur Implementierung beschreibt zunächst die grundlegenden Techniken bei der Realisierung des Konzepts und damit die Architektur von Rebeca. Dabei spielen vor allem das Konzept von Brokern und die Implementierung von Funktionalitäten, vor allem durch ihre Kombination, eine wichtige Rolle. In dieses Konzept der Trennung von Basis- und weitergehender Funktionalität ist das in dieser Arbeit vorgestellte Konzept eingepasst. Dabei wird bei der initialen Platzierung grundsätzlich auf die vorhande-

ne Publish/Subscribe-Funktionalität zurückgegriffen. Nur wenige Klassen und zusätzliche Funktionalitäten sind dafür nötig. Für die laufende Platzierung sind dagegen weitere Implementierungsschritte und Funktionalitäten notwendig. Im Rahmen der Implementierung werden zusätzliche Schnittstellen- und abstrakte Klassen definiert, von denen konkrete Klassen und Implementierungen abgeleitet werden. Weiterhin wurde erläutert, welche Aufgaben die Statistik- und Strategieklassen haben und wie diese bei der Platzierungsentscheidungsfindung ineinander greifen.

Auf der Grundlage der Implementierung erfolgt die simulative Evaluation des Ansatzes. Das in diesem Kapitel vorgestellte Anwendungsbeispiel der verteilten Detektion komplexer Ereignismuster unterstreicht die Rolle des in dieser Arbeit vorgestellten Ansatzes zur Steigerung der Effizienz in verteilten, intelligenten Umgebungen. Dazu wurde herausgearbeitet, welche bisherigen Methoden der verteilten Detektion die bisher vorgestellten Ansätze haben und wie sie im Vergleich zur hier vorgestellten Lösung arbeiten. Nach diesem Teil und der kurzen Vorstellung der Simulationsumgebung erfolgt die Darstellung der Evaluationsexperimente. Dabei hat sich herausgestellt, dass der Platzierungsansatz deutliche Verbesserungen hinsichtlich des relevanten Ressourcenverbrauchs liefert. Er ist in der Lage, auf sich ändernde Umgebungsbedingungen und Anwendungszustände zu reagieren. Der Erfolg der Anwendungsplatzierung ist zudem unabhängig von der initialen Verteilung der Anwendungskomponenten. Damit ist sichergestellt, dass der Ansatz deutliche Verbesserungen der Kostensituation liefert, sobald er auf eine gültige, initiale Platzierung zurückgreifen kann. Weiterhin ist der Ansatz durch gegenseitigen Komponentenaustausch in der Lage, selbst bei beschränkten Ressourcen und unter der Gefahr möglicher Verklemmungen deutliche Kosteneinsparungen zu erzielen.

6.2 Beurteilung der Beiträge der Arbeit

Ubiquitäre Umgebungen und Anwendungen verbreiten sich zunehmend, angefangen bei der Vernetzung im Rahmen intelligenter Wohnungen und Häuser, bei der Betreuung von hilfsbedürftigen Menschen oder schlicht durch die Vernetzung mobiler Geräte. Durch die Sammlung und Verknüpfung von Informationen aus unterschiedlichen Quellen und Blickwinkeln kann so ein umfassenderes Bild eines Zustands aufgebaut werden.

Das Ziel bei Beginn der Forschungsarbeit war die ressourceneffiziente Verteilung von Detektoren komplexer Ereignismuster. Die Platzierung sollte anhand eines Kräftemodells durchgeführt werden, wobei die möglichen Einsparungen von Kosten als Kräfte modelliert werden und die Platzierung der Detektoren beeinflussen. Dieses aus der Physik entnommene Prinzip der sich ausgleichenden Kräfte wurde bereits in der Vergangenheit bei der Platzierung von Komponenten in heterogenen Umgebungen eingesetzt, jedoch wurden dabei Aspekte dynamischer Umgebungen nicht ausreichend beachtet oder es gab Schwächen bei der initialen

oder der laufenden, adaptiven Platzierung. Ziel der Arbeit war es, einen ganzheitlichen Ansatz für die initiale Platzierung und die laufende Anpassung der Anwendungsplatzierung zu ermöglichen, da in einem dynamischen Umfeld mit wechselnden beteiligten Anwendungen und Geräten sowie laufenden Änderungen in Produktion und Konsum von Informationen zu rechnen ist. Weiterhin ist die Platzierung automatisch auszuführen, da sich die Nutzer i.d.R. nicht um das Platzierungsmanagement kümmern möchten und können.

In dieser Arbeit wurde das Ziel erreicht, eine Methode zu etablieren, welche mit lokalem Wissen Platzierungsentscheidungen trifft, die automatisch durchgeführt werden und sich adaptiv den ändernden Umgebungsbedingungen anpassen. Dabei wurde ein zweistufiger Ansatz entwickelt, bestehend aus dem Finden einer gültigen, initialen Platzierung und der laufenden Anpassung. Während der zweite Teil sich allein auf lokales Wissen stützt und so zusätzlichen und teuren Informationsaustausch vermeidet, verwendet die initiale Platzierung globales Wissen. Diese initiale Platzierung greift allerdings weitgehend auf die Bordmittel der Publish/Subscribe-Kommunikation zurück und ist schlank implementiert.

Mit Hilfe eines Kräftemodells werden die für die Platzierung relevanten Kosten identifiziert und quantifiziert. Dabei spielen alle Kosten eine Rolle, angefangen von der Kommunikation über Ausführung und Speicherung. Diese wurden in einem einheitlichen Kostenmodell zusammengefasst. Dies ist deutlich umfangreicher als das zunächst geplante Modell. Die Kosten wurden in Gruppen gegliedert und ihre Wirkung auf die Aufwendungsplatzierung beschrieben. Es wirken drei Arten von Kräften auf die Anwendungsplatzierung ein. Das Kostenmodell und die Kräfteermittlung sind dabei so gestaltet, dass weitere Kostenarten hinzugefügt werden können, womit der Platzierungsansatz erweiterbar ist.

Weiterhin erfolgte die Implementierung des Ansatzes innerhalb der Publish/Subscribe-basierten Middleware REBECA. Die Implementierung des Konzepts erfolgte nach dem Grundsatz der *Feature Composition*. Damit wurde REBECA ein weiterer Aspekt hinzugefügt und dem Nutzer als Teil eines Gesamtpaketes zur Verfügung gestellt. Es ist damit keine losgelöste Inselimplementierung, sondern eine komponierbare Funktionalität. Dabei ist die Implementierung so gestaltet, dass zunächst Schnittstellen, (teilweise) abstrakte Klassen und dann konkrete Klassen implementiert wurden. Mit Hilfe dieser Vorgehensweise und den einheitlichen Schnittstellen sind spätere Erweiterungen und Spezialisierungen problemlos möglich. Bei der Kostenbewertung und der Entscheidungsfindung sind ebenfalls Räume für spätere Erweiterungen und Spezialisierungen gelassen worden, sodass auch die Implementierung adaptiv an das Umgebungsmodell angepasst werden kann. Eine bereits umgesetzte Erweiterung ist die Integration des gegenseitigen Austausches von Komponenten. Insofern geht der vorgestellte Ansatz über die Idee der Verschiebung von Anwendungskomponenten hinaus und ermöglicht ein aktives Platzierungsmanagement.

Ein weiteres Ziel der Arbeit war die Evaluation des Platzierungsansatzes bezüglich seines Nutzens im Hinblick auf die Einsparung von Ressourcen und seiner Adaptivität. Vorweg findet in Kapitel 5 eine Einordnung des vorgestellten Kon-

zepts in ein besonderes Anwendungsumfeld statt, nämlich die Einordnung des Ansatzes in die verteilte Detektion komplexer Ereignismuster. Hier wurde das Anwendungsumfeld genauer beschrieben und auch eine Abgrenzung zu bereits bestehenden Verfahren zur Verteilung komplexer Ereignismuster in dynamischen Umgebungen vorgenommen. Darüber hinaus wurden die Arbeitsweisen und die Besonderheiten der Detektion komplexer Ereignismuster und deren selbstorganisierte Durchführung und Organisation erläutert. Dieser Abschnitt des Kapitels präzisiert das Spannungsfeld, im dem das vorgestellte automatische, initiale und laufende Deployment von den bisherigen Publikationen zur verteilten Detektion komplexer Ereignismuster abgegrenzt wird. An dieser Stelle wird die Motivation und die Entwicklung des vorgestellten Konzepts weitergehend beschrieben und die Umsetzung in dem Umfeld von Publish/Subscribe gesondert betrachtet. Die Einordnung des erarbeiteten Platzierungsverfahrens und die Abgrenzung von bisherigen Arbeiten ist ein weiterer Beitrag der Arbeit, da unterschiedliche Lösungsansätze inkl. deren Probleme zusammengefasst werden. Für das vorgestellte Konzept bedeutet dieser Abschnitt, dass damit die Entwicklung, Eingrenzung und Umsetzung für das spezielle Anwendungsszenario vorgenommen wird. Die dann folgende Beschreibung der Simulationsumsetzung bildet das Bindeglied zwischen der Experimentdurchführung und der Implementierung.

Die Evaluation der in dieser Arbeit vorgestellten automatischen Anwendungsplatzierung ist ein weiterer Schwerpunkt und wichtiger Beitrag der vorliegenden Arbeit. Als neue, komponierbare Funktionalität der Middleware REBECA hat das implementierte Konzept seine Vorteilhaftigkeit in unterschiedlichen Szenarien bewiesen, wobei der Fokus der Evaluation auf der laufenden Platzierung, also der zweiten Stufe des Konzepts liegt. Die Ergebnisse wurden mit Hilfe einer Simulation erzielt, wobei jedoch nicht das implementierte Verfahren selbst in seinem Verhalten simuliert wurde, sondern auf die Komponenten von REBECA selbst zugegriffen wurde. Lediglich die Umgebung, sprich der Informationsfluss und die Infrastruktur, wurde simuliert. Das Platzierungskonzept hat unter verschiedenen Variationen der Umgebungsvariablen gezeigt, dass es in der Lage ist, die Anzahl von Nachrichten im Netzwerk signifikant zu reduzieren und damit dem Ziel von Ressourceneffizienz einen wichtigen Schritt näher kommt. Dies gilt bei der laufenden Migration bei unterschiedlichen Verarbeitungsketten im laufenden Betrieb und unterschiedlichem Verhalten von Komponenten, bei Veränderung der Platzierung von beteiligten Sendern und Empfängern von Informationen und der Variation des Schwellwertes unter der Maßgabe von beschränkten Ressourcen. Weiterhin wurde bewiesen, dass auch bei knappen Ressourcen durch den gegenseitigen Austausch bzw. den Handel von Komponenten beachtliche Ergebnisse bei gleichzeitig überschaubarem Aufwand erreicht werden. Weiterhin wurde nachgewiesen, dass die initiale Platzierung von Anwendungen keinen nennenswerten Einfluss auf die Platzierungsergebnisse hat und die laufende Platzierung unabhängig von einer initialen Platzierung erfolgversprechend durchgeführt werden kann. Das Platzierungsverfahren ist selbst bei ungleicher Ressourcenverteilung immer dann vorteilhaft, wenn eine gültige initiale Platzierung zustande gekommen ist.

6.3 Ausblick auf offene Forschungsfragen

Die Entwicklung eines Platzierungsansatzes auf der Grundlage physikalischer Prozesse ist gerade in Bezug auf die Abbildung natürlich vorkommender Prozesse im Bereich der Informatik ein erfolgversprechender Ansatz. Mittels des Messens und Kategorisierens von Kosten und ihrer Wirkung in Form von Kräften lassen sich allein mit lokalem Wissen Kosten erheblichen Ausmaßes einsparen.

Erweiterung der Ausführungskosten um ein Modell zur Haftung zwischen Anwendung und Ausführungsort. Das bisher entwickelte und präsentierte Kosten- und Kräftemodell berücksichtigt die bei der Ausführung der Anwendung anfallenden Kosten und die darauf zurückzuführenden Kräfte. Weitere Kräfte können dem Modell hinzugefügt werden, sofern eine Klassifizierung in eine der resultierenden Kraftrichtungen (Bewegung weg vom Ausführungsort, hin zum ehemaligen Ausführungsort, Gegenkraft als Reibung) erfolgen kann. Damit ist das Potential der Kräftemodellierung jedoch nicht ausgeschöpft. Die ebenfalls in dieser Arbeit vorgestellte Berechnung des Matchingwertes lässt sich ebenso in ein Modell pressen, bisher erfolgt die Berechnung nach Erfüllung von vorgegebenen, mitgelieferten Kriterien. Solch ein Modell, welches sich als „Glue-Model“ bezeichnen lässt, dient der Beschreibung des Zusammenhangs zwischen der Anwendung und des Ausführungsortes. Das Klebrigkeitsmodell mit einer formalen Beschreibung ließe sich nicht nur als Beschreibung des Zusammenhaltemaßes bei der initialen Platzierung nutzen, sondern auch für die laufende Platzierung einsetzen. Bisher greifen Broker nur dann in die Anwendungsplatzierung ein, wenn Anwendungen neu platziert oder zumindest neu auf dem jeweiligen Ausführungsort platziert werden. Mit dem Modell kann die Anwendung stattdessen (regelmäßig) überprüfen, welche und mit welcher Priorität Anwendungen bzw. Anwendungskomponenten unbedingt an dieser Stelle ausgeführt werden müssen und bei welchen sich nach einem alternativen Ausführungsort umgesehen werden muss und proaktiv eine andere, mindestens ebenso gute Platzierung erreichen lässt. Die bei der Bewertung der Klebrigkeit gewonnenen Werte ließen sich dann über das Kostenmodell, speziell über die Ausführungskosten, in das Kräftemodell einbinden. Auf diese Weise ließen sich Ressourcen einsparen, sodass weitere, kommende Anwendungen schneller platziert werden können.

Erweiterter Tausch von Komponenten. In der Arbeit wurde bereits ein Verfahren integriert, mit dem Komponenten trotz hoher/voller Auslastung der Ausführungsorte miteinander getauscht werden und so Verbesserungen i.S. der Ressourceneffizienz ermöglichen. Dies ist jedoch nur durchführbar, wenn die Anforderungen der Anwendungen an die Ausführungsorte (z.B. bezüglich der Dienstgüte) erfüllbar sind. Ist dies in der direkten Nachbarschaft nicht möglich, findet auch kein Tausch der Anwendungen statt. Versuche, Methoden der künstlichen Intelligenz, wie bspw. das simulierte Abkühlen, bei denen Anwendungen mit

nachlassender Entfernung zum Ursprungsausführungsort neu positioniert werden, einzubinden, sind wegen der Anwendungsanforderungen nicht ohne weiteres praktikabel. Doch wie ist grundsätzlich die Neupositionierung einer Anwendung umsetzbar? Zunächst besteht bei einem dezentralen und darüber hinaus dynamischen Netzwerk die realistische Möglichkeit, während der Laufzeit eine bessere Platzierung zu finden, als dies zum Zeitpunkt der initialen Platzierung möglich war. Allerdings lassen sich die auf die Platzierung einwirkenden Kosten nicht ohne Weiteres abschätzen, müssten doch bspw. zunächst die entsprechenden Kommunikationsströme etabliert werden. Selbst dann kann mit lokalem Wissen jedoch nur die nachbarschaftliche Kostensituation evaluiert werden, die Gesamtkosten werden nicht betrachtet. Allerdings werden bei miteinander verbundenen Anwendungskomponenten die weiteren Anwendungen zunächst in der Nähe des ursprünglichen Ausführungsortes ausgeführt. Die dazugehörigen Anwendungen würden der zunächst verschobenen Anwendung möglicherweise folgen. Dies bleibt allerdings Spekulation. Dennoch könnte eine Replatzierung einer Anwendung von Zeit zu Zeit sinnvoll sein, dies sollte zumindest als Teil weiterer Forschungsaktivitäten betrachtet werden. So könnte die eine Anwendung ausführende Komponente entweder regelmäßig oder auf Abruf eine Komponente so behandeln, als müsse sie initial und damit neu platziert werden. Der bestmögliche Ausführungsort erhielte hier den Zuschlag und die Komponente würde zu diesem Ausführungsort hin migriert, auch wenn dieser nicht direkt in der Nachbarschaft läge.

Literatur

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing und S. Zdonik. „The Design of the Borealis Stream Processing Engine“. In: *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*. Asilomar, CA, 2005.

[2] Y. Ahmad und U. Cetintemel. „Network-aware query processing for stream-based applications“. In: *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*. Toronto, Canada: VLDB Endowment, 2004, S. 456–467.

[3] J. F. Allen. „Time and time again: The many ways to represent time“. In: *International Journal of Intelligent Systems* 6.4 (1991), S. 341–355.

[4] J. F. Allen und G. Ferguson. „Actions and events in interval temporal logic“. In: *Journal of logic and computation* 4.5 (1994), S. 531–579.

[5] M. Altinel und M. J. Franklin. „Efficient Filtering of XML Documents for Selective Dissemination of Information“. In: *Proceedings of the 26th International Conference on Very Large Data Bases*. VLDB '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, S. 53–64.

[6] A. Arasu, S. Babu und J. Widom. *The CQL Continuous Query Language: Semantic Foundations and Query Execution*. Technical Report 2003-67. Stanford InfoLab, 2003.

[7] C. W. Bachman. „The Programmer As Navigator“. In: *Commun. ACM* 16.11 (1973), S. 653–658.

[8] C. W. Bachman und M. Daya. „The Role Concept in Data Models“. In: *Proceedings of the Third International Conference on Very Large Data Bases - Volume 3*. VLDB '77. Tokyo, Japan: VLDB Endowment, 1977, S. 464–476.

[9] J. Bacon, K. Moody, J. Bates, C. Ma, A McNeil, O Seidel und M Spiteri. „Generic support for distributed applications“. In: *Computer* 33.3 (2000), S. 68–76.

[10] J. Bader, C. Feinen, J. Hedrich, R. D. do Carmo und P. M. Schol. *Proceedings of the Joint Workshop of the German Research Training Groups in Computer Science*. 2013.

[11] S. Bader und T. Kirste. „An Overview of the HELFERLEIN-System“. In: *Proceedings of MMS 2013*. Berlin, Germany, 2013.

[12] B. Balachandran. „Models of the Job Allocation Problem in Computer Networks“. In: *Proceedings of COMPCON 73* (1973).

[13] H. Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. Heidelberg: Spektrum, 2011.

[14] A. Beloglazov, J. Abawajy und R. Buyya. „Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing“. In: *Future generation computer systems* 28.5 (2012), S. 755–768.

[15] A. Beloglazov und R. Buyya. „Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers“. In: *Concurrency and Computation: Practice and Experience* 24.13 (2012), S. 1397–1420.

[16] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve und J. B. Rothnie Jr. „Query Processing in a System for Distributed Databases (SDD-1)“. In: *ACM Trans. Database Syst.* 6.4 (1981), S. 602–625.

[17] D. Bertsimas, J. Tsitsiklis u. a. „Simulated annealing“. In: *Statistical science* 8.1 (1993), S. 10–15.

[18] A. D. Birrell und B. J. Nelson. „Implementing remote procedure calls“. In: *ACM Transactions on Computer Systems (TOCS)* 2.1 (1984), S. 39–59.

[19] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta und K. Vahi. „The Role of Planning in Grid Computing“. In: *Icaps*. 2003, S. 153–163.

[20] S. Bokhari. „On the Mapping Problem“. In: *Computers, IEEE Transactions on* C-30.3 (1981), S. 207–214.

[21] B. J. Bonfils und P. Bonnet. „Adaptive and decentralized operator placement for in-network query processing“. In: *IPSN'03: Proceedings of the 2nd international conference on Information processing in sensor networks*. Palo Alto, CA, USA: Springer-Verlag, 2003, S. 47–62.

[22] P. Bonnet, J. Gehrke und P. Seshadri. „Towards sensor database systems“. In: *Mobile Data Management*. Springer. 2001, S. 3–14.

[23] C. Bornhövd, M. Cilia, C. Liebig und A. Buchmann. „An infrastructure for meta-auctions“. In: *Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop on*. 2000, S. 21–30.

[24] S. Bouchenak. „Making Java Applications Mobile or Persistent“. In: *6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001), San Antonio, Texas, USA*. USENIX, 2001, S. 159–172.

[25] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte und D. Winer. *Simple object access protocol (SOAP) 1.1*. 2000.

[26] D. Bradshaw, B. Nainani, K. MacDowell und D. Raphaely. „Oracle9i Application Developer's Guide—Advanced Queuing“. In: *Oracle Corporation* (2002).

[27] J. Brüning, P. Forbrig, E. Seib und M. Zaki. „On the Suitability of Activity Diagrams and ConcurTaskTrees for Complex Event Modeling“. In: *Perspectives in Business Informatics Research*. Hrsg. von N. Aseeva, E. Babkin und O. Kozyrev. Bd. 128. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2012, S. 54–69.

[28] D. Burdescu, M. Brezovan und D. MARGHITU. „High level petri nets and rule based systems for discrete event system modelling“. In: *International Journal of Smart Engineering System Design* 3 (2001), S. 81–97.

[29] S. Camazine, N. R. Franks, J. Sneyd, E. Bonabeau, J.-L. Deneubourg und G. Theraula. *Self-Organization in Biological Systems*. Princeton, NJ, USA: Princeton University Press, 2001.

[30] A. Carzaniga. „Architectures for an Event Notification Service Scalable to Wide-area Networks“. Diss. Politecnico de Milano, 1998.

[31] A. Carzaniga, D. S. Rosenblum und A. L. Wolf. „Design and Evaluation of a Wide-area Event Notification Service“. In: *ACM Trans. Comput. Syst.* 19.3 (2001), S. 332–383.

[32] A. Carzaniga und A. L. Wolf. *Fast Forwarding for Content-Based Networking*. Techn. Ber. CU-CS-922-0. Boulder, Colorado: University of Colorado, Department of Computer Science, 2002.

[33] A. Carzaniga und A. L. Wolf. „Forwarding in a Content-based Network“. In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '03. Karlsruhe, Germany: ACM, 2003, S. 163–174.

[34] C. Castel, L. Chaudron und C. Tessier. „What is going on? A high level interpretation of sequences of images“. In: *4th European conference on computer vision, Workshop on conceptual descriptions from images*. Cambridge, UK, 1996.

[35] S. Chakravarthy, V. Krishnaprasad, E. Anwar und S.-K. Kim. „Composite Events for Active Databases: Semantics, Contexts and Detection“. In: *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, S. 606–617.

[36] S. Chakravarthy, V. Krishnaprasad, E. Anwar und S.-K. Kim. „Composite events for active databases: Semantics, contexts and detection“. In: *VLDB*. Bd. 94. 1994, S. 606–617.

[37] S. Chakravarthy und D. Mishra. „Snoop: An expressive event specification language for active databases“. In: *Data & Knowledge Engineering* 14.1 (1994), S. 1–26.

[38] X. Chang. „Network simulations with OPNET“. In: *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*. ACM. 1999, S. 307–314.

[39] A. Cheung und H.-A. Jacobsen. „Dynamic Load Balancing in Distributed Content-Based Publish/Subscribe“. In: *Middleware 2006*. Hrsg. von M. Steen und M. Henning. Bd. 4290. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, S. 141–161.

[40] M. Cilia, L. Fiege, C. Haul, A. Zeidler und A. P. Buchmann. „Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware“. In: *Proceedings of the 2nd International Workshop on Distributed Event-based Systems*. DEBS '03. San Diego, California: ACM, 2003, S. 1–8.

[41] L. P. Clare, G. J. Pottie und J. R. Agre. *Self-organizing distributed sensor networks*. 1999.

[42] T. C. Collier und C. Taylor. „Self-organization in sensor networks“. In: *Journal of Parallel and Distributed Computing* 64.7 (2004), S. 866 –873.

[43] *Common Object Request Broker Architecture (CORBA) Event Service Specification*. Version 1.2. Object Management Group (OMG). 2004.

[44] *Common Object Request Broker Architecture (CORBA) Notification Service Specification*. Version 1.1. Object Management Group (OMG). 2004.

[45] *CORBA Event Service Specification*. Norm. 2001.

[46] P. Costa, M. Migliavacca, G. P. Picco und G. Cugola. „Introducing Reliability in Content-based Publish-subscribe Through Epidemic Algorithms“. In: *Proceedings of the 2Nd International Workshop on Distributed Event-based Systems*. DEBS '03. San Diego, California: ACM, 2003, S. 1–8.

[47] G. Cugola, E. Di Nitto und A. Fuggetta. „The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS“. In: *IEEE Trans. Softw. Eng.* 27.9 (2001), S. 827–850.

[48] F. Dabek, R. Cox, F. Kaashoek und R. Morris. „Vivaldi: A Decentralized Network Coordinate System“. In: *Proceedings of the ACM SIGCOMM '04 Conference*. Portland, Oregon, 2004.

[49] J. Dai, M. Morisio und J. Bernal. „A seamless services migration framework with JVM Tool Interface“. In: *Computer Science and Education (ICCSE), 2010 5th International Conference on*. 2010, S. 106–110.

[50] C. J. Date. *An Introduction to Database Systems*. 8. Aufl. Boston, MA: Pearson Addison-Wesley, 2004.

[51] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey u. a. „The Hipac project: Combining active databases and timing constraints“. In: *ACM Sigmod Record* 17.1 (1988), S. 51–70.

[52] K. R. Dittrich und S. Gatziu. „Time issues in active database systems“. In: *Proceedings of the International Workshop on Infrastructure for Temporal Databases, Arlington, TX*. Citeseer. 1993.

[53] F. Dressler. *Self-Organization in Sensor and Actor Networks*. Chichester: John Wiley & Sons, 2007, S. 384.

[54] S. Duarte, J. L. Martins, H. J. Domingos und N. Preguiça. „A Case Study on Event Dissemination in an Active Overlay Network Environment“. In: *Proceedings of the 2Nd International Workshop on Distributed Event-based Systems*. DEBS '03. San Diego, California: ACM, 2003, S. 1–8.

[55] W. Ebeling, J. Freund und F. Schweitzer. *Komplexe Strukturen: Entropie und Information*. Stuttgat-Leipzig: Teubner Verlag, 1998.

[56] J. Eckert, F. Villanueva, R. German und F. Dressler. „Distributed Mass-Spring-Relaxation for Anchor-Free Self-Localization in Sensor and Actor Networks“. In: *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*. 2011, S. 1–8.

[57] J. A. Edwards. „A Two-level Hierarchical Control System for a Distributed Network of Computers.“ AAI7603093. Diss. University of Oklahoma, 1975.

[58] K. Efe. „Heuristic Models of Task Assignment Scheduling in Distributed Systems“. In: *Computer* 15.6 (1982), S. 50–56.

[59] R. Epstein, M. Stonebraker und E. Wong. „Distributed Query Processing in a Relational Data Base System“. In: *Proceedings of the 1978 ACM SIGMOD International Conference on Management of Data*. SIGMOD '78. Austin, Texas: ACM, 1978, S. 169–180.

[60] P. T. Eugster, P. A. Felber, R. Guerraoui und A.-M. Kermarrec. „The many faces of publish/subscribe“. In: *ACM Comput. Surv.* 35.2 (2003), S. 114–131.

[61] P. T. Eugster, R. Guerraoui und C. H. Damm. „On Objects and Events“. In: *SIGPLAN Not.* 36.11 (Okt. 2001), S. 254–269.

[62] E. D. Falkenberg. „Concepts for Modelling Information“. In: *Modelling in Database Management Systems*. Hrsg. von G. M. Nijssen. North-Holland Publishing, 1976, S. 95–109.

[63] E. Fidler, H. a. Jacobsen, G. Li und S. Mankovski. „The padres distributed publish/subscribe system“. In: *In 8th International Conference on Feature Interactions in Telecommunications and Software Systems*. 2005, S. 12–30.

[64] L. Fiege, M. Cilia, G. Mühl und A. Buchmann. „Publish-subscribe grows up: support for management, visibility control, and heterogeneity“. In: *Internet Computing, IEEE* 10.1 (2006), S. 48–55.

[65] L. Fiege, M. Mezini, G. Mühl und A. Buchmann. „Visibility as a central abstraction in event-based systems“. In: *Concrete Communicatio Abstractions of the Next 701 Sistributed Object Systems (ECCOP 2002 Workshop)*. Hrsg. von A. Beugnard, D. L. Sadou S. und E. Jul. ACM, 2002, S. 385 –392.

[66] L. Fiege. „Visibility in event based systems“. Diss. TU Darmstadt, 2005, S. 1–214.

[67] L. Fiege, F. C. Gärtner, O. Kasten und A. Zeidler. „Supporting Mobility in Content-based Publish/Subscribe Middleware“. In: *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Middleware '03. Rio de Janeiro, Brazil: Springer-Verlag New York, Inc., 2003, S. 103–122.

[68] L. Fiege, G. Mühl und F. C. Gärtner. „Modular Event-based Systems“. In: *Knowl. Eng. Rev.* 17.4 (Dez. 2002), S. 359 –388.

[69] G. S. Fishman. *Principles of discrete event simulation*. Techn. Ber. 1978.

[70] C. L. Forgy. „Expert Systems“. In: *Artificial Intelligence*. Hrsg. von P. G. Raeth. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990. Kap. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, S. 324–341.

[71] C. Frank und K. Römer. „Algorithms for Generic Role Assignment in Wireless Sensor Networks“. In: *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*. SenSys '05. San Diego, California, USA: ACM, 2005, S. 230–242.

[72] E. Friedman. *Jess in Action: Rule-based Systems in Java*. Greenwich, CT, USA: Manning Publications Co., 2003.

[73] X. Fu und V. Karamcheti. „Planning for network-aware paths“. In: *Distributed Applications and Interoperable Systems*. Springer. 2003, S. 187–199.

[74] A. Gafni. „Rollback mechanisms for optimistic distributed simulation systems.“ In: *SCS Multiconference on Distributed Simulation*. 1988, S. 61–67.

[75] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Entwurfsmuster, Elemente wiederverwendbarer objektorientierter Software*. Hrsg. von E. Gamma. [6. Aufl.] Addison-Wesley, 2011.

[76] S. Gatziu und K. Dittrich. „Detecting composite events in active database systems using Petri nets“. In: *Research Issues in Data Engineering, 1994. Proceedings Fourth International Workshop on Active Database Systems*. 1994, S. 2–9.

[77] S. Gatziu und K. R. Dittrich. „SAMOS: An active object-oriented database system“. In: *IEEE Data Eng. Bull.* 15.1-4 (1992), S. 23–26.

[78] S. Gatziu und K. R. Dittrich. „Events in an Active Object-Oriented Database System“. English. In: *Rules in Database Systems*. Hrsg. von N. Paton und M. Williams. Workshops in Computing. Springer London, 1994, S. 23–39.

[79] S. Gatziu, A. Geppert und K. R. Dittrich. „The SAMOS active DBMS prototype“. In: *SIGMOD Conference*. 1995, S. 480.

[80] N. H. Gehani, H. V. Jagadish und O. Shmueli. „Composite Event Specification in Active Databases: Model & Implementation“. In: *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, S. 327–338.

[81] N. Ghanem, D. DeMenthon, D. Doermann und L. Davis. „Representation and Recognition of Events in Surveillance Video Using Petri Nets“. In: *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*. 2004, S. 112–112.

[82] J. Gosling. *The Java language specification*. Addison-Wesley Professional, 2000.

[83] J. N. Gray. „Notes on data base operating systems“. In: *Operating Systems*. Springer, 1978, S. 393–481.

[84] S. D. Gribble, M. Welsh, R. Von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph u. a. „The Ninja architecture for robust Internet-scale systems and services“. In: *Computer Networks* 35.4 (2001), S. 473–497.

[85] F. J. Grüneberger, T. Heinze und P. Felber. „Adaptive Selective Replication for Complex Event Processing Systems.“ In: *BD3@ VLDB*. Citeseer. 2013, S. 31–36.

[86] V. Gylys und J. Edwards. „Optimal Partioning of Workload for Distributed Systems“. In: *Proceddings of the Compcon Fall 1976* (1976), S. 353 –357.

[87] J Hahn, S Queins, M Jeckle, B Zengler und C RUPP. „UML 2 glasklar– Praxiswissen für die UML-Modellierung und–Zertifizierung“. In: *Hanser Fachbuchverlag* (2005).

[88] H. Haken. *Information and self-organization: a macroscopic approach to complex systems*. Springer series in synergetics. Springer-Verlag, 1988.

[89] Y. Hassin und D. Peleg. „Sparse communication networks and efficient routing in the plane (extended abstract)“. In: *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. Portland, Oregon, United States: ACM, 2000, S. 41–50.

[90] T. Heider und T. Kirste. „Smart Environments and Self-Organizing Appliance Ensembles“. In: *Mobile Computing and Ambient Intelligence: The Challenge of Multimedia*. Hrsg. von N. Davies, T. Kirste und H. Schumann. Dagstuhl Seminar Proceedings 05181. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.

[91] H.-U. Heiss und M. Schmitz. „Decentralized dynamic load balancing: The particles approach“. In: *Information Sciences* 84.1 (1995), S. 115–128.

[92] M. Hellenschmidt und T. Kirste. „A generic topology for ambient intelligence“. In: *Ambient Intelligence*. Springer, 2004, S. 112–123.

[93] T. R. Henderson, S. Roy, S. Floyd und G. F. Riley. „Ns-3 Project Goals“. In: *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*. WNS2 '06. Pisa, Italy: ACM, 2006.

[94] M. Henning. „The Rise and Fall of CORBA“. In: *Queue* 4.5 (Juni 2006), S. 28–34.

[95] T. Herfet, T. Kirste und M. Schnaider. „EMBASSI Multimodal Assistance for Infotainment and Service Infrastructures“. In: *Computers & Graphics* 25.4 (2001), S. 581–592.

[96] K. Herrmann. „Self-Organizing Infrastructures for Ambient Services“. Diss. Technische Universität Berlin, 2006.

[97] K. Herrmann. „Self-Organizing Infrastructures for Ambient Services“. In: *KiVS*. Hrsg. von T. Braun, G. Carle und B. Stiller. Informatik Aktuell. Springer, 2007, S. 299–306.

[98] K. Herrmann. „Self-organizing Replica Placement - A Case Study on Emergence“. In: *First International Conference on Self-Adaptive and Self-Organizing Systems, 2007. SASO '07.* 2007, S. 13–22.

[99] K. Herrmann. „Self-Organized Service Placement in Ambient Intelligence Environments“. In: *ACM Trans. Auton. Adapt. Syst.* 5.2 (2010), S. 1–39.

[100] K. Herrmann, K. Geihs und G. Mühl. „Ad hoc Service Grid - A Self-Organizing Infrastructure for Mobile Commerce“. In: *Proceedings of the IFIP TC8 Working Conference on Mobile Information Systems (MOBIS 2004), IFIP – International Federation for Information Processing*. Springer, 2004, S. 15–17.

[101] K. Herrmann, M. Werner und G. Mühl. „A Methodology for Classifying Self-Organizing Software Systems“. In: *International Transactions on Systems Science and Applications* 2.1 (2006), S. 41–50.

[102] A. Hinze. „Efficient Filtering of Composite Events“. In: *Proceedings of the British National Database Conference*. 2003, S. 207–225.

[103] A. Hinze und A. Voisard. „A parameterized algebra for event notification services“. In: *Temporal Representation and Reasoning, 2002. TIME 2002. Proceedings. Ninth International Symposium on*. IEEE. 2002, S. 61–63.

[104] A. Hinze und A. Voisard. „EVA: an event algebra supporting complex event specification“. In: *Information Systems* 48 (2015), S. 1–25.

[105] A. Hinze und A. Voisard. *EVA: An EVent Algebra supporting adaptivity and collaboration in event-based systems*. Techn. Ber. Humboldt Universität Berlin, University of Waikato, New Zealand, Fraunhofer Institut füt Software und Systems Engineering (ISST), Berlin, Germany und Freie Universität, Berlin, Germany, 2009.

[106] J. Holland. *Adaption in Artificial Systems*. 1975.

[107] A. Howard, M. J. Matarić und G. Sukhatme. „Relaxation on a mesh: a formalism for generalized localization“. In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. Bd. 2. IEEE. 2001, S. 1055–1060.

[108] Y. Huang und H. Garcia-Molina. „Publish/Subscribe in a mobile enviroment“. In: *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*. MobiDe '01. Santa Barbara, California, USA: ACM, 2001, S. 27–34.

[109] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek und K. Tzoumas. „Opening the black boxes in data flow optimization“. In: *Proceedings of the VLDB Endowment* 5.11 (2012), S. 1256–1267.

[110] M. A. Jaeger, G. Mühl, M. Werner und H. Parzyjegla. „Reconfiguring Self-Stabilizing Publish/Subscribe Systems“. In: *17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2006)*. Hrsg. von R. State, S. van Meer, O. Declan und T. Pfeifer. Bd. 4269 of LNCS. Dublin, Ireland: Springer, Okt. 2006, S. 233–238.

[111] M. A. Jaeger, H. Parzyjegla, G. Mühl und K. Herrmann. „Self-organizing broker topologies for publish/subscribe systems“. In: *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. Seoul, Korea: ACM, 2007, S. 543–550.

[112] M. Jarke und J. Koch. „Query Optimization in Database Systems“. In: *ACM Comput. Surv.* 16.2 (Juni 1984), S. 111–152.

[113] *Java Message Service (JMS) Specification*. 1.1. Sun Microsystems, Inc. 2002.

[114] K. R. Jayaram und P. Eugster. „Scalable Efficient Composite Event Detection“. In: *Proceedings of the 12th International Conference on Coordination Models and Languages*. COORDINATION'10. Amsterdam, The Netherlands: Springer-Verlag, 2010, S. 168–182.

[115] D. Jefferson und H. A. Sowizral. „Fast concurrent simulation using the time warp mechanism“. In: (1982).

[116] C. Jenny. International Business Machines Corporation, Research Division, 1977.

[117] A. Jentzsch, T. Pape und S. Pasewaldt. *Proceedings of the 8th Joint Workshop of the German Research Training Groups in Computer Science*. Books On Demand - Proquest, 2014.

[118] R Jeyarani, N Nagaveni und R. V. Ram. „Self Adaptive Particle Swarm Optimization for Efficient Virtual Machine Provisioning in Cloud“. In: *International Journal of Intelligent Information Technologies* 7.2 (2011), S. 25–44.

[119] M. Kafil und I. Ahmad. „Optimal task assignment in heterogeneous distributed computing systems“. In: *Concurrency, IEEE* 6.3 (1998), S. 42–50.

[120] H. Kamiya, H. Mineno, N. Ishikawa, T. Osano und T. Mizuno. „Composite Event Detection in Heterogeneous Sensor Networks“. In: *IEEE/IPSJ International Symposium on Applications and the Internet* 0 (2008), S. 413–416.

[121] T. Karnagel, D. Habich, B. Schlegel und W. Lehner. „Heterogeneity-Aware Operator Placement in Column-Store DBMS“. In: *Datenbank-Spektrum* 14.3 (2014), S. 211–221.

[122] Q. Ke, M. Isard und Y. Yu. „Optimus: a dynamic rewriting framework for data-parallel execution plans“. In: *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM. 2013, S. 15–28.

[123] J. O. Kephart und D. M. Chess. „The Vision of Autonomic Computing“. In: *Computer* 36.1 (2003), S. 41–50.

[124] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.-L. Wu, H. Andrade und B. Gedik. „COLA: Optimizing stream processing applications via graph partitioning“. In: *Middleware 2009*. Springer, 2009, S. 308–327.

[125] T. Kichkaylo. „Timeless planning and the component placement problem“. In: *ICAPS Workshop on Planning and Scheduling for Web and Grid Services*. 2004.

[126] T. Kichkaylo und V. Karamcheti. „Optimal Resource-Aware Deployment Planning for Component-Based Distributed Applications“. In: *HPDC*. IEEE Computer Society, 2004, S. 150–159.

[127] T. Kohonen. „The Self-Organizing Map“. In: *Proceedings of the IEEE* 78 (1990), S. 1464–1480.

[128] D. Kossmann. „The State of the Art in Distributed Query Processing“. In: *ACM Comput. Surv.* 32.4 (Dez. 2000), S. 422–469.

[129] S. Lacour, C. Pérez und T. Priol. „Generic application description model: toward automatic deployment of applications on computational grids“. In: *GRID*. IEEE, 2005, S. 284–287.

[130] B. W. Lampson und H. E. Sturgis. *Crash recovery in a distributed data storage system*. Xerox Palo Alto Research Center Palo Alto, California, 1979.

[131] J. Ledlie, M. Mitzenmacher, M. I. Seltzer und P. Pietzuch. „Wired Geometric Routing.“ In: *IPTPS*. 2007.

[132] R. Lewis. *Advanced Messaging Applications with MSMQ and MQSeries*. Que, 2000.

[133] G. Li, A. Cheung, S. Hou, S. Hu, V. Muthusamy, R. Sherafat, A. Wun, H.-A. Jacobsen und S. Manovski. „Historic Data Access in Publish/Subscribe“. In: *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*. DEBS '07. Toronto, Ontario, Canada: ACM, 2007, S. 80–84.

[134] G. Li und H.-A. Jacobsen. „Composite subscriptions in content-based publish/subscribe systems“. In: *In ACM/IFIP/USENIX 6th International Middleware Conference*. 2005, S. 249–269.

[135] G. Li, V. Muthusamy und H.-A. Jacobsen. „Adaptive Content-based Routing in General Overlay Topologies“. In: *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '08. Leuven, Belgium: Springer-Verlag New York, Inc., 2008, S. 1–21.

[136] V. M. Lo. *Task assignment in distributed systems*. Champaign, IL, USA, 1983.

[137] V. M. Lo. „Heuristic algorithms for task assignment in distributed systems“. In: *IEEE Transactions on Computers*. 11. IEEE, 1988, S. 1384–1397.

[138] D. C. Luckham und J. Vera. „An event-based architecture definition language“. In: *Software Engineering, IEEE Transactions on* 21.9 (1995), S. 717–734.

[139] S. Madden, M. J. Franklin, J. Hellerstein und W. Hong. „TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks“. In: *In OSDI*. 2002.

[140] M. Mansouri-Samani und M. Sloman. „GEM: A generalized event monitoring language for distributed systems“. In: *Distributed Systems Engineering* 4.2 (1997), S. 96.

[141] T. Martins, A. Sato und M. Tsuzuki. *Adaptive Neighborhood Heuristics for Simulated Annealing over Continuous Variables*. INTECH Open Access Publisher, 2012.

[142] F. Mattern. „Algorithms for distributed termination detection“. In: *Distributed computing* 2.3 (1987), S. 161–175.

[143] F. Mattern. „Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation“. In: *Journal of Parallel and Distributed Computing* 18.4 (1993). (Reprinted in: Z. Yang, T.A. Marsland (Eds.), "Global States and Time in Distributed Systems", IEEE, 1994, S. 27-36), S. 423–434.

[144] I. S. Microsystems. *Java Message Service, Version 1.0.2 (JMS specification)*. Techn. Ber. Sun Microsystems, Inc., 1998.

[145] H. Modi und C. S. Serra. *Phantom: Heterogeneous Process Migration in Java*. Techn. Ber. Madison, WI, USA: University of Wisconsin-Madison, Computer Science Department, 1998.

[146] A. Montresor und M. Jelasity. „PeerSim: A scalable P2P simulator“. In: *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. 2009, S. 99–100.

[147] I. Motakis und C. Zaniolo. „Formal semantics for composite temporal events in active database rules“. In: *Journal of Systems Integration* 7.3-4 (1997), S. 291–325.

[148] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein und R. Varma. *Query Processing, Resource Management, and Approximation ina Data Stream Management System*. Technical Report 2002-41. Stanford InfoLab, 2002.

[149] G. Mühl. „Large-Scale Content-Based Publish/Subscribe Systems“. Diss. Technische Universität Darmstadt, 2002.

[150] G. Mühl und L. Fiege. „Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs“. In: *IEEE Distributed Systems Online (DSOnline)* 2.7 (2001).

[151] G. Mühl und L. Fiege. „Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs“. In: *IEEE Distributed Systems Online (DSOnline)* 2.7 (2001).

[152] G. Mühl, L. Fiege, F. C. Gärtner und A. Buchmann. „Evaluating Advanced Routing Algorithms for Content-Based Publish/Subscribe Systems“. In: *In Proc. MASCOTS 2002*. IEEE Press, 2002, S. 167–176.

[153] G. Mühl, L. Fiege und P. R. Pietzuch. *Distributed Event-Based Systems*. Springer, Aug. 2006.

[154] G. Mühl, A. Schröter, H. Parzyjegla, S. Kounev und J. Richling. „Stochastic Analysis of Hierarchical Publish/Subscribe Systems“. In: *Proceedings of the 15th European Conference on Parallel Processing (Euro-Par 2009)*. Hrsg. von H. Sips, D. Epema und H.-X. Lin. Bd. 5704. LNCS. Delft, The Netherlands: Springer-Verlag, 2009, 97–109.

[155] G. Mühl, A. Ulbrich, K. Herrmann und T. Weis. „Disseminating Information to Mobile Clients Using Publish-Subscribe“. In: *IEEE Internet Computing* 8.3 (2004), S. 46–53.

[156] G. Mühl, A. Ulbrich und H. Ritter. „Content Evolution Driven Data Propagation in Wireless Sensor Networks“. In: *2. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*. Hrsg. von T. Fuhrmann. Karlsruhe, Germany, Feb. 2004, S. 55–57.

[157] G. Mühl, M. Werner, M. A. Jaeger, K. Herrmann und H. Parzyjegla. „On the Definitions of Self-Managing and Self-Organizing Systems“. In: *KiVS 2007 Workshop: Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS 2007)*. Informatik aktuell. Berlin: VDE Verlag, März 2007, S. 291–301.

[158] M. Mühlhäuser und A. Schill. „Remote Procedure Call“. German. In: *Software Engineering für verteilte Anwendungen*. Springer-Lehrbuch. Verlag Springer Berlin Heidelberg, 1992, S. 71–140.

[159] S. Mullender, Hrsg. *Distributed Systems*. 2nd edition. Addison Wesley, 1993.

[160] C. Müller-Schloer. „Organic computing-on the feasibility of controlled emergence“. In: *International Conference on Hardware/Software Codesign and System Synthesis, 2004. CODES+ ISSS 2004.* IEEE. 2004, S. 2–5.

[161] C. Müller-Schloer, C. von der Malsburg und R. P. Würt. „Organic computing“. In: *Informatik-Spektrum* 27.4 (2004), S. 332–336.

[162] C. Müller-Schloer, H. Schmeck und T. Ungerer. *Organic Computing-A Paradigm Shift for Complex Systems.* Springer-Verlag, 2011.

[163] R. V. Nehme, K. Works, C. Lei, E. A. Rundensteiner und E. Bertino. „Multi-route query processing and optimization“. In: *Journal of computer and system sciences* 79.3 (2013), S. 312–329.

[164] G. Nicolis und I. Prigogine. *Self-organization in nonequilibrium systems: from dissipative structures to order through fluctuations.* A Wiley-Interscience Publication. Wiley, 1977.

[165] D. O'Keeffe. *Distributed Complex Event Detection for Pervasive Computing.* Techn. Ber. 783. University of Cambridge, Juni 2010.

[166] S. Paal, R. Kammüller und B. Freisleben. „Dynamic Software Deployment with Distributed Application Repositories“. In: *KiVS.* Hrsg. von P. Müller, R. Gotzhein und J. B. Schmitt. Informatik Aktuell. Springer, 2005, S. 41–52.

[167] J. Pan und R. Jain. „A survey of network simulation tools: Current status and future developments“. In: *Washington University in St. Louis, Tech. Rep* (2008).

[168] H. Parzyjegla, M. Jaeger, G. Mühl und T. Weis. „Model-Driven Development and Adaptation of Autonomous Control Applications“. In: *Distributed Systems Online, IEEE* 9.11 (2008), S. 2–2.

[169] H. Parzyjegla. „Engineering Publish/Subscribe Systems and Event-driven Applications“. Diss. Universität Rostock, Fakultät für Informatik und Elektrotechnik, 2013.

[170] H. Parzyjegla, D. Graff, A. Schröter, J. Richling und G. Mühl. „Design and Implementation of the Rebeca Publish/Subscribe Middleware“. In: *From Active Data Management to Event-Based Systems and More.* Hrsg. von K. Sachs, I. Petrov und P. Guerrero. Bd. 6462. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, S. 124–140.

[171] H. Parzyjegla, M. A. Jaeger, G. Mühl und T. Weis. „A Model-driven Approach to the Development of Autonomous Control Applications“. In: *Proceedings of the 1st Workshop on Model-driven Software Adaptation (M-ADAPT '07) at ECOOP 2007.* Berlin, Germany, 2007, S. 25–27.

[172] H. Parzyjegla, G. Mühl und M. A. Jaeger. „Reconfiguring Publish/Subscribe Overlay Topologies“. In: *5th Intl. Workshop on Distributed Event-based Systems (DEBS'06).* Lisbon, Portugal: IEEE Press, 2006, S. 29.

[173] H. Parzyjegla, A. Schröter, E. Seib, S. Holzapfel, M. Wander, J. Richling, A. Wacker, H.-U. Heiß, G. Mühl und T. Weis. „Model-Driven Development of Self-organising Control Applications (MODOC)“. In: *Organic Computing - A Paradigm Shift for Complex Systems*. Hrsg. von C. Müller-Schloer, H. Schmeck und T. Ungerer. Bd. 1. Autonomic Systems. Springer Basel, Mai 2011, S. 131–144.

[174] N. W. Paton und O. Díaz. „Active database systems“. In: *ACM Comput. Surv.* 31.1 (1999), S. 63–103.

[175] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh und M. Seltzer. „Network-aware operator placement for stream-processing systems“. In: *In ICDE*. 2006.

[176] P. R. Pietzuch und J. Bacon. „Hermes: A Distributed Event-Based Middleware Architecture“. In: *ICDCS Workshops*. 2002, S. 611–618.

[177] P. R. Pietzuch, B. Shand und J. Bacon. „A framework for event composition in distributed systems“. In: *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*. Rio de Janeiro, Brazil: Springer-Verlag New York, Inc., 2003, S. 62–82.

[178] P. R. Pietzuch, B. Shand und J. Bacon. „Composite Event Detection as a Generic Middleware Extension.“ In: *IEEE Network* 18.1 (2004), S. 44–55.

[179] C. G. Plaxton, R. Rajaraman und A. W. Richa. „Accessing Nearby Copies of Replicated Objects in a Distributed Environment“. In: *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*. SPAA '97. Newport, Rhode Island, USA: ACM, 1997, S. 311–320.

[180] N. Pollner, C. Steudtner und K. Meyer-Wegener. „Placement-Safe Operator-Graph Changes in Distributed Heterogeneous Data Stream Systems“. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2015) - Workshopband, 2.-3. März 2015, Hamburg, Germany*. Hrsg. von N. Ritter, A. Henrich, W. Lehner, A. Thor, S. Friedrich und W. Wingerath. Bd. 242. LNI. GI, 2015, S. 61–70.

[181] S. Poslad. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Bd. 1. Auflage. Hoboken, USA: John Wiley & Sons, Mai 2009.

[182] C. Prehofer und C. Bettstetter. „Self-organization in communication networks: principles and design paradigms“. In: *Communications Magazine, IEEE* 43.7 (2005), S. 78–85.

[183] C. Prehofer und C. Bettstetter. „Self-organization in communication networks: principles and design paradigms“. In: *Communications Magazine, IEEE* 43.7 (2005), S. 78–85.

[184] R. Raman, M. Livny und M. Solomon. „Matchmaking: Distributed resource management for high throughput computing“. In: *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*. IEEE. 1998, S. 140–146.

[185] G. F. Riley und T. R. Henderson. „The ns-3 network simulator“. In: *Modeling and Tools for Network Simulation*. Springer, 2010, S. 15–34.

[186] T. Robertazzi und P. Sarachik. „Self-organizing communication networks“. In: *Communications Magazine, IEEE* 24.1 (1986), S. 28–33.

[187] A. Schill und T. Springer. *Verteilte Systeme: Grundlagen und Basistechnologien*. Springer-Verlag, 2012.

[188] H. Schmeck. „Organic Computing.“ In: *Künstliche Intelligenz* 19.3 (2005), S. 68.

[189] H. Schmeck. „Organic computing-a new vision for distributed embedded systems“. In: *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*. IEEE. 2005, S. 201–203.

[190] A. Schröter, D. Graff, G. Mühl, J. Richling und H. Parzyjegla. „Self-optimizing Hybrid Routing in Publish/Subscribe Systems“. In: *DSOM '09: Proceedings of the 20th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. Venice, Italy: Springer-Verlag, 2009, S. 111–122.

[191] A. Schröter, G. Mühl, S. Kounev, H. Parzyjegla und J. Richling. „Stochastic Performance Analysis and Capacity Planning of Publish/Subscribe Systems“. In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. DEBS '10. Cambridge, United Kingdom: ACM, 2010, S. 258–269.

[192] A. Schröter, G. Mühl, J. Richling und H. Parzyjegla. „Adaptive Routing in Publish/Subscribe Systems Using Hybrid Routing Algorithms“. In: *Proceedings of the 7th Workshop on Reflective and Adaptive Middleware*. ARM '08. Leuven, Belgium: ACM, 2008, S. 51–52.

[193] E. Seib, H. Parzyjegla und G. Mühl. „Distributed Composite Event Detection in Publish/Subscribe Networks – A Case for Self-Organization“. In: *Proceedings of the Workshops der wissenschaftlichen Konferenz Kommunikation in verteilten Systemen 2011 (WowKiVS 2011)*. Hrsg. von T. Margaria, J. Padberg und G. Taentzer. Bd. 37. Electronic Communications of the EASST, März 2011, 2:1 –2:12.

[194] E. Seib, H. Parzyjegla und G. Mühl. „Adaptive distributed composite event detection“. In: *Proceedings of the 11th International Workshop on Adaptive and Reflective Middleware*. Hrsg. von P. Ferreira, L. Veiga und F. Araújo. ACM, 2012, 2:1 –2:6.

[195] S. Srikantaiah, A. Kansal und F. Zhao. „Energy aware consolidation for cloud computing“. In: *Proceedings of the 2008 conference on Power aware computing and systems*. Bd. 10. San Diego, California. 2008.

[196] B. Srivastava. „Realplan: Decoupling causal and resource reasoning in planning“. In: *AAAI/IAAI*. 2000, S. 812–818.

[197] U. Srivastava, K. Munagala und J. Widom. „Operator placement for in-network stream query processing“. In: *In PODS*. 2005, S. 250–258.

[198] H. S. Stone und S. H. Bokhari. „Control of distributed processes“. In: *Computer* 11.7 (1978), S. 97–106.

[199] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev und R. Snodgrass. *Temporal databases: theory, design, and implementation*. Benjamin-Cummings Publishing Co., Inc., 1993.

[200] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler und A. P. Buchmann. „A peer-to-peer approach to content-based publish/subscribe“. In: *DEBS*. Hrsg. von H.-A. Jacobsen. ACM, 2003.

[201] A. Ulbrich, G. Mühl, T. Weis und K. Geihs. „Programming abstractions for content-based publish/subscribe in object-oriented languages“. In: *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*. Springer Berlin Heidelberg, 2004, S. 1538–1557.

[202] A. Varga u. a. „The OMNeT++ discrete event simulation system“. In: *Proceedings of the European simulation multiconference (ESM'2001)*. Bd. 9. S 185. sn. 2001, S. 65.

[203] S. D. Viglas und J. F. Naughton. „Rate-based query optimization for streaming information sources“. In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM. 2002, S. 37–48.

[204] D. West. „Optimising Time Warp: Lazy Rollback and Lazy Reevaluation.“ Diss. University of Calgary, 1988.

[205] S. A. Wolfman und D. S. Weld. „Combining linear programming and satisfiability solving for resource planning“. In: *The Knowledge Engineering Review* 16.01 (2001), S. 85–99.

[206] K. Works, E. A. Rundensteiner und E. Agu. „Optimizing adaptive multi-route query processing via time-partitioned indices“. In: *Journal of Computer and System Sciences* 79.3 (2013), S. 330–348.

[207] G. Xu, Y. Ding, J. Zhao, L. Hu und X. Fu. „A novel artificial bee colony approach of live virtual machine migration policy using bayes theorem“. In: *The Scientific World Journal* 2013 (2013).

[208] Y. Yao und J. Gehrke. „The cougar approach to in-network query processing in sensor networks“. In: *ACM Sigmod Record* 31.3 (2002), S. 9–18.

[209] F. E. Yates, A. Garfinke und D. O. Walter. *Self-organizing systems : the emergence of order*. English. Plenum Press New York, 1987, S. xix, 661.

[210] L. Ying, Z. Liu, D. F. Towsley und C. H. Xia. „Distributed Operator Placement and Data Caching in Large-Scale Sensor Networks“. In: *INFOCOM 2008. 27th IEEE Intern. Conf. on Computer Communication*. Phoenix, AZ, USA, 2008, S. 977–985.

[211] K. Yordanova, A. Gladisch, K. Duske, D. Janusz und S. Bader. *Proceedings of the Joint Workshop of the German Research Training Groups in Computer Science*. Books On Demand - Proquest, 2012.

[212] A. Zeidler. „A Distributed Publish/Subscribe Notification Service for Pervasive Environments“. Diss. Darmstadt, Deutschland: Technische Universität Darmstadt, 2004.

[213] A. Zeidler und L. Fiege. „Mobility Support with REBECA“. In: *32nd International Conference on Distributed Computing Systems Workshops* 0 (2003), S. 354.

[214] Q. Zhang, C. H. Foh, B.-C. Seet und A. Fong. „Location estimation in wireless sensor networks using spring-relaxation technique“. In: *Sensors* 10.5 (2010), S. 5171–5192.

[215] R. Zhang und E. A. Unger. „Event specification and detection“. In: (1996).

[216] J. Zhao, L. Hu, Y. Ding, G. Xu und M. Hu. „A Heuristic Placement Selection of Live Virtual Machine Migration for Energy-Saving in Cloud Computing Environment“. In: (2014).

[217] D. Zimmer und R. Unland. „On the semantics of complex events in active database management systems“. In: *Proceedings, 15th International Conference on Data Engineering, 1999*. IEEE. 1999, S. 392–399.

[218] H. Zimmermann. „OSI reference model–The ISO model of architecture for open systems interconnection“. In: *Communications, IEEE Transactions on* 28.4 (1980), S. 425–432.

Erklärung

Hiermit erkläre ich, dass ich die eingereichte Dissertation mit dem Titel „*Automatisches Deployment und adaptive Platzierung ubiquitärer Anwendungen*“ selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Ich habe bisher noch keinen Promotionsversuch unternommen.

Rostock, 6. Dezember 2016 Enrico Seib

Thesen

1. Vernetzte, interagierende Anwendungen und Geräte formen als intelligente Umgebung ein Ensemble. In diesem Verbund bietet das Ensemble dem Nutzer vielfältige Informationen und Dienste an.

2. Die Geräte des Ensembles sind heterogen und unterscheiden sich hinsichtlich ihrer Systemarchitektur, ihres Betriebssystems und der zur Verfügung stehenden Ressourcen. Die Anwendungen differieren vor allem in ihren spezifizierten Ressourcenanforderungen bezüglich der sie ausführenden Geräte.

3. Publish/Subscribe ist ein geeignetes Kommunikationsparadigma für dynamische Umgebungen. Anwendungen und Geräte können dem Ensemble dynamisch beitreten oder es verlassen. Als ereignisbasiertes Kommunikationsparadigma lässt Publish/Subscribe die Ensembleteilnehmer entkoppelt miteinander kommunizieren und kooperieren.

4. Geräte in intelligenten Umgebungen sind vielfach mobil und besitzen begrenzte Energieressourcen. Die Kommunikation zwischen Geräten und Anwendungen verbraucht diese Ressourcen in entscheidendem Maße. Kommunikation ist ein bestimmender Kostenfaktor.

5. Die Umsetzung von Publish/Subscribe in Form einer Middleware bietet Kommunikationsfunktionalität als einheitliche Schnittstelle an. Zudem abstrahiert die Middleware von der zugrunde liegenden Hardware, indem sie eine einheitliche Ausführungsplattform für die auszuführenden Anwendungen unter Beachtung der zur Verfügung stehenden Ressourcen bereitstellt.

6. Ensembles aus Anwendungen und Geräten mit unterschiedlichen Ressourcen und Anforderungen können sehr schnell unübersichtlich werden. Sollen die Anwendungen gemäß ihrer Anforderungen und der im Ensemble verfügbaren Ressourcen manuell platziert werden, ist dies für die Nutzer zunehmend unmöglich.

7. Eine ungeschickte Anwendungsplatzierung kann zu Überlastsituationen im Netzwerk und dessen Komponenten sowie zu qualitativen Einbußen bei der Anwendungsausführung führen.

8. Eine einmal getroffene, möglichst gute Platzierungsentscheidung ist, egal wie sie getroffen wurde, durch die dem System inhärente Dynamik nicht dauerhaft. Wenn die Anwendungsplatzierung dauerhaft einem definierten Anspruch an Qualität und Kosten zu genügen hat, muss sie laufend optimiert werden.

9. Zentrale Optimierungsansätze sind in verteilten Umgebungen problematisch. Der Aufwand für eine statistische Optimierung liegt bereits im NP-schweren Bereich. Eine Lösungsmöglichkeit ist der Einsatz einer Heuristik. Um zusätzliche Nachrichten und Kosten durch Kommunikation zu vermeiden, werden lokal verfügbare Informationen verwendet.

10. Sowohl die initiale als auch die laufende Anwendungsplatzierung müssen automatisch und als Dienst der Middleware ausgeführt werden. Damit lassen sich die Vorteile einer Publish/Subscribe-basierten Middleware nutzen und der Nutzer wird von der initialen und laufenden Platzierung der Anwendungen entlastet. Die laufende Platzierung baut auf einer gültigen, nicht zwangsläufig optimalen, Platzierung auf. Eine vorherige Optimierung der initialen Platzierung findet nicht statt. Die zügige Bereitstellung von Diensten durch die Anwendungen hat Vorrang gegenüber einer möglichst guten Platzierung.

11. Der vorgestellte Ansatz für die initiale Platzierung ermöglicht das Finden einer gültigen Platzierung von Anwendungen, falls durch ein Gerät des Ensembles die nötigen Voraussetzungen erfüllt werden. Die laufende Platzierungsanpassung erfolgt gemäß den Anwendungsanforderungen, den verfügbaren Ressourcen und einem Kosten- und dem davon abgeleiteten Kräftemodell. Ziel der Optimierung ist es, Kosten unter Erfüllung der Anwendungsanforderungen und Ressourceneinschränkungen einzusparen. Das Kostenmodell enthält alle mit der Ausführung von Anwendungen einhergehenden Kosten. Die Kosten werden auf der Grundlage lokalen Wissens ermittelt. Die sich durch eine mögliche Platzierungsänderung ergebenden Einsparungen oder Mehrkosten wirken als Kräfte lokal auf die Anwendungsplatzierung ein. Werden die auf die individuelle Anwendungsplatzierung einwirkenden Kräfte verringert, vermindern sich auch die im Gesamtsystem anfallenden Kosten. Die Anwendungsplatzierung lässt sich durch eine Permutation von vier Basisoperationen verändern.

12. Die initiale Platzierung wurde als Dienst in die Middleware integriert und nutzt die vorhandenen Primitive von Publish/Subscribe. Für die laufende Platzierung wurden weitere Protokolle entworfen und implementiert. Die Logik wird in die Broker integriert, welche definierte Schnittstellen der Basisoperationen aufseiten der Anwendungen aufrufen. Der Anwendungsentwickler behält die Kontrolle über deren konkrete Ausgestaltung.

13. Durch die Implementierung der initialen und laufenden Anwendungsplatzierung als Plugins in die bestehende Middleware REBECA nach dem Prinzip der Feature Composition wird diese flexibel erweitert, andere bisher in die Middleware integrierte Funktionalitäten bleiben davon unberührt.

14. Die Evaluierung der erweiterten Middleware zusammen mit dem Simulator PeerSim ermöglicht eine aussagekräftige Beurteilung des Einsparpotentials des vorgestellten Optimierungsansatzes. Durch die Implementierung der Platzierungsanpassung als Plugins werden Experimente mit den spezifizierten Funktionalitäten durchgeführt und die Auswirkungen der automatischen Platzierung unabhängig von anderen Funktionalitäten untersucht.

15. Durch die Evaluation des Ansatzes gelingt der Nachweis, dass mit Hilfe der automatischen Platzierungsanpassung Nachrichten in beträchtlichem Umfang eingespart werden und der Ansatz adaptiv arbeitet. Bei den Simulationen werden unterschiedliche Verarbeitungsketten von Producern, Consumern und Workern genutzt. Die Weiterleitung der Notifikationen ist im Umfeld intelligenter Umgebungen ein bestimmender Kostenfaktor.

16. Die initiale Anwendungsplatzierung hat keine bzw. lediglich marginale Auswirkungen auf die Experimentergebnisse.

17. Der Optimierungsansatz reduziert auch bei unterschiedlichen Verarbeitungsketten und unterschiedlichen Verstärkungen der Ereignisströme die Gesamtanzahl von Nachrichten. Am deutlichsten ist die Reduzierung beim Einsatz der Replikation mit Migration bei Verarbeitungsketten mit mehr Producern als Consumern und einer Verstärkung kleiner als eins.

18. Der Optimierungsansatz ist adaptiv und reagiert auf Veränderungen des Systems. Dies wurde anhand eines Experiments verdeutlicht, bei dem die Anzahl der Produzenten, die Verstärkungen der Ereignisströme oder der Migrationsschwellwert variiert wurden. Nach induzierten Änderungen war der Ansatz stets in der Lage, signifikante Einsparungen zu generieren.