Universität Rostock

Traditio et Innovatio

University of Rostock

Faculty of computer science and electrical engineering
Institute of computer science

Dissertation
for obtaining the academic degree of
Doktor-Ingenieur (Dr.-Ing.)

Towards Reproducible Simulation Studies with JAMES II

Submitted by: Stefan Rybacki

urn:nbn:de:gbv:28-diss2016-0114-0

Dissertation, University of Rostock, Germany
2016

Date of Submission:	May 3, 2016
Date of Defense:	July 21, 2016
Reviewers:	Prof. Dr. rer. nat. habil. Adelinde M. Uhrmacher, University of Rostock, Germany
	Prof. Dr. Oliver Rose, University of the Federal Armed Forces Munich, Germany
	Prof. Luiz Felipe Perrone, Bucknell University, USA

Abstract

Knowing how scientific results are obtained, meaning to know which methods, techniques, algorithms, software etc. were used, is important for those scientific results to be credible. Documentation and provenance data provide exactly this information. However, providing documentation and provenance data is not trivial. In the domain of Modeling & Simulation a multitude of different building blocks take part in conducting a simulation study. Those blocks range from software products, over mathematical models, to analysis and interaction methods. Keeping track of all this information manually is nearly impossible. Thus, a suitable computer-based support is needed.

This thesis proposes a workflow-based approach for conducting simulation studies, using software aided automated documentation of workflow executions. In order for this to work a simulation study is divided into two layers, one dealing with the creation, verification and validation of a simulation model while the other deals with the execution of a simulation experiment with this model.

In this thesis workflows for both layers, covering a broad range of different types of simulation studies are presented and a framework for executing those workflows as well as automatically collecting provenance data and documentation is developed and implemented. Additionally, an approach is presented and evaluated that aims at improving workflow execution in distributed environments by adaptively reacting to environmental changes optimizing execution, e.g., by speed and resource usage, utilizing machine learning techniques.

Zusammenfassung

Wissenschaftliche Resultate gelten als zuverlässig, wenn man weißwie diese zu Stande gekommen sind, d.h. man kann z.B. nachvollziehen welche Methoden, Algorithmen und Softwareprodukte wurden verwendet, um besagte Resultate zu erzeugen. Dokumentation und Provenienz stellen genau solche Informationen bereit. Allerdings ist das Bereitstellen der Dokumentation oder Provenienz nicht immer einfach. Im Bereich der Modellierung und Simulation existieren z.B. eine Vielzahl von unterschiedlichen Komponenten, die an einer Simulationsstudie beteiligt sind. Diese reichen von verschiedenen Softwareprodukten, über mathematische Modelle und Analysen bis hin zu interaktiven Methoden. Einen Überblick über all diese Informationen per Hand zu behalten, ist schwierig bis unmöglich. Hier können automatische computergestützte Methoden helfen.

In dieser Arbeit wird ein workflow-basierter Ansatz vorgestellt, um Simulationsstudien abzubilden und mit Hilfe von Software auszuführen und automatisch zu dokumentieren. Damit dies funktioniert wird eine Simulationstudie in zwei Stufen unterteilt, zum Einen in die Erstellung und Validierung des Simulationsmodells und zum Anderen in die Ausführung eines Simulationsexperiments mit diesem Modell.

Diese Arbeit stellt dabei für beide Stufen Workflows vor, die genutzt werden können, um eine Vielzahl von verschiedenen Simulationsstudien abzubilden. Für die Ausführung der Workflows und automatische Dokumentation derer Ausführung wird ein Framework konzipiert und umgesetzt. Zusätzlich wird ein Ansatz präsentiert und untersucht, der die Ausführung von Workflows in verteilten Umgebungen verbessern kann, indem auf Änderung in der Umgebung adaptiv, unter Einsatz von Techniken aus derm Bereich maschinelles Lernen, reagiert wird und die Ausführung in z.B. in Richtung Geschwindigkeit oder Resourcenverbrauch optimiert wird.

Contents

List of Figures

List of Tables

1 Introduction

A day without sunshine is like, you know, night.

Steve Martin

1.1 Motivation

Where does this information come from? What statistical method was used for the analysis? Questions like this or similar often arise when evaluating scientific results. Whereby it makes no difference whether the results are self conducted or conducted externally, e.g., found in literature. Interestingly, it is often not possible to answer those question because necessary information is simply not available in the documentation and provenance data provided with the scientific results. This is especially surprising, as important decisions are made based on such scientific results. Ideally, the results are correct and of high quality, whereby this cannot be ensured, e.g., it might not be impossible to determine if the appropriate analysis method was used if that information is not given in the documentation. Missing or insufficient documentation or provenance data for that matter, leads to results that are difficult or impossible to reproduce.

The credibility of results highly dependents on the techniques used to produce them. For instance, in the domain of network simulation Pawlikowski et al. identified two important aspects that simulation results need to be credible. On the one hand the use of an appropriate pseudo-random generator of independent uniformly distributed numbers is needed. On the other hand the simulation results need to be analyzed by appropriate analysis techniques. Thus, in order to assess the credibility and quality of simulation results it is imperative to know how those results were obtained. Therefore, incomplete or missing documentation and provenance data ultimately lead to a crisis of credibility (Pawlikowski et al., 2002; Kurkowski et al., 2005; Kurkowski, 2006).

However, typically this information is not missing on purpose. Firstly, information could simply be missing because of an incomplete documentation of the scientific process, although the necessary information was available at the time. Secondly, even with the right intention it is sometimes impossible to provide sufficient documentation because necessary information is not available to the scientist, e.g., software used to conduct scientific analyses or experiments, does not reveal internal information, such as the used analysis method, its parameters or used algorithms. For instance, the Modeling & Simulation framework JAMES II can be used to conduct simulation experiments or create simulation models. However, by default it does not provide documentation, hiding most of the internal information from the user, e.g., used plug-ins during simulation. This is even more amplified as JAMES II supports an arbitrary number of plug-ins which can be selected automatically by JAMES II if needed, making it hard to determine which plug-ins were used during experimentation.

Thus, when information is available but is not used it directly falls into the responsibilities of the scientist. A possible approach to cope with this issue is to provide a system that guides and assists the scientist during generation of documentation. However, even with guidance the resulting documentation might not be sufficient, therefore ideally the scientist does not need to be involved in the documentation process at all. The second problem, if information is not available but needed for documentation this needs to be addressed on the software side. The software needs to be able to provide all the desired information that is imperative for the documentation.

In the industry workflow systems provide support for documentation, execution, monitoring and controlling of processes described using workflows. They help to find and eliminate problems in a production process that might induce errors in the final product (Van Der Aalst and Van Hee, 2004). Thus, the application of workflows and their workflow management systems for Modeling & Simulation processes poses a viable option for addressing the documentation issue and additionally helps to structure and well-define involved processes at the same time.

Introducing workflows for the entire simulation study process would ideally lead to a seamless documentation from end to end. Thus, leading to results of higher quality, more credibility and results that simply are reproducible.

1.2 Contributions

This thesis focuses on the architecture and implementation of reproducible simulation studies with JAMES II. Herein workflows are identified as means to reach this goal.

The first step in the direction of reproducibility is the structuring of the simulation study process. This is achieved by firstly introducing a two layer separation of this process. Layer one deals with the creation of a valid and verified simulation model, whereby for the validation and verification simulation experiments handled by layer two can be used. Since this layer is of interactive nature with a lot of human interaction that requires a high degree of freedom in terms of the order of task execution during execution, a declarative workflow approach is proposed and used to implement a general workflow covering the creation of a simulation model utilizing conceptual model, formal model and different data artifacts.

Additionally, a conceptual prototype is presented that uses the rule system Drools as foundation for the implementation of a workflow management system, easily integrable into JAMES II and specifically tailored to support artifact-based workflows, in particular the one developed in this work.

Layer two deals with the execution of a simulation experiment. For this layer a general workflow for executing simulation experiments with focus on replacing the experimentation layer of JAMES II was developed utilizing, in contrast to layer one, an imperative task-based workflow approach. The developed workflow exhibits a flexible design using the concept of templates and frames in order to cover a wide range of different simulation experiments.

In order to manage and execute the workflow for layer two and similar workflows a framework was developed, called WorMS (Workflows for Modeling & Simulation). As the name suggests, it facilitates the integration of such workflows into Modeling & Simulation software, e.g., JAMES II. WorMS supports the parallel and distributed execution of workflows out of the box. To provide a better work item distribution behavior across distributed nodes compared to methods such as round robin, an adaptive work item scheduling policy was developed. Adaptation is based on machine learning, determining suitability of specific nodes for specific work items, optimizing performance by constantly learning and adapting suitability over time.

Finally, by providing all the aforementioned contributions the whole process of conducting a simulation study becomes documentable, hence reproducible, which allows answering questions such as *What task was executed when?* and *What algorithm, hardware, system, software, etc. facilitated the execution of a specific task?*. This in return leads to more credible scientific research conducted with JAMES II.

To summarize, the following contributions are made by this thesis.

- structuring of the simulation study process, to be documented for reproducibility with the support of workflows
- identification of two layers of the processes that need to be handled by different workflow approaches due to their nature (interactive with high degree of freedom versus non-interactive, automatable and rigid)
- a general workflow representing the process of creating a valid and verified simulation model based on established life-cycle models in the domain of Modeling & Simulation, based on an artifact-based workflow
- a general workflow representing the process of executing a simulation experiment, that is flexible enough to support a wide range of different experiments, similar to experiments JAMES II supports, based on an imperative task-based workflow

- conceptional prototype of a workflow manangement system supporting artifact-based workflows, using a rule system internally for executing and documenting those workflows
- WorMS a framework that supports the integration of imperative task-based workflows into Modeling & Simulation software, which exhibits automatic parallel and distributed execution capabilities and documentation
- in concert with WorMS an adaptive work item scheduling policy, based on machine learning technology was developed that manages work item distribution across nodes optimizing performances by learning which node executes which work item best
- documentation and provenance of executed workflow can be gathered and questions like *What task was executed when, with what data and what was the output?* and *What was used, such as software, hardware, system configuration or algorithm to execute a specific task?* can be answered

1.3 Outline

This thesis comprises two main parts. The first part deals with the structuring of the simulation study process and the derivation of suitable workflow representations for the resulting structure. Chapter 2 presents concepts from the domain of workflow management. Those range from terminology and relations between terms, over workflows and their dominant application and workflow representation options, to how flexibility in workflows can be achieved. Chapter 3 covers established processes in the domain of Modeling & Simulation, called life-cycle models as well as existing workflow approaches already applied in Modeling & Simulation and finishes with the structuring of a simulation study. Chapter 4 focuses on JAMES II and its current state regarding workflow support. Furthermore, Chapter 4 presents the proposed approaches and later developed workflows for the previously introduced structuring of a simulation study.

The second part covers the development and implementation of the introduced workflows. Chapter 5 deals with the presentation of two different developed architectures, needed to support the workflows previously developed. Firstly, a conceptual architecture is given with a possible integration path into JAMES II, while secondly WorMS and its development and implementation is discussed. Additionally, in the context of WorMS a newly created adaptive work item scheduling policy is developed and also covered in this chapter.

Chapter 6 discusses the previous chapters and gives a brief outlook of possible future work, before Chapter 7 concludes and summarizes this thesis.

1.4 Bibliographic Note

The idea of using workflows in the domain of Modeling & Simulation, which is used throughout this thesis was proposed in the following publication, discussing and evaluating requirements for those workflows.

> Rybacki, S., Himmelspach, J., Seib, E., and Uhrmacher, A. (2010). Using workflows in m&s software. In *Winter Simulation Conference (WSC), Proceedings of the 2010*, pages 535–545. IEEE

A first architectural overview of WorMS, the framework for integrating workflows into Modeling & Simulation software, presented in Section 5.2.1 and introduced in Section 4.2.2.1 was firstly published in the following publication.

> Rybacki, S., Himmelspach, J., Haack, F., and Uhrmacher, A. (2011). Worms-a framework to support workflows in m&s. In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, pages 716–727. IEEE

The first application of WorMS as a proof of concept for the feasibility of replacing parts of the experimentation layer of JAMES II with workflows using WorMS was presented in the following publication.

> Rybacki, S., Himmelspach, J., and Uhrmacher, A. M. (2012a). Using workflows to control the experiment execution in modeling and simulation software. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 93–102, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)

In order to provide more flexible workflow descriptions, the concept of templates and frames was integrated into WorMS. Templates and frames are used to create the experimentation workflow for replacing the experimentation layer of JAMES II presented in Section 4.2.2.2 and their integration into WorMS was introduced in the following publication.

> Rybacki, S., Leye, S., Himmelspach, J., and Uhrmacher, A. (2012b). Template and frame based experiment workflows in modeling and simulation software with worms. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*, pages 25–32. IEEE

The idea of using an adaptive work item scheduling policy using reinforcement learning presented in Section 5.2.2 stems from the work presented in the following publications, which developed an adaptive component for selecting simulation algorithms adaptively for increasing simulation execution performance.

> Helms, T., Ewald, R., Rybacki, S., and Uhrmacher, A. M. (2013). A generic adaptive simulation algorithm for component-based simulation systems. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '13, pages 11–22, New York, NY, USA. ACM

> Helms, T., Ewald, R., Rybacki, S., and Uhrmacher, A. M. (2015). Automatic runtime adaptation for component-based simulation algorithms. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 26(1):7

Using a declarative workflow approach, in particular the artifact-based workflow approach for describing the process of model creation and validation as well as the workflow used in Section 4.2.1 was firstly presented in the following publication.

> Rybacki, S., Haack, F., Wolf, K., and Uhrmacher, A. M. (2014). Developing simulation models - from conceptual to executable model and back - an artifact-based workflow approach. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*,

SIMUTools '14, pages 21–30, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)

2 Workflows

Get your facts first, then you can distort them as you please.

Mark Twain

2.1 Terminology and Concepts

In this section the terminology that is later used throughout this thesis as well as the concept of how those terms relate and interact with each other is established. This is necessary because there are a multitude of terms that are used throughout different sources (WfMC, 1999; Center of Excellence, 2015; Van Der Aalst and Van Hee, 2004) in the domain of workflow management.. However, the main concepts those sources identify are very similar or equal to each other. The terms used for them can however differ. To illustrate, the *Workflow Management Coalition* (WfMC) defines a workflow as: "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." (WfMC, 1999). While the Business Process Management - Center of Excellence (BPM - CoE) describes it as: "An orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information." (Center of Excellence, 2015). And last but not least Van der Aalst and van Hee define a workflow as: "A workflow comprises cases, resources and triggers that relate to a particular process." (Van Der Aalst and Van Hee, 2004). All these definitions are very similar and make use of the notion of a (business) process that is to be described, orchestrated or automated by a workflow. By looking at the definitions of the involved process for each of the sources: "A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships." (WfMC, 1999), "[...] a collection of related, structured activities or tasks that produce a specific service or product [...]." (Center of Excellence, 2015). and "A business process is one focused upon the production of particular products." (Van Der Aalst and Van Hee, 2004). The use of different terms can be identified for the following concepts. Firstly, there is the concept of product, which is the result of performing a business process, also called goal, objective or service. Secondly, there is the concept of process, also called pattern or procedural rules. Thirdly, there is the concept of task, that builds up a process, also called procedure, activity or action.

In the following, the terminology and relations between terms used throughout this thesis is presented. Generally, there is a distinction between two phases the terms belong to. Firstly, there is the *description* phase in which *Work*, i.e., the real world process, is transformed into a workflow. Secondly, there is the *instantiation/execution* phase in which a *Workflow Management System* instantiates and then executes a workflow.

In Figure 2.1 p. 11 the terms are shown and their relation and dependencies to each other is depicted using differently styled edges. Solid arcs represent a sequential timely relation, meaning for instance that *Work* is firstly transformed into a *Workflow* description which can then be instantiated before execution. Edges with a diamond describe a composed of relation, where the term closest to the diamond is composed of the related term. If no cardinality is provided, a one-to-one relation ship is assumed. Dashed arcs represent instantiation relations, e.g., a *Workflow* is instantiated as *Workflow Instance* by a *Workflow Management System*, while a *Task* is instantiated as *Work Item* in the context of an instantiated *Process* (*Case*). The terminology presented is inspired by Van Der Aalst and Van Hee (2004). The terms are used as follows:

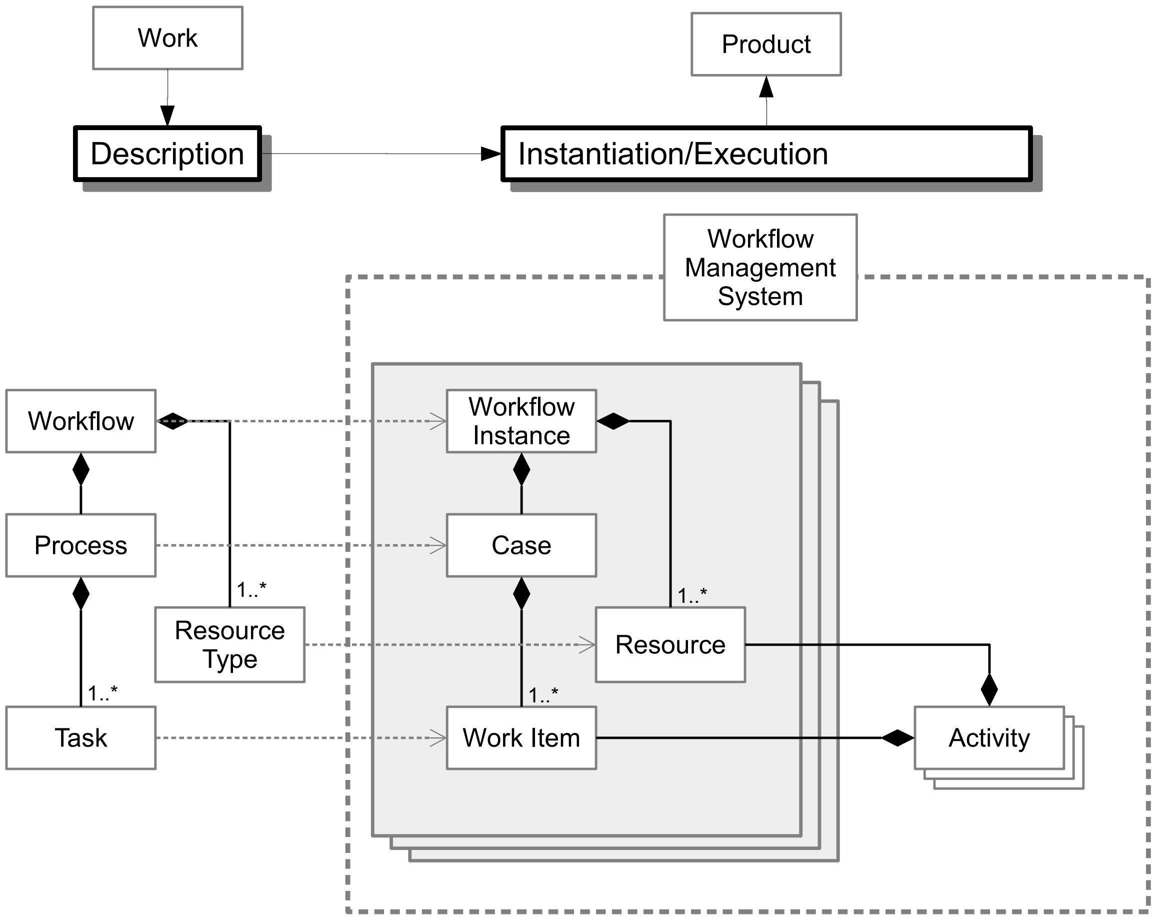


Figure 2.1: Terminology and relationships between terms

Work When using the term *Work* a real world process, sometimes referred to as business process that is to be transformed into a *Workflow* that can be handled (instantiated and executed) by a *Workflow Management System* is meant. The work consists of linked, structured and related *Task* s described as a *Process*. The work revolves around a *thing*, called *Product*, that is produced or modified over time as a *Case* according to a *Process*.

Product The *thing Work* revolves around. The *Product* is not necessarily a physical item such as a boat or house but can be of more abstract form, e.g., an insurance claim, a statistical analysis or an objective, such as the optimization of model parameters.

Workflow A *Workflow* is a means to describe, to orchestrate or to automate

Work. It comprises *Resource Types* and the *Process* describing the *Work* in terms of *Tasks*.

Process A process is used to describe the work to be performed. It consists of tasks, where a task might be a sub-process. It specifies which tasks and sub-processes need to be carried out. Conditions and constraints determine the order of tasks and sub-processes within the process. A process is instantiated as case for execution by a workflow management system. It defines the life-cycle of a case.

Resource Type Resource types specify classes of resources, such as person, machine or group of persons. However, types can be more specific, basically being sub-types of other resource types. For instance, a specific resource type can be *manager* or *administrator*, which in turn are sub-types of *person*. Resource types are associated to tasks of a process by a workflow and specify which resources are allowed or required to be associated with work items of an actual case.

Workflow Instance Before a workflow can be executed, it is instantiated by a workflow management system. The workflow instance comprises a case with work items and actual resources available through the workflow management system, which are associated to resource types defined in the associated workflow.

Case The case is a process instantiation for an actual *Product* that is produced or modified over time according to that *Process*. A case is also known as process instance. The case itself has a limited lifetime and working on a case is discrete in nature. It is represented by an internal state which consists of case attributes, conditions and content. The internal state makes a case unique and makes it possible to refer to a specific case for a given process. A case belongs to a single process while a process can have multiple cases.

Workflow Management System A workflow management system is responsible for defining, creating and managing the execution of workflows using one or more workflow engines. It is responsible for the instantiation of workflows, the management of workflow instances, cases, work items, available resources and activities and the assignment of resources to work items, which will build up an activity. Furthermore, it is responsible for error handling, monitoring and controlling of the execution of a workflow instance.

Task A task is the main building block of a process and is used to structure it. A task describes work as a logical unit which is eventually carried out by a resource as a whole (also referred as an atomic process). However, a task is a generic piece of work within a process but not bound to a specific case. Tasks can be of automatable or of manual nature, which means they can be performed by a computer without human interference or require human intelligence (for judgment call, decision making) respectively.

Work Item Unlike a task which is of generic nature, a work item represents a task within the context of a case. It can be seen as the actual work to be performed for a specific case, hence it is sometimes referred to as task instance. The work item is bound to a specific case.

Resource The resource is an entity that is able to perform a specific task of a process in general. Technically speaking a resource is used to carry out a specific work item. Here, a resource can be a machine, a person, a group of persons or machines (think Grid or Cluster). A resource does not necessarily carry out a work item independently, it is however responsible for the execution of the work item. Resources can be put into resource classes which in return are used in the workflow definition as constraint on what resources can execute what task for a given process. This avoids the necessity of specifying actual resources in workflow definitions.

Activity The activity is the part where all comes together and a task for a case is actually executed by a resource. A work item turns into an activity once a resource is assigned to that work item and the resource starts carrying out the work item.

2.2 Classes of Workflows - Dominant Application Areas

2.2.1 Business Workflows

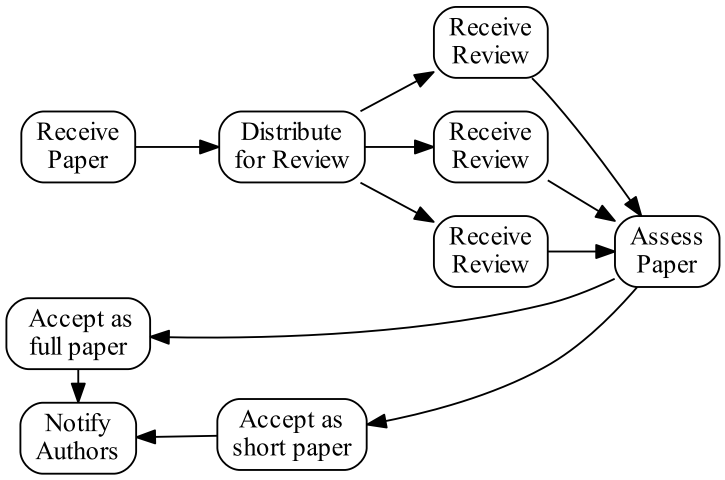


Figure 2.2: Example of a business workflow using control-flow pattern

As an example for a business process Figure 2.2 p. 14 shows a paper reviewing process. The process revolves around a scientific paper, which is submitted by authors, reviewed by multiple reviewers before it is accepted as either full or short paper and eventually authors are notified. This is described via tasks such as *Receive Paper*, *Assess Paper* and *Notify Authors*, to name just a few. However, for simplification reasons, special cases such as rejecting a paper as well as having multiple revisions going back and forth between authors and editor are omitted (see Figure 2.13 p. 36 in Section 2.4.1 p. 35 for this extension).

The purpose of a business workflow is to model business processes by defining a partial order over tasks and to coordinate tasks by automating their execution including scheduling, monitoring and controlling. The goal is to define clear business rules within an organizational hierarchy providing an orchestration of the people (resources) involved in the business process, as well as the efficient execution of the workflow in a heterogeneous technical and organizational environment (Yildiz et al., 2009; Ludäscher et al., 2009). It tries to capture the interaction among different operating human entities (resources) (Migliorini et al., 2011). It contains the automatable parts of the business process (Ludäscher et al., 2009).

The workflow is more or less continuously reengineered, meaning verified, analyzed, optimized and eventually changed according to the optimization and analysis results. In order to re-engineer a workflow, information collected during execution is used and analyzed (mined) to identify which parts of the workflow can be optimized further (Yildiz et al., 2009). Optimization can refer to, e.g., adding or removing tasks from the process and changing the order of tasks.

Many tasks are provided as web-services, making a business workflow typically service oriented. Here an activity invokes a service during execution with a given input which returns the output of the web-service (Ludäscher et al., 2009).

Business workflow systems can handle numerous independent workflow instances. They are complex systems that have to support different types of workflows, involving different resource types and different tasks (Sonntag et al., 2010c). Additionally, workflow systems are part of a larger business organization which interacts with other organizations, where it is important to comply to specific standards, e.g., ISO 9001 (International Organization for Standardization (ISO), 2008; Jahnes and Schüttenhelm, 2008) and where interoperability between software tools, departments and external agencies is crucial. This requires, the use of, e.g., standardized methods according to the desired standard and standardized process definitions, such as BPMN or workflow nets.

2.2.2 Scientific Workflows

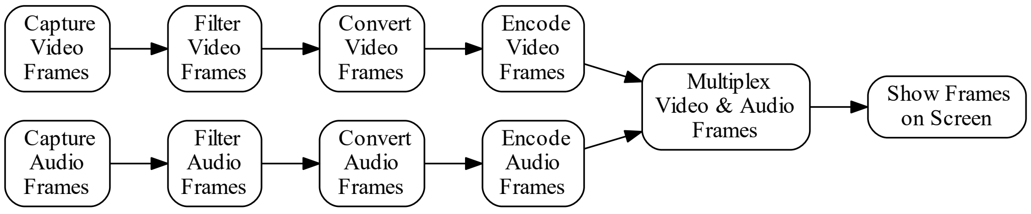


Figure 2.3: Example of a scientific workflow using data-flow pattern

An example of a scientific workflow is shown in Figure 2.3 p. 15. The workflow describes a video recording process, which produces a video stream including audio, which is captured, e.g., by a video recorder. The process starts with capturing video and audio frames, which are then filtered (e.g., sharpened), converted (e.g., scaled) and encoded (e.g., compressed). Encoded video and audio frames will be multiplexed into a single stream, which is eventually shown on a screen.

The purpose of a scientific workflow is to implement an e-Science process and it is used in the field of scientific knowledge discovery (Ludäscher et al., 2009; Yildiz et al., 2009). An e-Science process is characterized as a large-scale, complex, data-centric and computational intensive process, involving the management of large-scale data (Yildiz et al., 2009; Ludäscher et al., 2009; Zinn, 2010; Goble and De Roure, 2009). It requires an environment to chain specialized applications, services and components to solve a computational problem (Migliorini et al., 2011; Zinn, 2010). It is used for the automation and optimization (resource-wise) of error-prone and repetitive tasks, such as: data access, data transformation, analysis and visualization (Wassink et al., 2008; Ludäscher et al., 2009; Migliorini et al., 2011; Zinn, 2010; Goble and De Roure, 2009). They provide systematic and automated means for conducting analyses across datasets and applications. Therefore, saving human *cycles* as well as machine *cycles* (Goble and De Roure, 2009).

A scientific workflow is a precise description of a scientific procedure as a multi-step process to coordinate multiple tasks and resources. Here a task is typically of computational nature (Goble and De Roure, 2009; Romano, 2008).

The process is usually characterized as involving very few interactions with humans. If interactions are required by the workflow it usually involves simple things like providing credentials for, e.g., accessing web-services that need authentication (Weske, 2007; Yildiz et al., 2009; Migliorini et al., 2011).

The nature of scientific workflows is exploratory, because they rapidly evolve over time, e.g., with the introduction of new analysis methods and new hypotheses to test, the workflow is changed to reflect this. Consequently, a scientist typically executes and builds the workflow in a trial and error manner (Sonntag et al., 2010c). This makes it crucial for scientific workflows to be easily modifiable and reusable (Ludäscher et al., 2009).

An established scientific method is, to provide a record of how results were obtained, including methods used, resource information (e.g., machine specifications) and parameters, together with the results itself (Pawlikowski et al., 2002; Kurkowski, 2006; Fomel and Claerbout, 2009). In e-Science, even more information is collected, e.g., workflow activities invoked, services and databases accessed, data sets used, and so forth. Such information is useful for a scientist to interpret their workflow results and for other scientists to establish trust in the experimental result (Barga and Digiampietri, 2008). So capturing provenance information of execution as well as the ability to query such information is very important in scientific workflow systems (Zinn, 2010). It allows for reproducible and repeatable results. Making them more credible and of higher quality (Pawlikowski et al., 2002; Kurkowski et al., 2005; Kurkowski, 2006; Fomel and Claerbout, 2009; Dalle, 2012). Provenance is a very active research field and considering the fast evolving nature of scientific workflow, provenance of the change history of a workflow is also an important field of research and necessary to provide reproducible results and is particularly challenging in a dynamic scientific environment where resources come and go and workflows are modified on the go (Freire et al., 2006; Ludäscher et al., 2009; Anand et al., 2009; Sonntag et al., 2010c).

Requirements for systems for scientific workflow as proposed by (Ludäscher et al., 2006; Gil et al., 2007; Rybacki et al., 2010) or listed in the following.

Seamless access to resources and services in a distributed and/or remote environment, tasks and data sources can be remote and distributed as well (e.g., call to web-service or machine within a network and a database respectively). Access to them and local data sources should be supported by the system out of the box and should be accessible through a unified interface.

Service composition & reuse and workflow design A scientific workflow system should provide means to easily compose different tasks with heterogeneous interfaces and different data formats. To support different data formats data transformation between tasks should be supported directly in the workflow system. At the same time reusing existing compositions should be equally easy.

Smart semantic links The system should assist the design of a process by suggesting compatible tasks, that might possibly fit together (interface-wise). Ideally semantic information of the interfaces that tasks provide is available and automatic match making of compatible tasks or automatic transformation (shims) of data can be provided.

Scalability Scientific workflows can involve a large amount of data or many parallel tasks to execute or both. The workflow system should support such data- and compute-intensive workflows, e.g., by providing interfaces to Grid environments (DataGrid and ComputeGrid respectively)

Detached execution Scientific workflows can be long running by nature. The workflow system should provide an execution mode that runs the workflow execution in background without the need of a client continuously connected to it. Consequently, the engine must be reattachable to a client or monitoring application at any time.

User-interaction Usually user interaction is rare in scientific workflows. They are mainly used to make decisions on alternative branches or provide credentials for specific data sources, e.g., database credentials. However, as workflows get more complex further user interaction might be desired, e.g., incorporating visual analytics techniques to inspect data and provide derived data based on those inspections for further processing. Especially challenging is the integration of the user when dealing with detached executions, as there needs to be a mechanism to notify the user that interaction is required before continuing the workflow execution.

Reliability and fault-tolerance/System stability Computational environments can be unreliable. For instance, a service used by a task can break or be unavailable at some point. So scientific workflow systems should provide means to deal with failures during execution and provide means to compensate, e.g., by providing alternative services in case of service unavailability.

Data provenance Scientific workflows are used to conduct computational experiments. Analogously to a conventional experiment, which should be documented for reproducibility, an experiment conducted using a scientific workflow system should be reproducible as well. The system should capture activities performed, methods and parameters used as well as intermediate results if applicable automatically. It should provide means to generate reports as well as use this data to rerun those experiments, e.g., ideally using the *Smart-Rerun* feature when for instance changing parameters.

Smart reruns Scientific workflows are usually run multiple times, either from the beginning or starting from a specific task, typically involving changing some parameters or input data. The system should be able to determine the influence those changes have when a workflow is rerun. It should reuse as much already computed and generated data and only rerun affected parts of the workflow.

Workflow Roles Rybacki et al. propose additionally to *User-interaction* the support for specific user roles provided as special resource types in order to control who is allowed to perform which user-interaction in a scientific workflow (Rybacki et al., 2010). The idea is to involve experts for specific tasks of the workflow to be able to make decisions, ideally leading to a more founded decision than a non expert would make.

For instance, in a process that involves accreditation of previously created data, results or predictions, such as needed in medical studies, e.g., studies involving selection of appropriate medications. The accrediting entity usually needs to be an independent party authorized and certified for issuing accreditation. In this case restricting the accreditation to be performed by only certified accreditation entities would lead to a viable accreditation.

2.2.3 Control-flow vs Data-flow

Different execution patterns for process definitions exist. Execution patterns determine or restrict the order of tasks in which they can be executed.

The execution model used in the application field of business processes focuses on the execution state of tasks rather than data availability when defining the order of tasks. Which means, that the executability of a task is determined by the finished state of other tasks. For instance, let's take the tasks called *Assess Paper* and *Notify Authors* which are part of the example business process shown in Figure 2.2 p. 14. Executability is defined based on finished tasks. In this case it is defined that *Notify Authors* can only be carried out once *Assess Paper* has finished This type of description is referred to as control-flow based or control-flow oriented. Control-flow establishes a partial order of tasks based on their finished status. The partial order is established on a temporal basis and makes no assumption on data availability. It only implies data availability but does not necessary explicitly specify it (Yildiz et al., 2009; Migliorini et al., 2011).

In a control-flow oriented description concurrency between tasks is modeled explicitly using independent partial orders of tasks (refer to Figure 2.6 p. 27) or by allowing multiple instances of tasks (Yildiz et al., 2009; Migliorini et al., 2011).

In contrast to control-flow based descriptions, the application field of scientific workflows prefers a description based on data being exchanged between tasks. Such descriptions are referred to as data-flow oriented or data-centric descriptions. This means, execution order or executability of tasks is achieved by defining data dependencies between them.

For instance, let's look at the tasks *Capture Video Frames* and *Filter Video Frames* from the scientific workflow example shown in Figure 2.3 p. 15. Obviously there is a dependency between *Filter Video Frames* and *Capture Video Frames*, because without capturing video frames there is nothing *Filter Video Frames* can be applied to.

At first glance this seems describable using control-flow oriented descriptions. Specifying that *Filter Video Frames* can only be executed after *Capture Video Frames* has finished should be sufficient. However, while this works in theory in practice this indicates that the capturing task *Capture Video Frames* has to be finished first, which means the *entire* video is captured before the filter task can be executed. But this might come at a great cost, as all frames have to be temporarily stored between *Capture Video Frames* and *Filter Video Frames*. This is true between all tasks if a control-flow oriented description is used. In the example in Figure 2.3 p. 15 the end product is the display of a video stream on a screen, which means in case the video source provides a fixed number of video and audio frames that all frames will be filtered, converted and encoded before they appear on screen. Additionally, to the memory overhead this introduces, it also delays the output to the screen depending on the number of video and audio frames as well as computational expense of the filter, convert and encode tasks. Going even further, assuming the video source is a live feed (e.g., a surveillance camera) continuously providing video and audio frames, the control-flow oriented approach wouldn't produce an output on the screen because the capturing tasks would never finish.

However, by using a data-flow oriented approach only the data dependency between *Filter Video Frames* and *Capture Video Frames* would be defined. This means *Filter Video Frames* can be executed as soon as *Capture Video Frames* produces data, that captures the first frame. This will eliminate the need to store all captured frames temporarily. Only frames that are not yet filtered need to be stored and can be discarded as soon as *Filter Video Frames* consumes them.

Since tasks only depend on data but not on the finish state of other tasks, data-flow oriented descriptions are concurrently executable by nature. Basically all tasks are executed in parallel (Yildiz et al., 2009; Zinn, 2010; Migliorini et al., 2011).

To achieve this, data depended tasks are connected through unbounded channels, sometimes also referred to as streams or pipes. Data is exchanged between tasks over those channels. Tasks push data onto channels and a dependent task consumes data from them blocking if data is needed and not yet available (see Figure 2.4 p. 22 for a sample partial UML sequence diagram for the process shown in Figure 2.3 p. 15). Data tokens are used to indicate data availability and data boundaries and are not shared between channels and are directly consumed. This isolates tasks and their data, which in return avoids side-effects when accessing and working with it (Yildiz et al., 2009; Migliorini et al., 2011).

However, data-flow makes it harder to express control-flow like behavior, such as selecting between options, which needs to be modeled with data-flow patterns using specialized control tasks which in return clutter the process. This would lead to overly complex models, making it harder to maintain and to understand

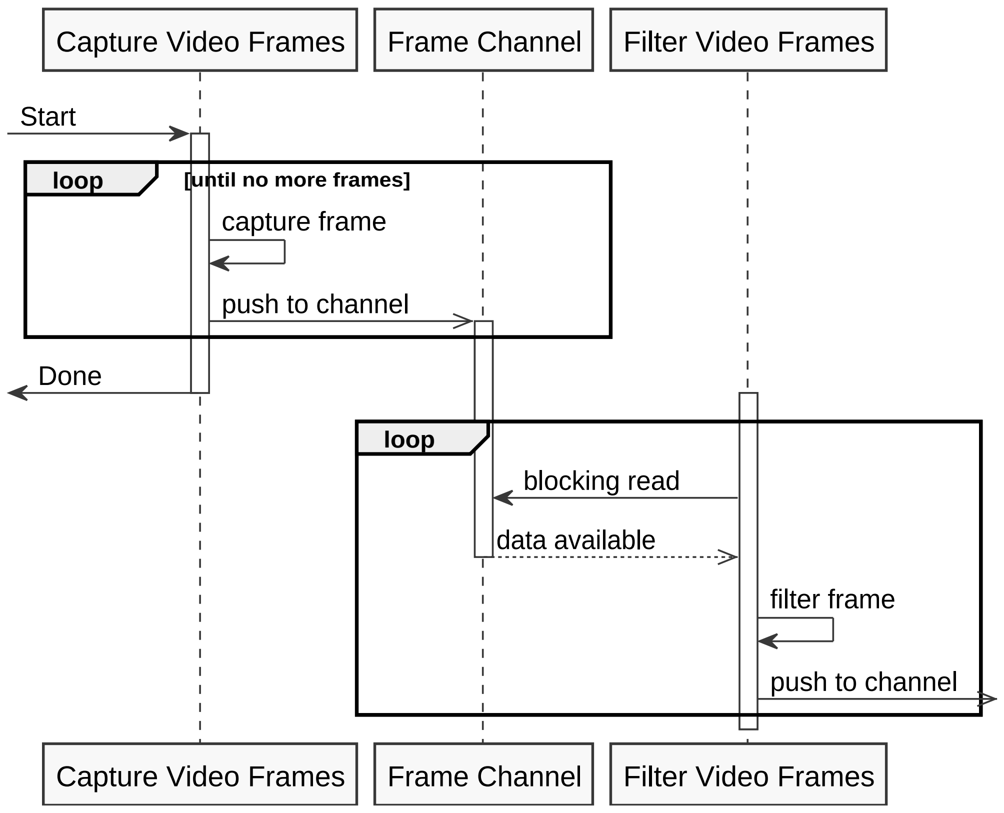


Figure 2.4: A partial sequence diagram for the process shown in Figure 2.3 p. 15. Here only *Capture Video Frames* and *Filter Video Frames* and the connected channel are depicted. The procedure is as follows, both tasks are started (as activity for a case). While there is no data on the channel available the filter activity waits, e.g., by using a blocking read from the channel. The capture activity captures a frame and pushes it asynchronous onto the channel and immediately loops back capturing the next frame, before pushing it again asynchronously onto the channel. Meanwhile, the channel forwards the first frame to the waiting filter activity, which in return filters the frame before asynchronously pushing it onto its output channel and loops back waiting for the next frame on the channel.

them (Bowers et al., 2006; Wassink et al., 2008; Yildiz et al., 2009; Migliorini et al., 2011). However, this is alleviated in some systems by providing explicit control-flow constructs as part of their feature set to reduce this complexity (Migliorini et al., 2011).

2.2.4 Tool Support

	control-flow	data-flow	Business Workflows	Scientific Workflows
YAWL (FlexY)	×		×	
Activiti	×		×	
IBM Business Process Manager	×		×	
Kepler		×		×
Taverna		×		×
Pegasus/DAGMan		×		×
Microsoft Trident	×			×
eBioFlow	×			×
Sedna	×			×
BioFlow		×		×

Table 2.1: Workflow Management Systems, their application field and flow pattern used. Interestingly, Business Workflows use Control-flow, whereas Scientific Workflows utilize both Control-flow and Data-flow pattern, depending on the system at hand.

In order to create, manage and execute workflows a workflow management system is needed. A number of such workflow systems exist. They range from commercial, free to open source systems, each with its own strengths, weaknesses and feature set. Some systems are of general nature trying to support most of the workflows out there, while others specialize in a specific application field or specific type of workflows.

However, it appears that workflow systems are divided into two camps. One camp supports business workflows while the other camp supports scientific oriented workflows. This stamps from the fact, that the requirements for supporting business workflows is quite different from supporting scientific workflows.

In the application field of business workflows a number of business workflow systems exist, e.g., YAWL (FlexY), Activiand IBM Business Process Manager (Russell and ter Hofstede, 2009; Schick et al., 2011; Rademakers, 2012; Dyer et al., 2012). All of which focus on the control-flow between tasks when defining the process. They use different languages for the process definition. Some of those languages are BPMN, XPDL, YAWL and BPEL, with BPMN being the most popular one these days in the domain of business process modeling (Andrews et al., 2003; Shapiro and Marin, 2008; White and Miers, 2008; Russell and ter Hofstede, 2009; Van Gorp and Dijkman, 2011). These languages provide a concise, elegant, and well readable representation of well structured processes. A comparative study of languages for business processes and rules can be found in zur Muehlen and Indulska (2010) and Ko et al. (2009).

In the application field of scientific workflows, different approaches exist. For instance, a special way of defining scientific processes is through scripts. Script-based approaches were used before the term scientific workflow was crafted. They required programming expertise, had no automatic provenance and little to no design support. Furthermore, it was hard to use parallel and distributed resources without extensive programming knowledge let alone to support this on a multitude of different platforms, systems and in heterogeneous environments (Zinn, 2010).

However, there are also scientific workflow systems and languages, such as Kepler, Taverna, Pegasus/DAGMan, Microsoft Trident, Triana, Sedna, BioFlow and eBioFlow, just to name a few (Altintas et al., 2004; Hull et al., 2006; Deelman et al., 2005; Barga et al., 2008b; Taylor et al., 2007; Wassermann et al., 2007; Jamil and El-Hajj-Diab, 2008; Wassink et al., 2008). A comparative overview of scientific workflow tools might be found in Curcin and Ghanem (2008); Deelman et al. (2009) and Talia (2013).

Each of those systems provides its own ecosystem, including modeling language, design suite, software components and execution model. Different platforms have various capabilities and different purposes and they have little compliance with standards. This makes it hard to reuse workflows outside of the platform it was specified for. This is even more amplified when scientists use their own software components which makes workflows not even reusable by others on the same platform (Goble and De Roure, 2009).

Scientific platforms typically use the data-flow paradigm when specifying processes. Microsoft Trident and eBioFlow are scientific workflow systems based on business workflows, employing the control-flow pattern rather then the data-flow pattern when defining processes (Sonntag et al., 2010c). Sedna even uses BPEL for describing scientific processes (Wassermann et al., 2007; Goble and De Roure, 2009).

Scientific workflow systems are developed for long running computational intensive tasks with a huge amount of data (Migliorini et al., 2011). Dealing with this amount of data and computational requirements, scientific workflow systems are developed with Grid or Cluster environment in mind and are usually first-class citizen of those systems, providing ready to use components for data/code shipping, job scheduling, authentication, authorization etc. (Sonntag et al., 2010c; Zinn, 2010; Migliorini et al., 2011).

Scientific workflow systems usually consist of three components (Roure et al., 2008; Goble and De Roure, 2009):

Execution platform executes the workflow including monitoring and optimization of the execution (resource-wise). Additionally, it is responsible for handling data, logging and security. In case of failure it is also responsible for recovering from that failure.

Visual design component is a visual scripting application for authoring and sharing workflows. It shields the author (scientist) from complexities of the used applications. Additionally, it allows for an easier understanding of workflows without specialists and allows scientist to build their own data pipelines.

Development kit which lets you add new functionality and or embed workflows into other applications. When embedded into another application the workflow functions as explicit and reusable specification.

2.3 Description of Workflows

2.3.1 Imperative Description

Processes can be described in different ways. One way is to use imperative descriptions (Pesic et al., 2007).

Imperative descriptions provide information on what to do and how to do it. For instance, to define that *Task 2* has to be executed some time after *Task 1*, it has to be explicitly specified what can happen between the end of *Task 1* and the start of *Task 2*, e.g., all the possible tasks that are allowed to execute in between or in parallel.

Imperative descriptions and their constructs are mainly driven by the underlying execution model that is later used to execute that description. Different execution models in both control-flow and data-flow oriented process descriptions exist.

For control-flow oriented processes typical execution models base on petri nets (Petri, 1966; van der Aalst, 1998; Ludäscher et al., 2009; Migliorini et al., 2011) and its extension color petri nets (Jensen, 1986). A workflow specific variant of colored petri nets, the workflow nets were introduced in Van Der Aalst and Van Hee (2004). High-level petri nets extend a colored petri net by hierarchy and time and are used as basis of a workflow net which additionally adds special routing constructs (explicit AND split/join, explicit OR split/join) to them. Petri net based descriptions usually model tasks as transitions (boxes) and preconditions, such as task finished or resource availability using places (circles). Tokens in places represent fulfilled preconditions. The use of a formalism such as workflow nets, allows for analysis and verification of process definitions, e.g., evaluating soundness or check for dead/live locks. It also allows for optimization of the described process.

In order to imperatively describe a control-flow oriented process, common routing pattern are used to specify task execution order. Tasks can be optional, can be executed in parallel, can be executed after each other or might be executed multiple times. This can be achieved by selective, parallel, sequential and iterative routing respectively. In the following those routing constructs are explained and depicted using the notion of workflow nets as found in Van Der Aalst and Van Hee (2004).

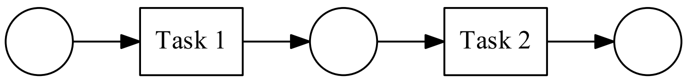


Figure 2.5: Sequential Routing depicted in the notion of a workflow net

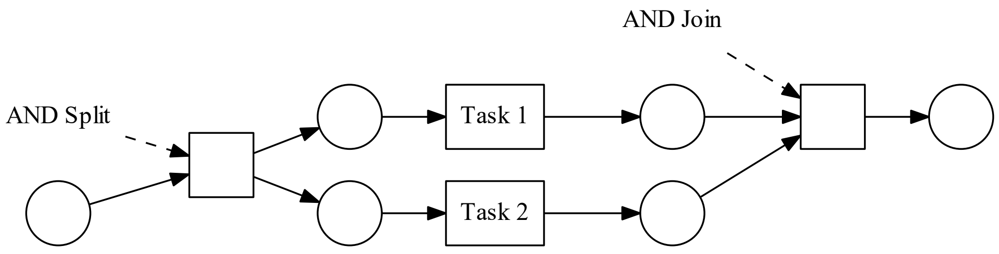


Figure 2.6: Parallel Routing depicted in the notion of a workflow net. Dashed arcs are not part of the workflow net, they simply indicate the type of Join or Split used.

Sequential Routing Sequential routing is used where tasks need to be performed after each other, possibly having some kind of dependency. For instance, in Figure 2.2 p. 14 the tasks *Receive Paper* and *Distribute for Review* are modeled using sequential routing, because distributing a paper for review cannot be carried out if a paper is not received. There is clearly a dependency between both tasks. Modeling a sequential routing in workflow nets is shown in Figure 2.5 p. 27, here *Task 1* is carried out before *Task 2*. A place between *Task 1* and *2* indicates whether *Task 1* is finished (a token is present).

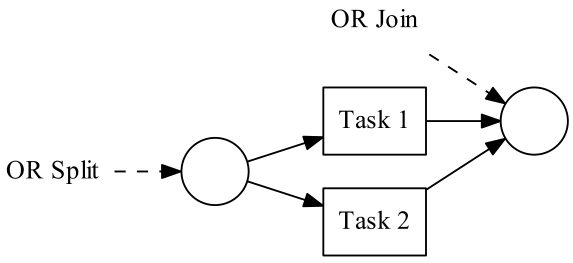


Figure 2.7: Selective Routing depicted in the notion of a workflow net. Dashed arcs are not part of the workflow net, they simply indicate the type of Join or Split used.

Parallel Routing Parallel Routing is used where tasks can be performed concurrently to each other, so having no dependency to each other. For instance, the multiple tasks *Receive Review* in Figure 2.2 p. 14 can be considered to run in parallel as reviews are usually conducted by multiple assets independently from each other and are independently submitted and received. In workflow nets parallel routing as shown in Figure 2.6 p. 27 is introduced by an *AND Split*, which indicates that all outgoing arcs are to be followed. This means the split simultaneously activates the paths with *Task 1* and *Task 2* which enables them to be carried out concurrently. Eventually, parallel routes are joined using an *AND join*. Both *AND split* and *AND join* are modeled using transitions.

Selective Routing In contrast to parallel Routing, selective routing selects a specific route from a multitude of possibilities. For instance, in Figure 2.2 p. 14 only one of the tasks *Accept as full Paper* and *Accept as short paper* should be executed. This can be achieved using selective routing. In workflow nets selective routing can be modeled as shown in Figure 2.7 p. 28. Here an *OR split* is responsible for selecting one route only. Routes are later join using an *OR join*. Both *OR split* and *OR join* are modeled as places.

Iterative Routing A special routing pattern is the iterative routing, as it can al-

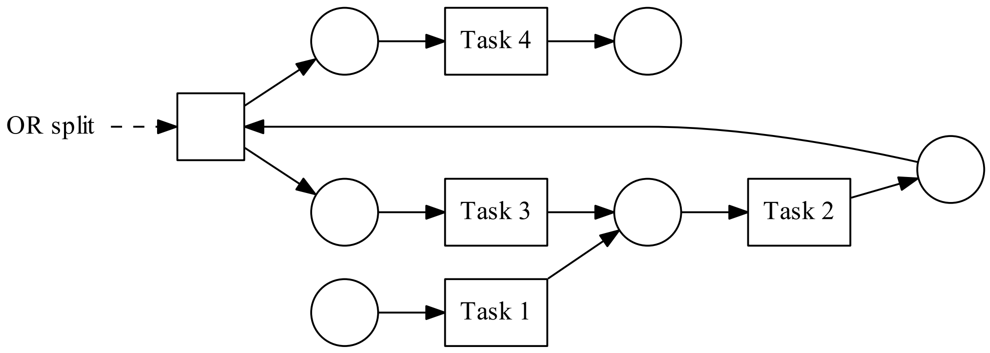


Figure 2.8: Iterative Routing depicted in the notion of a workflow net. Dashed arcs are not part of the workflow net, they simply indicate the type of Join or Split used.

ready be described using selective routing. It is used to specify the repeated execution of one or more tasks. In order to model iteration in a workflow net a dedicated *OR split* transition is used that selects either the iterative route or the iteration break route (see Figure 2.8 p. 29). The *OR split* is an explicit or split and a feature of the workflow net formalism compared to the implicit *OR split* seen in Figure 2.7 p. 28.

Since data-flow oriented or data centric descriptions are concurrent in nature and focus on data dependencies a number of other execution models are used.

Models based on Directed Acyclic Graphs (DAGs) are typically used where the target is the execution on grid environments. DAG based execution models capture serial and task parallel execution workflows, where loops are not supported and each activity is executed only once (Ludäscher et al., 2009; Zinn, 2010). Other execution models that are not restricted by a DAG are based on Kahn's Data-flow Process Networks (KPN) (Gilles, 1974; Lee and Parks, 1995; Migliorini et al., 2011), Synchronous Data-flow (SDF) (Lee and Messerschmitt, 1987) or Collection Oriented Modeling And Design (COMAD) (Zinn, 2010).

Data-flow Process Networks based on Kahn's process networks are popular when modeling data-flow oriented processes. A KPN models a process with tasks and channels between tasks that are unbounded. There can only be one producer per channel and one consumer. Pushing a data token onto a channel is non-blocking while reading a token is, which means a task does not need to wait for its output to be accepted by another task to continue working, while a task can only read one token at a time. A special form of KPN is SDF, which schedules tasks statically. Here for each task it is specified how many data token it generates and how many data token it consumes. That is, SDF based descriptions can always be converted into an implementation that is guaranteed to take finite-time to complete all tasks and use finite memory. For instance, having three tasks A, B, C (see Figure 2.9 p. 30), where A produces 20 tokens in total, while B produces 20 tokens for each 10 token it consumes and C consumes 10 tokens per run. This information allows to determine how often each task has to run to finish. A runs once, B runs twice and C has to run for times until all tokens are consumed. Based on this information and knowledge about data dependencies between A and B as well as B and C a scheduling algorithm can determine a valid execution order, such as $ABBCCCC$ or $ABCCBCC$.

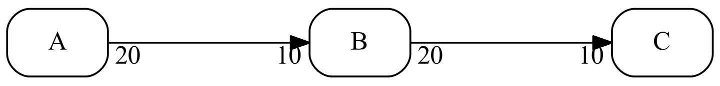


Figure 2.9: Synchronous Dataflow (SDF) example in SDF notation

2.3.2 Declarative Description

Control-flow oriented workflow approaches model workflows (Van Der Aalst and Van Hee, 2004) as a set of tasks that are arranged using various control-flow oriented patterns that permit sequential, parallel, or alternative execution and can be structured using complex tasks that represent sub-processes. Data-flow oriented approaches model workflows as tasks that are put into relation using data dependencies between them. Here the structure is defined by those dependencies.

Both approaches provide concise, elegant and readable representations, e.g., when using modeling languages like BPMN, Workflow Nets or Synchronous Dataflow, for highly structured processes, be it data or control-flow driven. However, they have well-known limitations when expressing workflows with a large amount of flexibility during execution, also referred to as loosely structured processes.

Such processes usually involve many non-dependent tasks which can usually take place in any order with only some tasks depending on data of or the execution of another task. Often such processes involve multiple unordered iterations of tasks. For instance, an example loosely structured process is shown in Pesic et al. (2007) and depicted in Figure 2.10 p. 31.

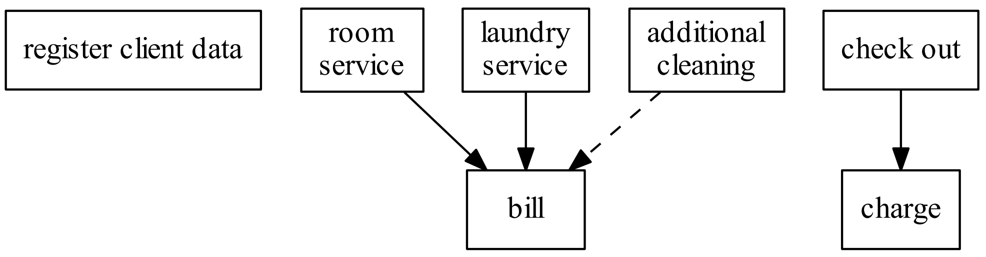


Figure 2.10: Declarative description of a hotel room renting process. It consists of seven tasks with constraints between them. Tasks are depicted as boxes and arcs are constraints symbolizing that the arcs target task needs to be executed some time after the arcs source task has been executed. Constraints are defined for tasks involving hotel services (room service, laundry and cleaning), which need to be billed eventually. Also, after checking out the hotel room occupant must be charged the billed services and room rental.

It models a hotel room renting process, consisting of seven tasks:

register client data which needs to be executed once when the client or guest arrives at the hotel and collects client name and address as well as payment information

bill which is used to add items to the overall bill of the client

room service registers room service for the client

laundry service registers laundry service for the client

additional cleaning registers additional cleaning on top of regular cleaning in special cases

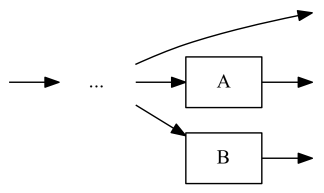
check-out client checks out of the hotel

charge client is charged with the accumulated bill

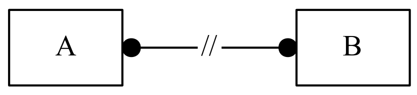
The tasks are loosely coupled as they can appear in arbitrary order and arbitrary often, except for the tasks *register client data*, *check-out* and *charge*. Also there are dependencies between tasks, for instance *room* and *laundry service* need to be put on the bill after execution, as well as the client needs to be *charge*d after *check-out*. In order to describe such a process using imperative methods all possible scenarios would have to be modeled explicitly and would lead to over-specification of the process making modeling, reading and maintaining the process hard. Using declarative descriptions, e.g., using constraints to describe task dependencies and order, allows for an easy modeling of such processes (Pesic et al., 2007). Constraint based approaches describe relations between tasks using rules and constraints. For example, if there are two tasks A and B and A cannot be executed if B is executed and vice-versa, an imperative description might look like Figure 2.11(a) p. 33, however this over-specifies the process. A declarative approach just defines a constraint stating A and B can never be executed both (see Figure 2.11(b) p. 33).

Another loosely structured process is presented in Jablonski (2010) and shown in Figure 2.12 p. 34. Here, an agenda is to be written by an assistant which in turn is reviewed by a manager. The agenda is used to write a letter, again by an assistant, in order to invite people and publish the agenda. Before the letter can be sent out however, it needs to be reviewed and signed by the manager.

At first glance this scenario seems quite structured and Figure 2.12(a) p. 34 shows a straight forward imperative description of the process. However, in reality this description puts too many restrictions on the process. For instance,



(a) imperative description of exclusive A or B (every possible path can only have A, B or no A and no B)

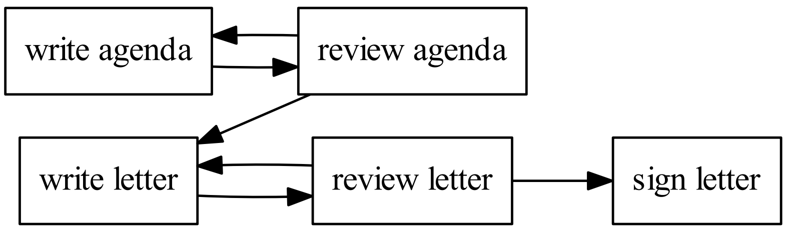


(b) declarative description of exclusive A or B (notation based on DECLARE)

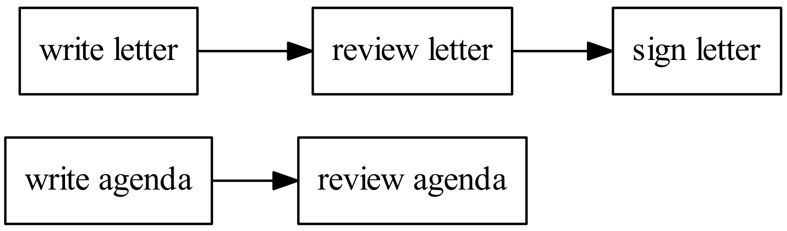
Figure 2.11: Imperative vs. Declarative description of a process where A and B cannot be executed both

what if during reviewing the letter, the manager decides that the agenda needs to be changed. This is not possible with the shown model. Of course an additional connection between *review letter* and *write agenda* can be inserted to model this edge case. However, what if this happens during *writing letter* or *sign letter*, all possible edge cases need to be explicitly modeled which for this example, leads already to a complex over-specified process description. Using a declarative approach similar to the one shown in Figure 2.11(b) p. 33, simply specifying constrains, such as *the agenda needs to be reviewed, after writing it*, *the letter needs to be reviewed, after writing it* and *the letter needs to be signed, after reviewing it*, will be sufficient to include the edge case presented while still maintaining a straight forward easy to read description (see Figure 2.12(b) p. 34).

Several declarative workflow approaches exist (Pesic et al., 2007; Dourish et al., 1996; Mangan and Sadiq, 2002; Cohn and Hull, 2009). Besides the already mentioned DECLARE, which uses constraints based on linear temporal logic (LTL) on tasks (Pesic et al., 2007), another approach is to use descriptions based on artifacts (Fritz et al., 2009; Cohn and Hull, 2009; Eckermann and Weidlich, 2011). In artifact centric process constraints are derived from business relevant *artifacts*, e.g., documents or manufactured items that are created, evolved and archived as they pass through a business. An artifact has an inherent life-cycle. They combine both, data and process aspects and serve as building blocks for processes.



(a) imperative description of the letter/agenda writing and reviewing process



(b) declarative description of the letter/agenda writing and reviewing process with constraints depicted as arcs implying a must execute some time after constraint.

Figure 2.12: Secretary process example not very well suited for imperative process descriptions

Activities cause progress in that life-cycle of one or several artifacts. In this sense, the life-cycle of an artifact imposes constraints on the execution order of tasks. Consequently, an artifact centric process is mainly controlled by the artifacts.

2.3.3 Tool Support

A reasoning (or planning) component in a process engine derives execution orders of tasks from the artifacts.

	imperative	declarative
YAWL (FlexY)	×	
Activiti	×	
IBM Business Process Manager	×	
Kepler	×	
Taverna	×	
Pegasus/DAGMan	×	
Microsoft Trident	×	
eBioFlow	×	
Sedna	×	
DECLARE		×
BioFlow		×

Table 2.2: Workflow Management Systems and their type of description.

2.4 Flexible Workflows

2.4.1 Structural Flexibility

Imperative task-based process descriptions provide a concise, elegant, and well readable representation of well structured routine procedures but have well-known limitations when processes with a large amount of flexibility need to be expressed.

There are attempts to provide adaptation capabilities into task-based business workflow models such as the ADEPT approach (Reichert and Dadam, 1998; Dadam and Reichert, 2009; Dadam et al., 2010) but they mainly focus on the adaptation of the process definition during execution. They support punctual modifications such as insertion and deletion of tasks rather than major rearrangements in the control logic. For instance, taking the example process shown in Figure 2.2 p. 14, during the processing of a paper there is a decision to support a multi-step reviewing process, e.g., requesting alterations of a paper according to reviews from the authors before acceptance of the paper. Figure 2.13 p. 36 shows the process while the paper is currently under review and the changed elements of the process (outlined with dashed lines). Here the process is extended by inserting additional tasks. However, there are limitations and requirements in order not to compromise integrity of the process. For instance, it might not be allowed to remove a currently running activity, as well as inserting tasks arbitrarily into the process definition may cause a dead or live lock.

Other systems such as case management systems (Reijers et al., 2003; van der Aalst et al., 2005) try to soften the imperative structure of task-based workflows systems by allowing the re-execution of already finished activities or skipping of unnecessary ones. However, when re-executing an activity all subsequent work items need to be re-executed as well.

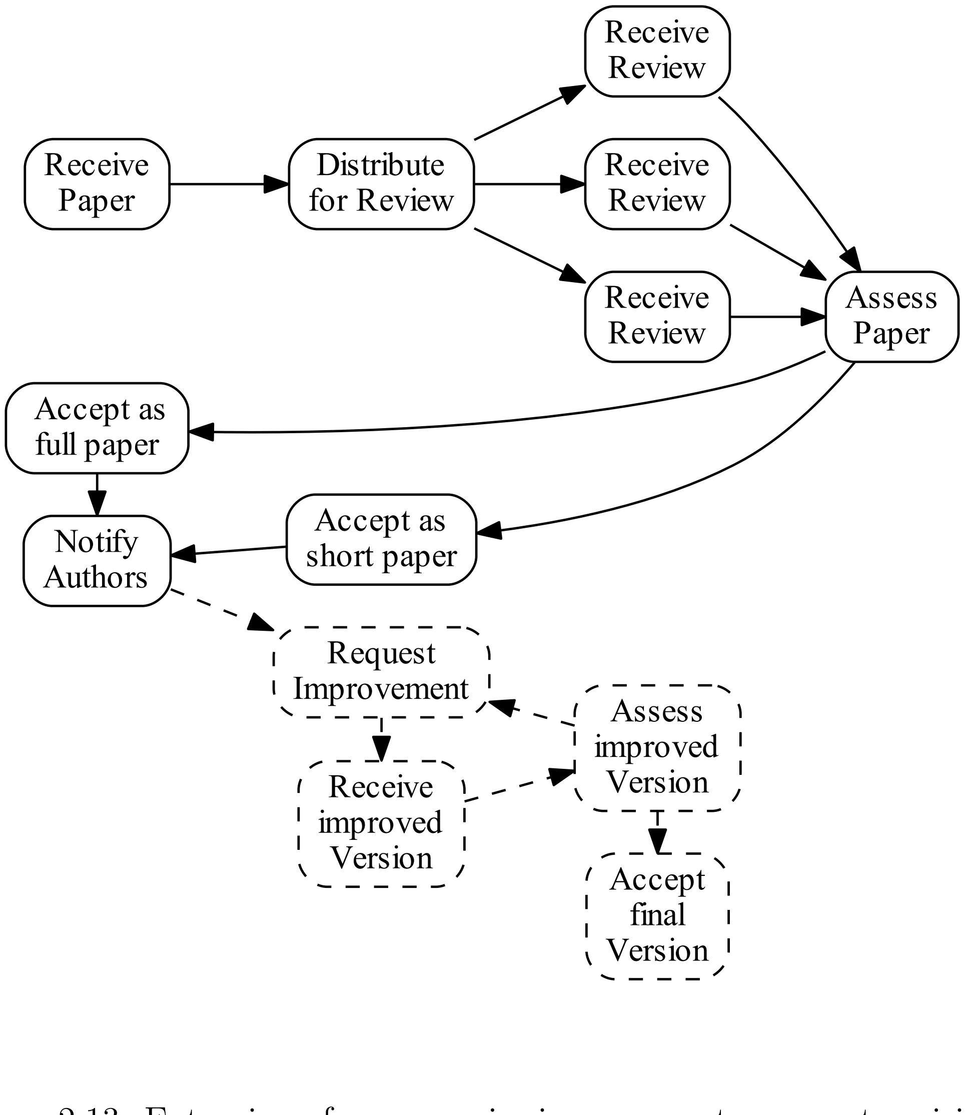


Figure 2.13: Extension of paper reviewing process to support revisions

Declarative process descriptions, be them constraint- or artifact-based also allow for structural changes to the process during execution. For constraint-based descriptions such as the ones of DECLARE this means, tasks can be added or removed and constraints can be added, removed or changed. Here DECLARE will perform analyses on new, changed or deleted constraints and execution history in order ensure integrity and consistency of the process (Pesic et al., 2007). For artifact-based descriptions, tasks can be added and removed and artifacts can be added, removed and changed (Cohn and Hull, 2009).

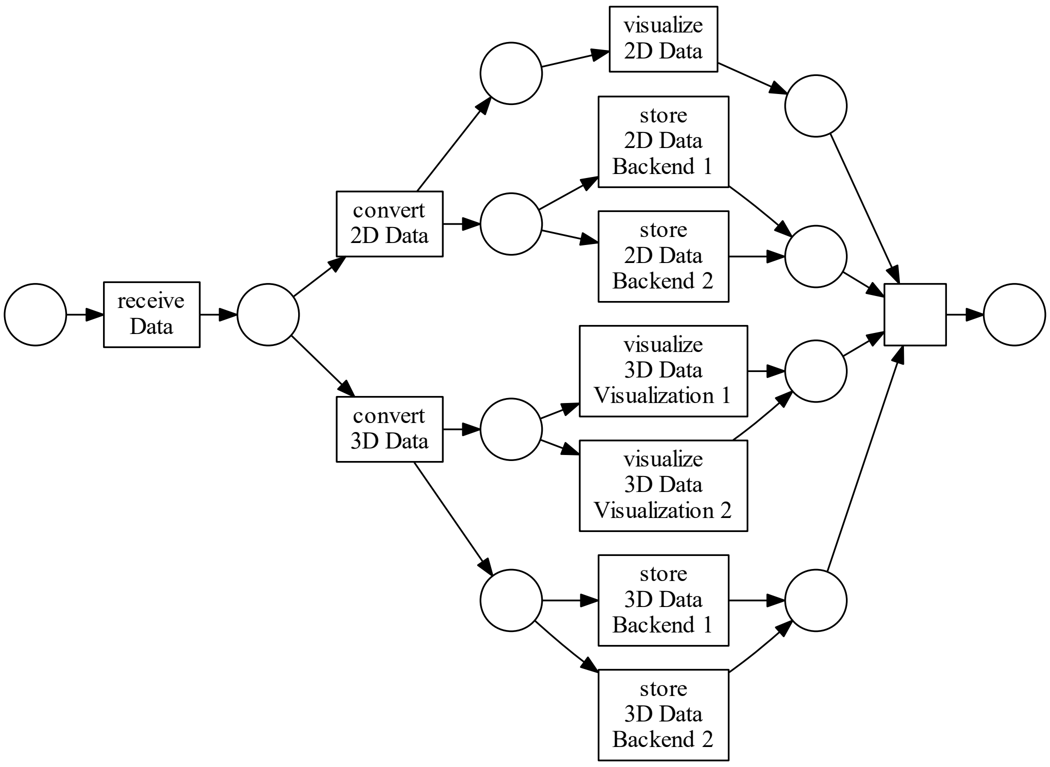
2.4.2 Flexibility in Execution

2.4.2.1 Least Commitment by Templates

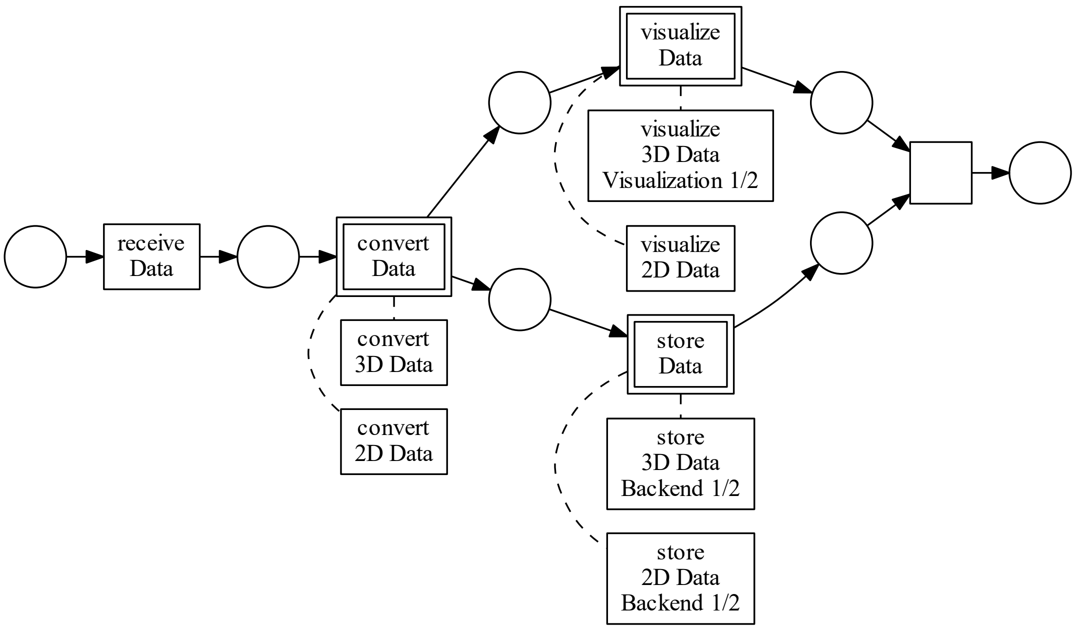
Additional to structural changes to processes, providing flexibility to the execution of a process, the least commitment approach can also be used (Weld, 1994). In least commitment or partial-order planing approaches the ordering of tasks is left open as long as possible and is determined iteratively during runtime. However, in the case of using templates, the planning does not affect the order of tasks but rather the exchange of template tasks with actual tasks. A process is defined by tasks and *template* tasks, which are evaluated and exchanged by actual tasks during runtime. Exchange can happen based on execution history of the process until the template task is reached, as well as based on initial configuration, which might define the actual task to be used for a specific template task, or it can also be selected by a user if there are multiple matching tasks for a template task.

For data-flow and control-flow oriented processes, the usage of template tasks provides adaptation capabilities by being able to exchange template tasks during execution by actual tasks (Ngu et al., 2008; Rybacki et al., 2012b).

However, since templates do not change the structure of the process flexibility is limited. Template task based flexibility is typically used in scenarios where there are a multitude of options which might change over time for a specific task in a process which otherwise would have to be modeled explicitly and changed with each option available or added. For instance, in Figure 2.14(a) p. 38 a process that can convert data of different type (2D, 3D), visualizes it and eventually stores it is depicted using a workflow net as description.



(a) normal description



(b) template-based description

Figure 2.14: Sample workflow illustrating Least Commitment using Templates

Here conversion of the supported data types is modeled explicitly using *convert 2D Data* and *convert 3D Data* tasks that are selected based on the source data type. The visualization also models a data path explicit for 2D and 3D data, where for 3D data there are two different visualizations modeled. Similarly, for the storing 2D and 3D data, different storage back ends are available and explicitly modeled for each data type. In order to support additional visualizations, data types or storage backends the process needs to be extended to integrate those.

Using the template based approach the shown process can be simplified as shown in Figure 2.14(b) p. 38. Template tasks are depicted using a double lined border and tasks that can be used in a template task are connected with a dashed line. Here a template task is introduced for *convert Data, visualize Data* and *store Data*, each of which has actual tasks attached to choose from, *convert 2D/3D Data*, *visualize 2D/3D Data Visualization 1/2* and *store 2D/3D Data Backend 1/2* respectively. Which task is used for conversion and visualization is determined during runtime based on data type that is received (2D or 3D) as well as by the user in case of 3D data, because two different visualizations for this data type exists. The storage task can also be determined by the data type plus user preference on the storage backend.

2.4.2.2 Least Commitment by Declarative Descriptions

Declarative process descriptions provide an intrinsic support for flexibility during the execution of a process based on the least commitment paradigm. By using constraints or artifacts a partial order of tasks is defined. This order might change during runtime depending on defined rules or constraints and is typical addressed with *least commitment* or *partial-ordering planning* approaches (Weld, 1994).

For instance, given the process shown in Figure 2.12(a) p. 34, which defines five tasks, *write letter*, *review letter*, *sign letter*, *write agenda* and *review agenda.* The process defines a partial order between the first three tasks and the latter two. In order to execute that process a partial-ordering approach would initial allow the tasks *write letter* and *write agenda* for execution, reevaluating which tasks can be executed next after each executed task ensuring given order constraints. Assuming *write letter* was executed then after replanning the *write agenda*, *write* *letter* and *review letter* tasks are executable next and so on.

Using a planning based approach that continuously evaluates the order and executability of tasks allows for adding and removing of tasks, constraints or artifacts, providing structural flexibility and adaptability. However, similar to the imperative description based process descriptions supporting structural flexibility, additional consistency and integrity checks or analyses need to be performed in order to ensure a concise process model, meaning e.g., constraints that are modified or added are not violated by prior actions or previously executed tasks in that very case. Such a violation easily occurs, e.g., when there are two ordered tasks such as *write letter* and *review letter* and both are already executed in that order and a new task for instance *spell check letter* is added with the constraint that it has to be executed in between the previously mentioned tasks.

2.5 Summary

Workflows are used to describe work or a procedure involving different tasks that are carried out by resources. The work is described in a process using tasks, sub-processes and information on how those tasks and sub-processes relate to and depend on each other. Workflow management systems are used to execute, monitor, design and analyze workflows. A process turns into a case when it is instantiated for execution by workflow management system, where all the tasks associated with that process turn into work items. A work item itself is the task associated with a specific case and turns into an activity once a resource is assigned responsible for execution.

There are different application domains of workflows resulting in different requirements to workflow management systems and process descriptions.

Workflows were introduced in the business world to describe tasks to be executed by a human entity (resource) throughout an organization and their interrelation, derived from office automation systems. Those workflows were used to control and monitor human work and to analyze human performance. Later, workflow management systems emerged helping to automate and orchestrate workflow between the humans involved even further, incorporating computationally automated tasks such as the invocation of web-services. Workflows involving humans, which orchestrate tasks within organizational structures are called business workflows. Business workflows describe the work as a process built of tasks and their partial order using control-flow constructs, such as sequential, parallel, iterative and selective flow.

Typically, activities are invoked with input data, executed and output data is generated before the next activity is invoked. An activity's action is typically implemented as a web-service or human interaction.

The other application domain involves scientific computational tasks also called computational experiments. Here, scientists conduct computations, such as transformations, analytics and visualization, based on data leading to new data in return. Computations are used to gain insights and knowledge in data, which are used to base decisions on, such as to invalidate or verify a hypothesis. This computation based processes are generally present in the domain of e-Science and can be computational and data intensive, repetitive and error-prone. Automation of such tasks is a desired goal. First attempts used scripts to automate calculations, transformation and so on. While this worked, it usually requires programming skills, does not allow for automated documentation, error handling and recovery as well as using distributed and parallel resources is hard. Therefore, scientific workflows were introduced to cope with these special kind of scientific work, alleviating some shortcomings of a script-based approach. For instance, scientific workflow systems typically provide means for automating the documentation of the workflow execution, including which steps taken, what input data and parameter used as well as which resources and methods were employed as well as automated error handling and recovery. Further, they provide automated optimization of execution, e.g., distributing the execution of activities to free resources and automatically making use of parallel resources such as multi-core architecture and grid or cluster environments.

Comparing business and scientific processes, an apparent observation is the different focus on how work is described using tasks and sub-processes. Most business workflows focus on the description of tasks and their interrelation using control-flow constructs, such as sequential, parallel, iterative and selective flow patterns. Scientific workflows on the other hand typically use a data-flow oriented description of tasks and their interrelation. Control-flow describes a partial temporal order of tasks based on already finished tasks, leaving the data perspective, e.g., the data that flows between tasks, typically as a secondary issue. Data-flow describes the order of tasks based on data dependencies between tasks while not implying a specific order of tasks. For instance, given the tasks $Task_1$ and $Task_2$ and a dependency between them $Task_1 \rightarrow Task_2$.

In business workflows this dependency means $Task_2$ is directly executed after $Task_1$ has finished. In scientific workflows this dependency means $Task_2$ depends on data from $Task_1$ at some point in time. This allows the execution of $Task_1$ and $Task_2$ concurrently with an established data channel from $Task_1$ to $Task_2$.

With some exceptions (Trident uses control-flow, DAGMan uses a directed acyclic graph based task oriented description), scientific workflow systems treat all tasks as concurrently executable, exchanging data through channels between them. In contrast, in business workflows concurrent execution of tasks has to be explicitly modeled as parallel flow or through multiple instances of the same task during runtime (tasks that are invoked multiple times at the same time, e.g., if there is multiple input data).

Workflow systems use different execution models. Business workflow systems typically employ a petri net based execution model. Scientific workflow systems, execution models such as directed acyclic graphs, Kahn's process model, Synchronous Dataflow and even business workflow oriented computation models are used. This typically restricts scientific workflows in describing control flow, e.g., DAG-based systems do not allow loops which is needed for the iterative control-flow pattern.

Another difference between business and scientific workflows is the involvement of humans in the completion of a workflow execution. Humans are an integral resource in the completion of a business workflow. The workflow consists of a process with automatable tasks and usually a number of tasks only executable by a human resource, e.g., evaluation of an application for a loan. In scientific workflows the involvement of humans is limited and typically only requires human interaction when a task, e.g., needs credential information for authentications or when selecting a analysis method. The remaining tasks are completely automatable in scientific workflows.

When designing workflows, business and scientific workflow systems provide visual designers aiding the design process. In the domain of business workflows standards, such as BPMN and BPEL exist to describe workflows and allow for reusing of workflows designed in one system, or switching from one system to another. Scientific workflow systems provide their own languages and tools for describing and designing workflows. This is partly because there are no established standards and because different computational models are employed for the execution of their workflows, e.g., DAG, KPN, SDF or Petri Nets.

Classical workflow systems, business as well as scientific ones, are rigid in terms of workflow execution as well as workflow definition. Adaptive workflows that allow changing of a workflow or the involved process during execution and flexibility on task order and execution, particularly for business workflows is scarcely supported in workflow systems. The ability to adapt to changes during execution, means adding a task, removing a task or moving a task within the process while it is running, e.g., to add another analysis task to the process of a scientific workflow or an additional verification task to a loan approval business process.

Flexibility issues also arise when designing or executing workflows. In traditional workflow systems the workflow is rigid and follows an imperative description of the work as process which implies that each possible task order needs to be explicitly described. This can lead to over-specified workflow descriptions, while specific patterns are still very hard to express, e.g., the exclusive execution of task A or B (if A was executed B can never be executed, and vice versa as well as A and B are not executed at all) is difficult to express imperatively.

However, systems exist that try to address these issues. Some tools, e.g., ADEPT and FlexY enable imperative workflows to be adaptive. Changes of workflows during execution are allowing by insertion, exchange or removal of tasks. In the scientific workflow domain, tools such as Kepler provide flexibility by adding templates to their workflows allowing exchange of tasks during runtime.

Other approaches that allow for flexible design and execution of workflows without over-specifying processes exist. There exist extensions for classical business workflow systems, e.g., case-handling systems that allow skipping of unnecessary tasks as well as the reexecution of already completed tasks, rewinding the workflow if you will. However, when reexecuting an already completed task the reexecution of all subsequent tasks is necessary in those systems. Other approaches choose to describe workflows declaratively rather than using imperative descriptions. For instance, systems such as DECLARE, establish a partial order and dependency of tasks by defining constraints using linear temporal logic (LTL) over them. This allows for an easy description of a workflow where task order and task execution count is only important for a small subset of tasks. This is usually the case when human entities are heavily involved and the process to model is loosely structured. However, one downside of this approach is that the workflows are hardly automatable because of the freedom of choice of possible tasks to execute. Another downside is that in highly structured processes where task order is important this can lead to numerous constraints which makes the workflows harder to understand and maintain.

An artifact-based workflow is built using artifacts, which consists of a data model and a life-cycle model. Additionally, tasks are defined that are orchestrated by constraints or rules on an artifact's data and its life-cycle state. The artifact changes between its life-cycle model states during workflow execution, storing data from associated executed tasks in its data model.

3 Life-Cycle Models and Workflow Approaches in Modeling & Simulation

Chairdrobe: (n) piling cloth on a chair in place of a closet or dresser; see also floordrobe

Daniel Dalton

3.1 Quality

Quality is a frequently used term and is considered to be a multi-faceted term (Garvin, 1984). Garvin distinguishes between transcendence, product based, user based, manufacturing based, and the value based views on quality.

Transcendent The transcendent view assumes that quality cannot be defined or measured precisely but merely be recognized through experience and expertise. This means, that only domain experts (someone with experience and expertise in a specific field) can decide on the quality of a product in that domain. In the domain of Modeling & Simulation a transcendent assessment of quality can be made, e.g., for a simulation model using its abstraction level because it is typically judged by domain experts.

Product Based The product based view implies that quality can be measured. The measurement is based on the quantity or quality of a desired ingredient of the product, e.g., in Modeling & Simulation the use of verification and validation as ingredient during model development can lead to a higher quality model. For instance, Balci describes in more detail a method to measure the quality of models (Balci, 2004).

User Based The user based view refers to a subjective definition of quality. Different users might have a different impression about the quality of a selection of products and make different choices among these. Most users regard the product that meets their preferences best as the product with the highest quality.

In Modeling & Simulation this can refer to simulations or simulation tools that provide visualizations and animations alongside simulations to have a higher subjective quality.

Manufacturing Based The manufacturing based view defines quality by how well a product creation process followed a specific specification (one that is regarded as high quality specification). This means that a product created with a well-defined process (according to a specification), which is well-documented, is of higher quality. This notion of quality has become an industry and certification standard (reflected in the ISO 9001 norm (International Organization for Standardization (ISO), 2008; Jahnes and Schüttenhelm, 2008)).

In the field of Modeling & Simulation such well-defined processes or life-cycle models exist for conducting studies (Balci, 2003; Brade, 2004; Law and Kelton, 2007; Lehmann, 2008; Rabe et al., 2009).

Value Based The value based view defines quality as the performance for an acceptable price. The ratio between performance and price is the main driving force for value based quality. So the quality of a product is always in relation to the cost it took to produce, also known as *affordable excellence*. In the domain of Modeling & Simulation a value based quality definition can be applied to models on different levels of abstraction. While in theory the quality of a higher resolution model is higher the cost of computation in terms of time or resources might render a lower resolution model of higher quality based on the cost performance ration.

In Modeling & Simulation all the aforementioned views on quality can be applied to judge the quality of a product (Himmelspach and Uhrmacher, 2009). This includes the process of developing Modeling & Simulation software, the conducting of a simulation study as well as the creation of a simulation model.

In order to provide products, such as simulation results or simulation models of high quality, a precondition is a well defined process for creating that product. With such a process the creation of the product is reproducible and transparent and it allows pinning down production errors or error sources that relate to the process at hand. If a simulation study, experiment, the creation of a simulation model or even the creation of a simulation algorithm is described as and supported by a process high efficiency and repeatability can be achieved.

By focusing on the manufacturing based view of quality, the process that is used to create a product is taken, as indication of a product's quality. This can be combined with the product based view, making a product also dependent on intermediate product quality and quantity. By reviewing life-cycle models and workflow approaches used in the field of modeling and simulation common structures and requirements for high quality Modeling & Simulation products are derived.

3.2 Modeling & Simulation Products

The aforementioned quality views always refer to a product. In Modeling & Simulation, products comprise results of intermediate steps that are created by a modeler, a simulation performer or other people involved.

Products in Modeling & Simulation can refer to:

Modeling & Simulation Software The Modeling & Simulation software created is a product which is used, e.g., to conduct simulation experiments, to support the creation of a simulation model or to analyze simulation results. In order to provide credible results, the software needs to meet desired quality standards, e.g., by following a software development process according to a specification.

Quality in software development is addressed by using life-cycle models for software development and by employing techniques to improve the quality (i.e., validation, verification, and code analysis). Albeit those techniques and life-cycle models exists they have to be used in order to work. However, this is not always the case as shown in Merali (2010), stating that the product quality of Modeling & Simulation software could be better. Another way to improve the software development process in Modeling & Simulation is to exploit reuse (Himmelspach, 2012). If software was created with reuse in mind, separation of concerns will inherently take place, meaning that software will be composed of smaller elements (Himmelspach, 2009).

The Problem Definition The problem definition is the question about a system that should be answered. It needs to be well formulated because it impacts subsequent products such as the models, the experiment and the analysis. Quality of the problem definition directly impacts the quality of further products (product based view of quality).

The Model In Modeling & Simulation there exist different types of models, herein the focus lies on simulation models. Those typically comprise conceptual model, formal model and implemented/executable model.

The quality of each of these models plays an important role because subsequent products, e.g., the experiment (simulation) are based on them.

The Experiment The experiment conducts simulations including replications if needed for one or more configurations (e.g., parameter scan, optimization). Simulation results will pass over to analysis and interpretation. Well defined and documented when executed experiments are mandatory for achieving credible and reliable results. An experiment is designed to answer the question of the problem definition. Albeit, it should produce a reliable answer efficiently, meaning with as few numbers of obsolete computations as possible.

The Analysis (and Interpretation) After simulation of a model an analysis of the results, considering the initial problem definition is usually performed. This typically implies the application of a number of mathematical (statistical) methods on the results produced and the interpretation of them using those methods.

The Process of the Modeling & Simulation Study All the aforementioned products can be created during the performance of a Modeling & Simulation simulation study. They depend on each other and have at least a partial order (i.e., an analysis cannot be performed before data, e.g., simulation results, is available, which is not available without a simulation model and so on). So the generation of those products follow some kind of process, which itself can be considered a Modeling & Simulation product itself.

3.3 Life-Cycle Models

3.3.1 Life-Cycle Models for Modeling & Simulation Software development

Life-Cycle models are a common means in software engineering to define the procedure of developing software. Models such as the waterfall model (Royce, 1970; Bell and Thayer, 1976), which describes a sequential procedure of tasks such as *Conception*, *Design*, *Implementation*, *Verification* and *Maintenance*, have been around for decades.

However, alternative models still emerge, e.g., models that follow the paradigm of agile software development, which focus on early delivery and continuous improvement, such as Scrum, as one of the recent approaches (Sims and Johnson, 2011).

Additionally, standards or norms such as the ISO 9000-3 (Hoyle, 2009), that are dedicated to software development process quality (Sommerville, 2007) guide developers through their software development process (waterfall, scrum, etc.) and help with the identification of work done.

This can be extended by detailed descriptions, e.g., test methods that must be performed and can help to increase the *quality* of the software developed.

Such software development models if used for creating Modeling & Simulation software can help to improve the quality of Modeling & Simulation products (Sommerville, 2007).

When performing a modeling and simulation study it is often needed to also perform a software development. For instance, this might be the case when a conceptual model needs to be transformed into an executable model, interpretable by computers. This transformation can either be done by using a domain specific language (modeling language, e.g., ML-Rules (Maus et al., 2011), by a simulation language, such as CSSL (Nilsen and Karplus, 1974)) or by using a general programming language. Independently from the transformation approach in use, intrinsically a software development process is assumed. This can either be the software process that is used to create the domain specific language and components that interpret and execute it or the process involving a general purpose programming language when implementing the executable model. The aforementioned software development processes can be employed for those as well to ensure *quality* products.

3.3.2 Life-Cycle Models for Modeling & Simulation studies

Similar to the process of software development, the process of conducting a simulation study has a multitude of proposed life-cycle models available. Comparable to software development those life-cycle models try to guide users through a study.

Among those life-cycle models are those proposed by Balci, Brade, Law, Sargent, Lehmann, Sawyer and Rabe (Balci, 2003; Brade, 2004; Law and Kelton, 2007; Sargent, 2008; Lehmann, 2008; Sawyer and Brann, 2008; Rabe et al., 2009).

The classical life-cycle models by Balci, Sargent, Law and Rabe are depicted in Figures 3.1 p. 51, 3.2 p. 52, 3.3 p. 53,3.4 p. 54 and 3.5 p. 55 respectively.

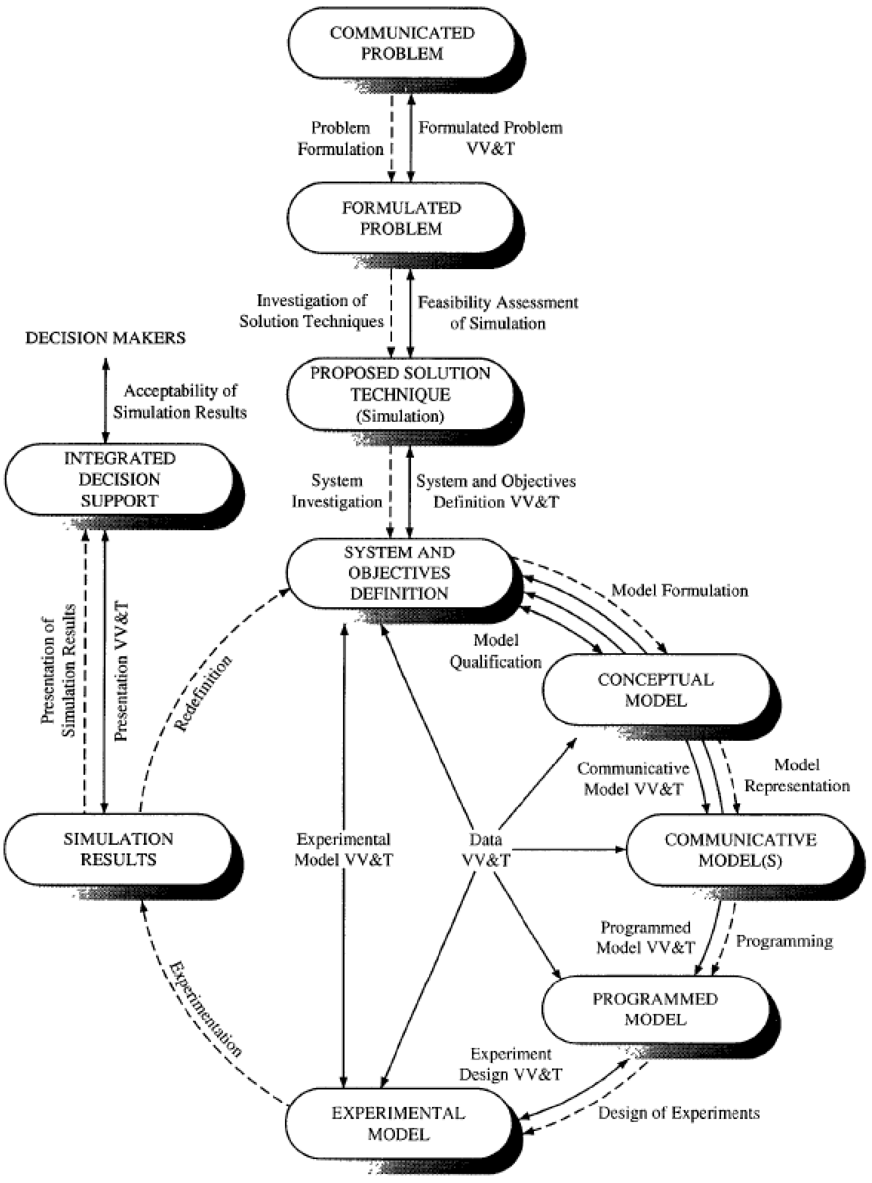


Figure 1: The Life Cycle of a Simulation Study

Figure 3.1: Life-cycle Model for Modeling proposed by Balci (taken from (Balci, 2003))

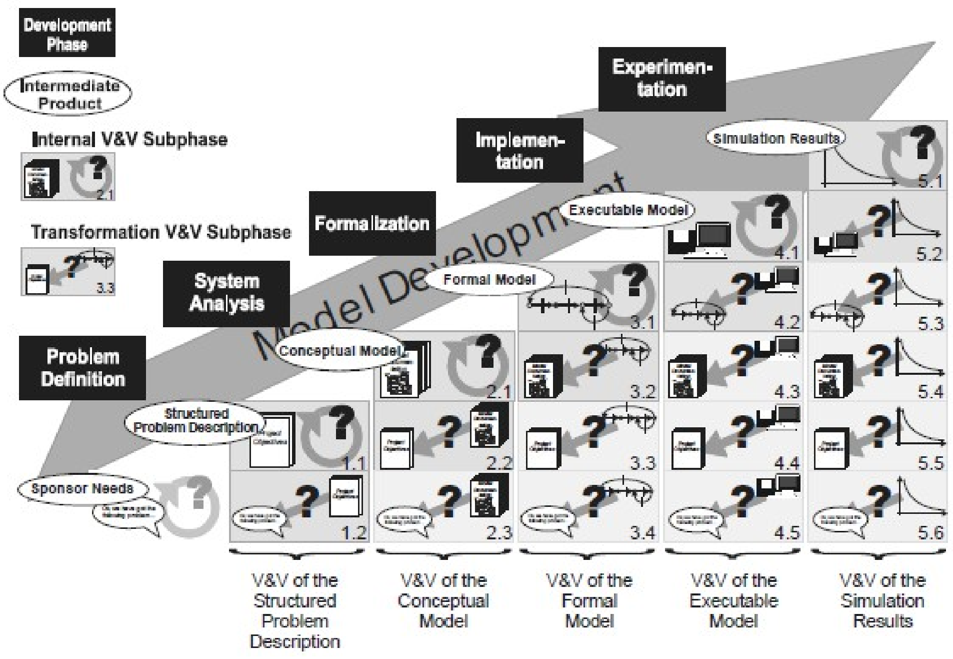


Figure 3.2: Life-cycle Model for Modeling proposed by Brade (taken from (Brade, 2004))

A different approach is taken by Sawyer and Brann (2008), they propose to employ an agile software development method for the creation of simulation models and simulations. They also point out that in early phases frequent feedback pays off if customers and model or simulation developers are different persons.

A salient feature of those life-cycles is their iterative nature (see Figure 3.1 p. 51, 3.4 p. 54 and 3.5 p. 55), in which typically the model is successively refined by elaboration and enrichment (Shannon, 1975, 1998). However, some life-cycles do not provide explicit iterative descriptions. For instance, the life-cycle models of Law and Brade focus on a single pass per simulation study only (see Figures 3.3 p. 53 and 3.2 p. 52).

An intrinsic and important part of each modeling and simulation life-cycle are validation and verification (V&V) processes (Sargent, 2013). Rabe et al. argue that each product of each phase within the life-cycle should be tested Rabe et al. (2009). For instance, conceptual model, formal model, executable model, simulation results, raw data, and prepared data should all be subject to validation

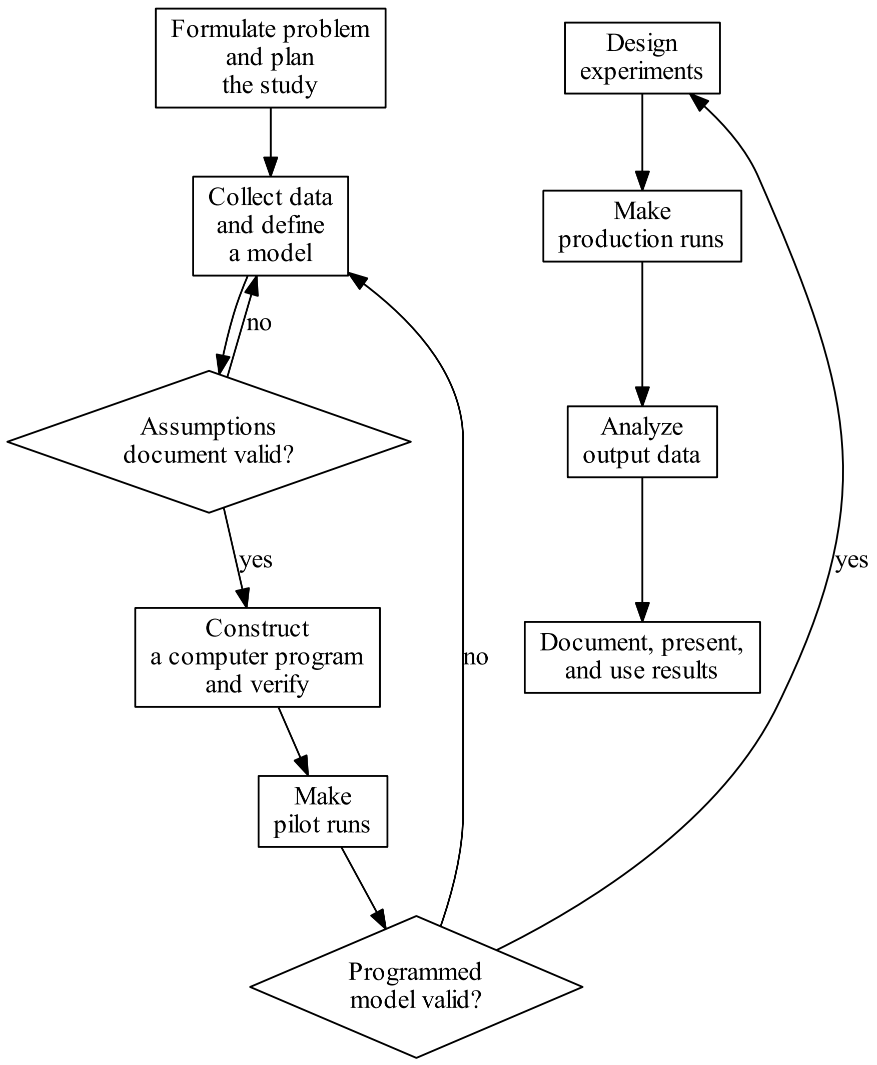

Formulate problem and plan the study
Collect data and define a model
no
Assumptions document valid?
yes
Construct a computer program and verify
Make pilot runs
Programmed model valid?
no
yes
Design experiments
Make production runs
Analyze output data
Document, present, and use results

Figure 3.3: Life-cycle Model for Modeling proposed by Law (adapted from (Law and Kelton, 2007))

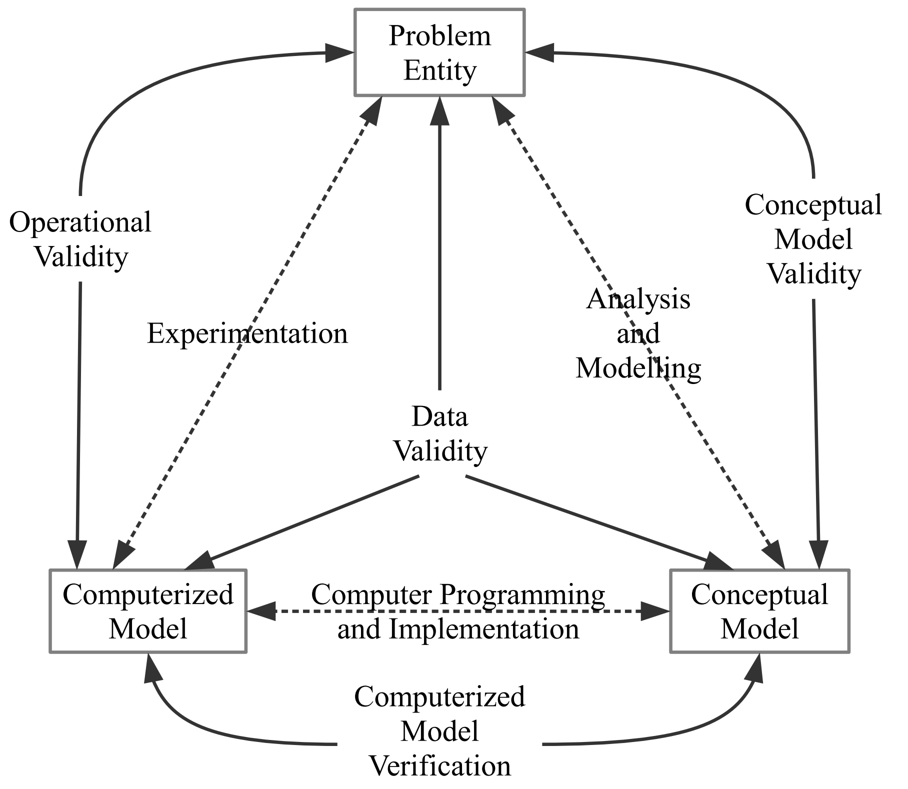


Figure 3.4: Life-cycle Model for Modeling proposed by Sargent (adapted from (Sargent, 2008))

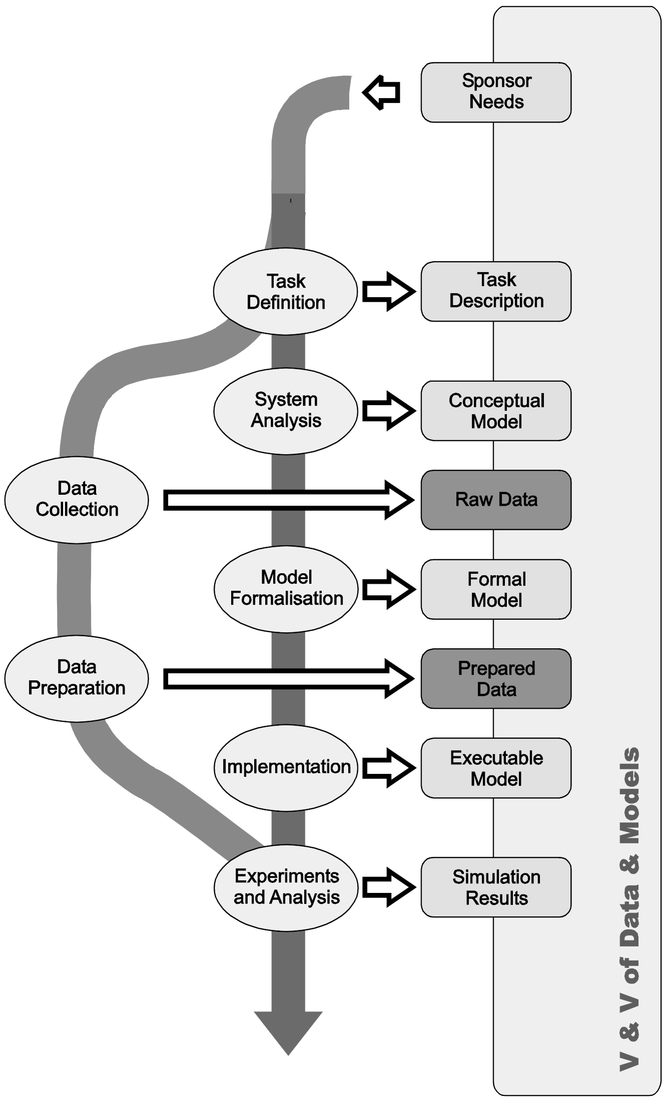


Figure 3.5: Life-cycle Model for Modeling proposed by Rabe (taken from (Rabe et al., 2009))

and verification. Albeit, some analyses refer to a single result, often analyses refer to results or products of different phases. This also implies that if a result is revised based on the analysis all depending phases need to be revisited, too. The life-cycle by Brade (2004) shown in Figure 3.2 p. 52 shows the V-model applied to the field of Modeling & Simulation. It clearly shows the dependency of products in subsequent phases on results produced in the phases before.

Actual methods, such as verification methods, analysis methods or methods for executing models, are not part of the life-cycle but are reserved for the concrete instantiation of a life-cycle, because they highly depend on the application and question at hand. However, intelligent techniques can help to select concrete methods for the problem at hand, e.g., selecting suitable methods for executing models (Ewald et al., 2010b; Lattner, 2013; Leye, 2014).

3.4 Workflow Approaches in Modeling & Simulation

Workflows are recently receiving more and more attention in the scientific domain. So it is not surprising that workflow systems that specifically support scientific processes are emerging, such as Project Trident (Barga et al., 2008a), Taverna (Hull et al., 2006; Oinn et al., 2006), Kepler (Altintas et al., 2004; Ludäscher et al., 2006).

Albeit, those systems focus on the support of arbitrary scientific processes and let scientists create their own workflows, they can be used to express and execute Modeling & Simulation processes, e.g., Taverna is used by Ribault and Wainer (2012) and Wang and Wainer (2015) to deploy simulations into the cloud for emergency planning or to perform crowd simulation in a service oriented manner respectively. Also, there exist extensions (Lee and Neuendorffer, 2007) and systems, such as SYCAMORE (Weidemann et al., 2008), SWANTOOLS (Perrone et al., 2008) and SAFE (Perrone et al., 2012) to explicitly support Modeling & Simulation processes. Furthermore, Sonntag et. al employ workflow techniques from the field of business processes and apply them to scientific processes and Modeling & Simulation processes in particular (Sonntag et al., 2010b,a; Görlach et al., 2011; Sonntag et al., 2011).

The SwanTools provide a web based framework for the automation of the entire simulation workflow with SWAN (Liu et al., 2001). It assists and guides the scientist when configuring models with parameters by using an information rich interface that should enhance the understanding of what each parameter does. It also helps the scientist to manage and create simulation experiments and their configuration by letting the user define data for simulation runs and also to select or provide a specific simulator. Additionally, simulation runs are generated for the scientist and can automatically be distributed in, e.g., a cluster.

Sycamore is a web based front end supporting, e.g., Copasi (Hoops et al., 2006) as simulation engine, different online resources like databases, and locally available tools. It guides through the process of setting up a model by selecting kinetic data from a connected database, adjusting parameters, model checking, parameter estimation, sensitivity analysis, and simulation execution using Copasi.

The simulation automation framework for experiments (SAFE) (Perrone et al., 2012) is explicitly designed to support simulation experiment specifications using a XML based format. It is build around the network simulator engine NS-3 (NS-3 Consortium, 2013). The SAFE workflow is implicit and the process supports the following tasks: *experiment specification*, *model configuration* and *model execution* as well as *data collection* and *data analysis*. In addition, different guiding schemes are supported based on user profiles, e.g., novice or expert user.

While SwanTools, SAFE and Sycamore provide explicit guidance through a simulation process the actual workflow is hidden and predefined. However, the underlying processes are inspired by classic Modeling & Simulation life-cycle models.

In the workflow approach for simulation using business process modeling techniques the modeled process is tailored to a specific question, model and simulation algorithm, such as solid body simulation using OPAL (Binkele and Schmauder, 2003) or simulating the diffusion of ink in a box of water using DUNE (Blatt and Bastian, 2007) using the workflow shown in Figure 3.6 p. 58.

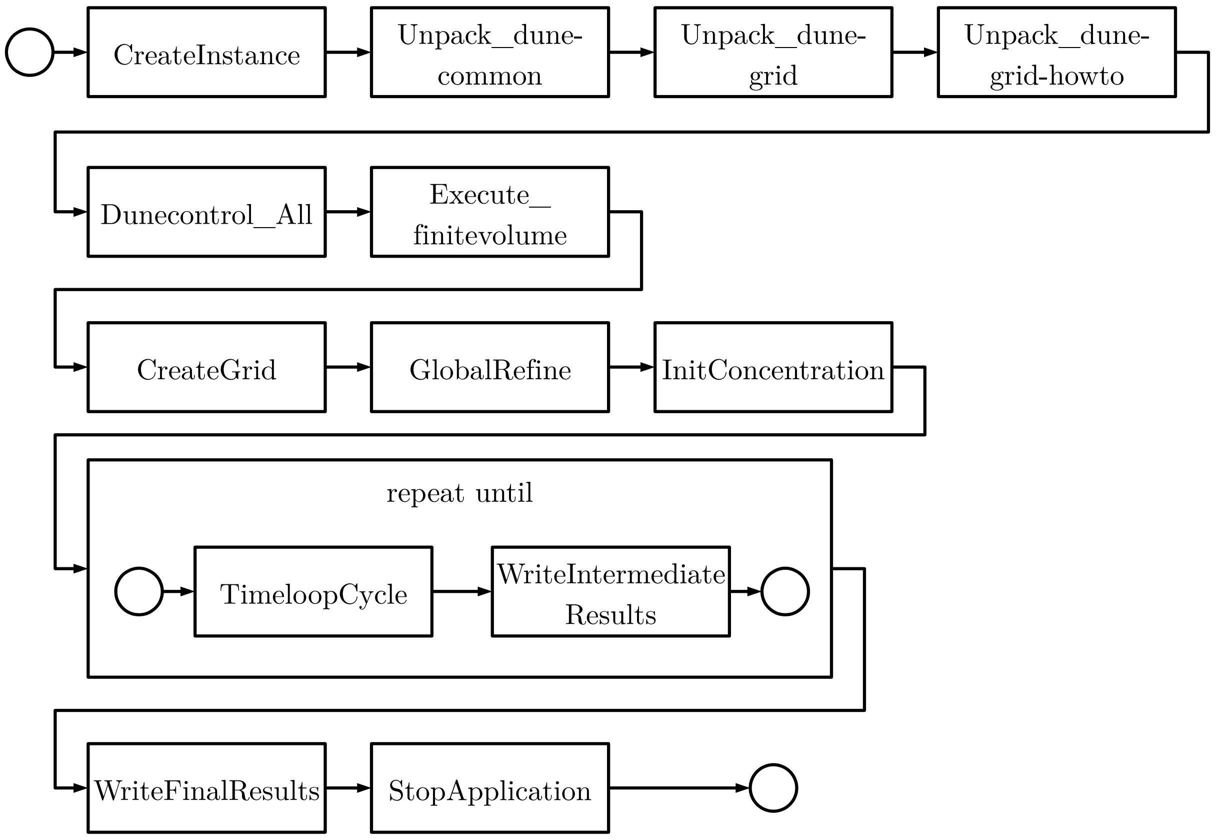


Figure 3.6: Simulation workflow using business process modeling techniques (adapted from (Görlach et al., 2011))

3.5 Anatomy of a Simulation Study

Life-cycle models lend themselves as entry point for determining which tasks constitute a simulation study. Generally, a simulation study can be divided into two main layers. Layer one deals with the creation of a simulation model, that can be used with layer two to conduct simulation experiments. Layer one itself comprises different phases, such as *conceptual modeling* or *conceptual model V&V* phase. Similarly, layer two itself is also divided into multiple phases, e.g.,*experiment specification*, *model configuration* and *model execution* phase. Interestingly, the presented Modeling & Simulation Life-Cycle models mainly deal with layer one, with conducting an experiment (layer two) being typically only one atomic task in that Life-Cycle. However, the presented workflow approaches and tools mainly deal with layer two and typically expect an already validated and verified model to begin with.

3.5.1 Layer One — The Model Creation

The model is the primary product of the modeling and simulation life-cycle. However, when referring to model, different terms like *abstract model*, *conceptual model*, *communicative model*, *formal model*, *programmed model*, *computational model*, *executable model*, *operational model* (Nance, 1986; Balci, 2003; Sargent, 2008; Rabe et al., 2009; Robinson, 2011; Chwif et al., 2013) can be found. Not all proposed life-cycle models and workflow approaches distinguish between models at the same granularity, e.g., in Rabe et al. (2009) the terms *conceptual model*, *formal model*, and *executable model* are used. Albeit the same terms are used, the meaning of those terms varies. Sometimes, *abstract* or *conceptual model* means the not yet expressed perception that a person has about a system. Sometimes, the terms *conceptual* or *communicative model* are used to refer to a model, which uses variables and inter-dependencies between those variables to describe a system. *Programmed model*, *computational model*, or *executable model* typically refer to an implementation of a model on a computer. However, an explicit phase for creating an *executable model* can be neglected in cases where the *formal model* can and is automatically transformed into an *executable model*. The *operational model* refers to a successfully validated *executable model*, ready to use for simulation experiments.

Despite the diversity of the terms used, most of the approaches distinguish between *conceptual*, *formal*, and *executable model*. Thereby, a clear distinction between *conceptual* and *formal model* is of central importance. The *conceptual model* is an integral part of the modeling process and builds the foundation of the *formal* and *executable model*. It is one of the very first products and it shall be independent from the simulation formalism or the simulation algorithm that will be used later for simulation (Chwif et al., 2013).

Albeit *conceptual* and *formal model* being distinguishable, diverse methods are proposed to support conceptual modeling that use formal approaches like Petri Nets and DEVS (Heavey and Ryan, 2006; Chwif et al., 2013). However, in

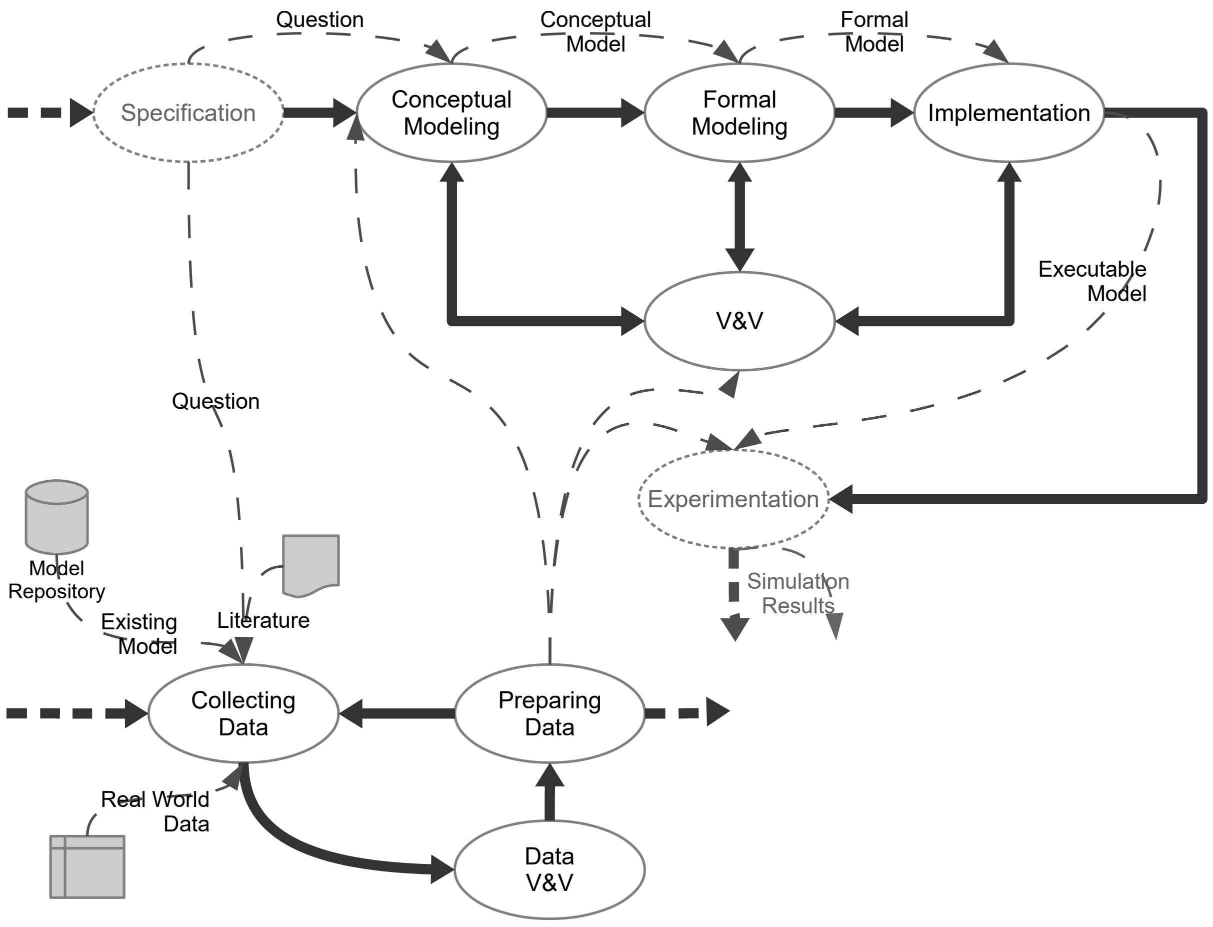


Figure 3.7: Model creation Life-Cycle used for Layer One

this case *conceptual* and *formal model* would not be distinguishable anymore. In addition, using such a formal approach for describing a *conceptual model*, would restrict implementation and simulation algorithm to that formalism or it would make it hard to transform it into another *formal* or *executable model* using a different formalism, e.g., using Petri Nets for conceptual modeling and partial differential equations (PDE) to describe the *formal model*.

A modeling formalism constrains what can be described and how it can be described. The answers to the questions what should be formally described and how should it be described are one result of the conceptual modeling phase.

Based on the presented Life-Cycle models as well as workflow approaches shown, the following tasks and products for layer one, the model creation layer (see also Figure 3.7 p. 60) are derived:

Problem Definition The problem definition is the question about a system that should be answered. It needs to be well formulated because it builds the foundation for subsequent products and tasks such as the conceptual, formal and executable model as well as the various verification & validation (V&V) tasks.

Conceptual Modeling Based on the problem definition and its requirements a model is created that can be used to help people (scientists and stakeholders) to understand and communicate the system modeled. The resulting model is a conceptual model. A conceptual model sometimes represents the not yet expressed perception that a person has about a system. However, typically the model uses variables and inter-dependencies between those variables to describe the system to be modeled.

Conceptual Model V&V Once the conceptual modeling task is finished and a conceptual model is present, it needs to be verified and validated according to the problem definition as well as to the system that is modeled. For instance, this can be done by reviewing variables used as well as the defined dependencies between variables. If verification or validation fails, the conceptual model needs to be revised.

Formal Modeling The next task turns the conceptual model into a formal model using a suitable formalism, e.g., DEVS, Petri Nets or MLRules. The result is a formal model according to the conceptual model.

Formal Model V&V Similar to conceptual model V&V, the formal model needs to be verified and validated as well and if either fails at least the formal model needs to be revised.

Executable Model Creation Once the formal model is available an computer executable version, the executable model, needs to be created. This can be done manually, by using, e.g., a generic programming language or a toolbox such as Matlab (MathWorks, 2005). However, in case where the formal model is specified using a domain specific language for a given formalism, e.g., MLRules, the transformation of formal to executable model can be automated. Assuming that the executable model can be automatically derived from the formal model a V&V task for it should not be necessary as long as the transformation algorithm was verified.

Executable Model V&V In order to ensure verifiability and validity across the modeling process, the executable model needs to be verified and validated as well. However, this task might be skipped in cases in which the executable model is created using an automated transformation based on the formal model and the transformation algorithm itself was verified.

Data Collection Alongside the modeling tasks data can be collected that relate to the problem definition and is necessary to conduct validation on e.g., the formal or executable model.

Data Preparation The data collected might not directly be usable for, e.g., validation or modeling. For instance it might be needed to derive additional data, such as kinetic rates based on some analysis methods, before it can be used for validation or modeling.

Data V&V When using data, it needs to be ensured, that it is valid, e.g., that it was obtained using appropriate methods and prepared using proper analysis methods.

It should be noted that as shown in Figure 3.7 p. 60, the presented tasks are not necessarily performed in sequence, the entire model creation process is merely an iterative process. For instance, if a conceptual model V&V fails the conceptual model needs to be revised. The same is true for the formal model, if formal model V&V fails. However, it might even be necessary to revise the conceptual model in that case, as the problem in the formal model stems from a problem in the conceptual model. In addition, tasks such as data collection and preparation as well as executable model creation or formal model V&V can be performed concurrently. However, proper actions have to be taken in failure cases, e.g., a task such as executable model creation is executed on a not yet validated formal model and formal model V&V fails, in which case the executable model needs to be revised or discarded. This also implies that V&V tasks only need to take place once previous V&V tasks succeeded, e.g., formal model V&V makes only sense if conceptual model V&V succeeded.

Eventually, layer one is completed when an executable model was created and all used intermediate products, such as conceptual model, formal model and data were successfully passing V&V.

3.5.2 Layer Two — The Simulation Experiment

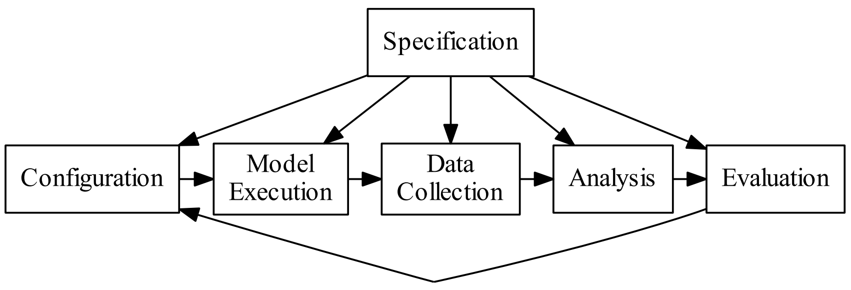


Figure 3.8: The Six Tasks of a Simulation Experiment and their interaction adapted from (Leye, 2014)

In the Modeling & Simulation life-cycle models the simulation experiment is typically considered only after model creation as a black box task. The presented workflow approaches, however, try to also distinguish between different tasks involved in a simulation experiment in order to provide clearly defined process of how to conduct a simulation experiment. Leye identifies the common tasks that make up a simulation experiment based on life-cycle models and workflow approaches. Six common tasks are identified, the “Six Tasks of a Simulation Experiment” (Leye, 2014), which layer two (Simulation Experiment) of conducting a Simulation Study is based on. Those tasks are *Specification*, *Configuration of Model Parameters*, *Model Execution*, *Data Collection*, *Analysis* and *Evaluation*, and are briefly described in the following and depicted in Figure 3.8 p. 63.

Specification The specification is an essential task of a simulation experiment because it connects the experiment's goal with the actual execution. All subsequent tasks of a simulation experiment have to work according to that specification which ideally is described formally. In addition, the specification creates the foundation of reproducible simulation experiments by providing clear and sufficient information about the experiment to conduct.

Configuration of Model Parameters The configuration of model parameters is similar to defining test cases in software development. Model parameter settings are being selected that are to be investigated during the performance of the simulation experiment. Different methods exist to create such parameter settings. On the one hand, those parameter settings can be selected statically, such as when conducting a parameter scan. On the other hand, parameter settings can be generated using parameter search methods. Parameter search methods typically require a feedback loop from the simulation execution because they use data or analysis results to generate further parameter settings if necessary. They are typically used in optimization experiments.

Model Execution Using a simulation algorithm the model execution is performing the computation of the simulation model using given parameter setting. The algorithm should be accurate and efficient, however some algorithms trade accuracy for efficiency.

Data Collection The data collection task is responsible for collecting data during model execution, which is needed in subsequent tasks, such as analysis. It builds the foundation for evaluating the results of the simulation experiment. However, similar to model execution this task should be efficient and only collect necessary data to save memory on the one hand and computation cycles on the other in order to not slow down model execution too drastically.

Analysis Data from data collection is analyzed, e.g., to detect a steady state. Knowledge gathered through analysis can be used to steer the model execution, e.g., stopping or continuing. In addition, it can further be used in subsequent tasks. The analysis itself is a complex task which can be divided into two phases. Phase one involves the analysis of only a single model execution. Phase two involves the analysis of multiple model executions (for the same parameter setting), e.g., using aggregated collected data. However, phase two is only necessary in stochastic simulations.

Evaluation The evaluation is the final task of a simulation experiment. It bases on results from the previous analysis task or tasks. The purpose is two fold. On the one hand it can give feedback to other tasks such as the configuration task, in case more parameter settings are necessary. On the other hand, it provides the results generated by the simulation experiment e.g., visualization and sensitivity information of parameters, on which decisions concerning the objective of a simulation study can be based.

3.6 Summary

When conducting a simulation study a number of different products are used, created and altered over time. They range from software products, over problem definitions or experiment specifications, simulation model, analysis methods to the used process during execution of the study. Since each product is created differently involving different processes, it exhibits a different indication of its quality.

Quality is a desired property of every product involved in the performance of a simulation study. However, quality depending on the product at hand is a more or less tangible metric. Thus, different views on quality exists, e.g., proposed by Garvin those are the transcendence, product-based, user-based, manufacturing-based, and value-based view. However, a common understanding of quality of a product is that it is directly related to the process used to create that product. In other words, a well defined process is essential in order to provide products of quality. When employing a well defined process reproducibility and transparency can be ensured. Errors can be pinned down to their sources within the process more easily improving the process over time. An underlying process that is tested, established and well defined directly reflect on the products quality positively.

Those established, tested and applied processes can be found in so-called life-cycle models of specific product groups. For instance there exist different life-cycle models for the development of software products, such as the waterfall model or more recent scrum model from the domain of agile software development. Developing software based on those life-cycle models ensures a certain quality of software taking part in a simulation study constituting to the overall quality of the simulation study. There even exist dedicated norms, e.g., ISO 9000-3, dealing with process quality of software development processes.

Moreover, life-cycle models can also be found in the domain of Modeling & Simulation. Among classic established models are the ones proposed by Balci, Sargent, Law and Rabe. Alternative models similar to the ones from the area agile software development are also proposed, e.g., by Sawyer and Brann. Nevertheless, all of them exhibit an iterative nature in which the simulation model is successively refined, verified and validated. The verification and validation (V&V) of each phase and product is a salient feature of those life-cycle. Albeit V&V plays an intrinsic part in those life-cycles, actual V&V methods and techniques are not part of the definition of the life-cycle models. They are rather specific for each instance of such a life-cycle as V&V varies with the specific problem and question at hand.

Additional to life-cyle models, workflows and workflow systems tailored for the scientific domain are emerging. Systems include for instance Taverna, Kepler and Trident, which are generally targeted at scientific processes but can also be applied in the particular domain of Modeling & Simulation. Furthermore, systems dedicated to support Modeling & Simulation processes exist, e.g., SAFE and SYCAMORE. There are even systems from the domain of business process management that are applied to Modeling & Simulation processes. However, the workflow used internally by those systems is hidden but is based on established life-cycle models for Modeling & Simulation.

In order to create a well defined process for conducting a simulation study, it is imperative that the anatomy of a simulation study is clear. Generally, a simulation study can be divided into two layers. On the one hand there is the model creation layer, denoted as layer one. On the other hand there is the simulation experiment performance layer, denoted as layer two.

Interestingly, the aforementioned Life-Cycle models deal with layer one in a rather detailed way while they cover layer two typically as one atomic task. On the other side the workflow approaches deal with layer two and assume that a valid and verified model from layer one is already present, hence neglecting layer one in their consideration.

In layer one the model is the primary product. It exists in different forms throughout this layer, e.g., conceptual and formal and executable model. The proposed process model for the creation of model is based on the aforementioned life-cylce model, unifying and abstracting terms. The resulting process is still iterative in nature and involves different phases. It starts with the *problem definition* and iterates over *conceptual modeling, conceptual model V&V, formal modeling, formal model V&V, executable modeling, executable model V&V, data collection, data preparation* and *data V&V.*

Layer two on in contrast deals with the simulation experiment and its execution. Albeit it is a separate layer it might already take intrinsic part in layer one, e.g., when a V&V method uses simulation. However, it is typically considered taking part after model creation. The proposed process for layer two closely orients itself towards the six common tasks of a simulation experiment proposed by Leye. Those tasks are *specification, configuration of model parameters, model execution, data collection, analysis* and *evaluation.*

4 Workflows in the Modeling & Simulation framework JAMES II

I can resist everything except temptation.

Oscar Wilde

4.1 Overview of the Modeling & Simulation framework JAMES II

JAMES II is a framework for conducting simulation experiments and is developed in Java. It allows developing models in different modeling formalisms and running simulations using different simulation algorithms.

A plug-in-based architecture makes JAMES II highly extensible and customizable. By following the paradigm of separation of concerns components are clearly defined, exchangeable and used throughout the entire framework. For instance, there are plug-ins for random number generators, seed generators, event queues, simulation algorithms, modeling formalisms, data storages, model and simulation instrumenters, optimization algorithms, steady state detectors, replication criteria, simulation stop policies, model editors, model descriptions lan-

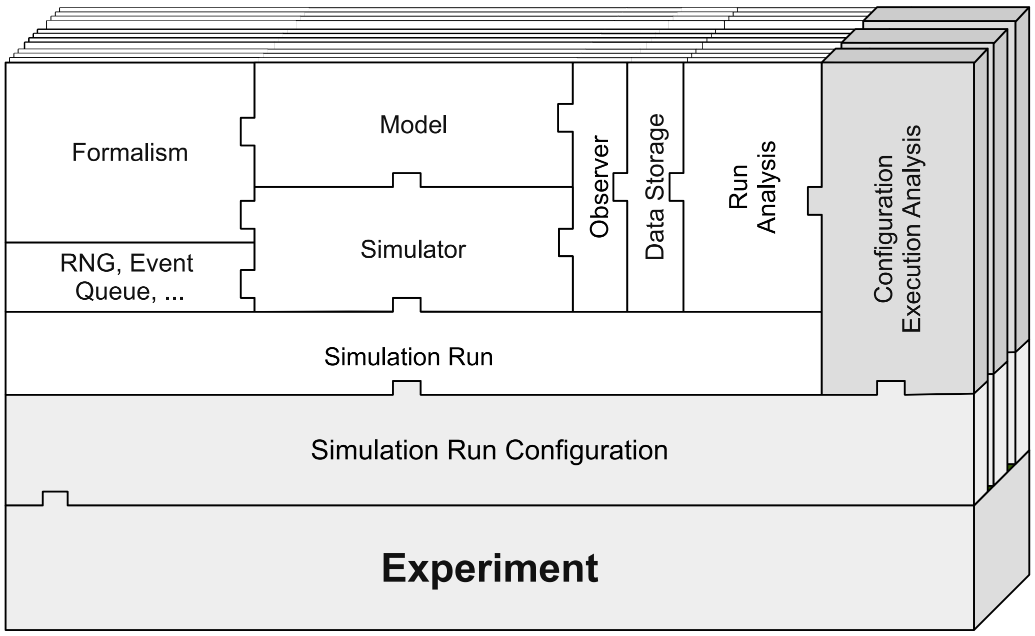


Figure 4.1: The experimentation layer of JAMES II

guages, experiment editors and so on. Although JAMES II can be used standalone, it can also be used as pure library providing reusable components to build a custom modeling and simulation environment.

4.1.1 Layer One — The Model Creation

In terms of model creation JAMES II does not restrict model development by any predefined patterns. Developing a model in JAMES II is done on the one hand, by using one of the available model editors. Those range from graphical editors, e.g., for cellular automata, to textual editors supporting different syntaxes, highlighting and instant error reporting when using one of the modeling languages supported by JAMES II, such as MLRules (Maus et al., 2011) or the Attributed π Calculus (John et al., 2008). On the other hand, models can be directly developed using the Java-based formalisms model API, e.g., for advanced cellular automata models (Kossow et al., 2015) or DEVS (Concepcion and Zeigler, 1988) models.

However, as far as guidance or user assistance during model development goes JAMES II provides no means besides syntax highlighting and error reporting when using text-based model editors. JAMES II models, when created can be seen as formal models regarding the life-cycle models presented in Chapter 3 p. 45. They are transformed into executable models automatically by the experimentation layer of JAMES II.

Nevertheless, JAMES II provides abilities, e.g., to validate model using validation experiments. Even so, a user is not directly assisted or encouraged to run, e.g., validation during model creation. Also, the model creation process is not documented, hence no provenance data of how a model evolved over time and what validation procedures where applied to a model is available.

4.1.2 Layer Two — The Simulation Experiment

As far as what simulation experiments are concerned JAMES II provides a flexible experimentation layer backed by many plug-ins, which is not limited to a specific modeling formalism or simulation algorithms. It can handle arbitrary modeling formalisms and simulation algorithms, such as MLRules (Maus et al., 2011), the Attributed π Calculus (John et al., 2008) and DEVS (Concepcion and Zeigler, 1988) as well as multitude of experiment types, such as parameter scan, optimization or validation experiments (Himmelspach et al., 2008).

The experimentation layer and its workflow is fixed and implicit and driven by its components and their dependencies. An experiment and its basic components are depicted in Figure 4.1 p. 70. Basically, an experiment is made of one or more simulation run configurations, which each having a number of simulation runs (replications). Each simulation run employs a model and a simulation algorithm (simulator) applying to a specific formalism. Internally, the simulation algorithm can use further components such as random number generators, event queues and so on. Moreover, a simulation run can be instrumented using observers and data can be stored using data storages, which can output to, e.g., XML, databases, CSV and more. Ultimately, different analyses can be applied on simulation results using results from one or more simulation runs. This can be achieved using previously stored data or in a streaming fashion (Schützel et al., 2014). Therefore, the experimentation layer resembles the six tasks of a simulation experiment presented in Section 3.5.2 p. 63.

Listing 4.1: A simple simulation experiment conducting a parameter scan specified in Java using JAMES II as a framework. Herein an MLRules model (Line 3) is simulated, either for one second (Line 15) or until the simulation time reached 10×10^4 (Line 20), whatever comes first. The model will be simulated with varying parameter configurations for $r1$ with 0.5, 1 and 1.5 (Line 7) and each parameter configuration will be replicated ten times (Line 41). During simulation species A is observed over time and outputted to the console whenever the simulation time passes a multiple of 50 (Line 33–36).

```java
 1 BaseExperiment exp = new BaseExperiment();
 2
 3 exp.setModelLocation(new URI("file-mlrj:///./model.mlrj"));
 4
 5 //set parameter r1 to be scanned at 0.5, 1 and 1.5
 6 ExperimentVariables vars=new ExperimentVariables();
 7 vars.addVariable(new ExperimentVariable<Double>("r1", new SequenceModifier<Double>(0.5d, 1d,
       1.5d)));
 8 exp.setExperimentVariables(vars);
 9
10 //set stop criterion to either wall clock time or simtime
11 List<Pair<ComputationTaskStopPolicyFactory<?>, ParameterBlock>> factories = new ArrayList<>();
12
13 //wall clock time stop policy setup
14 ParameterBlock stopParam = new ParameterBlock();
15 stopParam.addSubBl(WallClockTimeStopFactory.SIMEND, 1);
16 factories.add(new Pair<ComputationTaskStopPolicyFactory<?>, ParameterBlock<>(new
       WallClockTimeStopFactory(), stopParam));
17
18 //sim time stop policy setup
19 stopParam = new ParameterBlock();
20 stopParam.addSubBl(SimTimeStopFactory.SIMEND, 10e4);
21 factories.add(new Pair<ComputationTaskStopPolicyFactory<?>, ParameterBlock>(new
       SimTimeStopFactory(), stopParam));
22
23 //Build a combined stop policy based on above policies
24 ParameterBlock params = new ParameterBlock();
25 params.addSubBl(DisjunctiveSimRunStopPolicyFactory.POLICY_FACTORY_LIST, factories);
26 exp.setComputationTaskStopPolicyFactory(new
       ParameterizedFactory<ComputationTaskStopPolicyFactory<?>>(new
       DisjunctiveSimRunStopPolicyFactory(), params));
27
28 //set instrumenter to observe A, every 50 time steps (0, 50, 100, 150, etc.)
29 ParameterBlock parameters = new ParameterBlock();
30 parameters.addSubBl(MLRulesModelInstrumenterFactory.PARAM_OBSERVER_FACTORYNAME,
       MLRulesModelObserverFactory.class.getName());
31
32 //set the query that is used during observation
33 parameters.addSubBl(MLRulesModelInstrumenterFactory.PARAM_QUERY_STRING, "INSTRUMENT model
       SELECT COUNT(species.absquantity) WHERE species.name = 'A' GROUP BY species.name EVERY 50
       T;");
34
35 //define output destination of observed data
36 parameters.addSubBl(MLRulesModelInstrumenterFactory.PARAM_LOGGING_STREAM, System.out);
37
38 exp.setModelInstrumenterFactory(new ParameterizedFactory<ModelInstrumenterFactory>(new
       MLRulesModelInstrumenterFactory(), parameters));
39
40 //do ten replication for each parameter configuration
41 exp.setRepeatRuns(10);
42
43 exp.execute();
```

Listing 4.2: A simple SESSL experiment using JAMES II as simulation provider mirroring the same experimentation setup as shown in the Java version (see Figure 4.1 p. 72).

```
import sessl._
import sessl.james._

execute {
      new Experiment with Observation {
            model = "file-mlrj:///./model.mlrj"
            scan("r1" <~ (0.5, 1, 1.5))
            replications = 10
            stopCondition = AfterWallClockTime(seconds=1) and AfterSimTime(10e4)
            observe("A")
            observeAt(range(0, 50, 10e4))
            withRunResult {
                  result => println(result ~ "A")
            }
      }
}
```

Experiments are controlled by an experiment steerer that also defines the type of the experiment (parameter scan, optimization, etc.). A steerer can act automatically, but can also be driven manually using visual analytics techniques (Luboschik et al., 2012, 2014). In order to describe experiments there are two options in JAMES II. Firstly, experiments can be described programmtically using JAMES II' Java API, for instance to set up a simple parameter scan experiment the code shown in Listing 4.1 p. 72 can be used. Secondly, SESSL (Simulation Experiment Specification via a Scala Layer), a Domain Specific Language (DSL) can be used (Ewald and Uhrmacher, 2014). It uses a declarative approach to describe simulation experiments including analysis, data storage, replication criteria, stop policies and so on. A sample SESSL experiment is shown in Listing 4.2 p. 73 defining the same parameter scan experiment shown in Listing 4.1 p. 72).

Nevertheless, while the experimentation layer is flexible and covers a multitude of different experiment scenarios, it makes it hard to control, document and provide provenance information automatically. In fact, no documentation or provenance information is provided for the execution of a simulation experiment by JAMES II other than data that can be obtained using instrumentation or analysis. However, this information does contribute very little if at all to provenance information.

4.2 Extending JAMES II with workflows

In order to cope with the demand for provenance information and documentation, accompanied by the need for user guidance and assistance workflows mapping the modeling and simulation life-cycle to workflows should be integrated into JAMES II. Herein, documentation and provenance information should be provided for the whole range from model creation to conducting a simulation study. In fact, user assistance should be seamlessly derivable from the integrated workflows.

4.2.1 Layer One — The Model Creation

The process of model creation is a process with a lot of degrees of freedom in terms of task order, task iterations and task selection. It is highly human interaction driven with little automatable tasks. However, constraints for a high quality and credible model still apply. For instance, it is imperative to verify and validate a model during all stages of modeling, ranging from conceptual, over formal to executable model. Moreover, if a valid model is changed its valid flag needs to be revoked and the model has to be revalidated. This revocation also needs to propagate to subsequent models in order to be concise. Therefore, a declarative workflow description approach is chosen for defining the process of model creation.

In fact, an artifact-based workflow description is used to specify the modeling process derived from the life-cycle models presented in Section 3.5.1 p. 59. An artifact-based workflow description directs its focus onto the involved artifacts and their life-cycle. The life-cycle is subject to restrictions defined through constraints which influence tasks that can be executed and interact with artifacts. Using the artifact-based workflow description allows defining constraints declaratively.

Defined constraints can be interpreted, e.g., by forward planning algorithms. Albeit it is possible to generate an imperative workflow based on the artifact-based description precautions dealing with adaptation have to be taken as artifacts with their constraints can easily be added, deleted, or modified. It is up to the workflow engine to evaluate this description and to derive a sound workflow execution.

The artifact notation employed is the one of Guard, Stage and Milestone (GSM) as presented by Hull et al. (2011).

An artifact represents an entity in a process as a whole, including data and a life-cycle. The data corresponds to the artifact's information model which describes what data an artifact comprises. Each artifact's life-cycle consists of *stages*, *guards* and *milestones*. During the life-cycle an artifact goes through different stages and achieves specific milestones. Stages in turn are activated by guards, where a guard acts as precondition for stage activation depending on milestones, data or other active/inactive stages of the artifact. Milestones summarize the results of a stage, for instance a milestone can indicated whether a stage completed successfully or not. Stages can be nested and stages on the same level of nesting can be, as long as their guards allow it, activated simultaneously.

Moreover, the GSM notation also includes the notion of *sentry*. Sentries can be seen as global guards that can access multiple artifacts. They can be triggered internally or externally, due to changes within one or more artifacts, for artifacts are able to react to milestones and stage activation as well as are able to directly manipulate milestones. This is particularly helpful when milestone need to be cascaded between artifacts, e.g., if the validation of a previously successfully validated conceptual model fails the milestone of successful validation of the formal model needs to be reset to invalid as well.

In order to evolve an artifact's life-cycle tasks are executed. Which task are executable, depends on active stage information from artifacts. The result of a task execution is used to determine milestones for stages. In other words, tasks are used to change the life-cycle state an artifact is in.

4.2.1.1 Artifacts, Guards, Stages & Milestones

The artifact-based workflow model of the modeling process includes three artifacts, the *Conceptual Model* artifact, the *Formal Model* artifact, and the *Data* artifact. In the following those artifacts and their responsibilities are described.

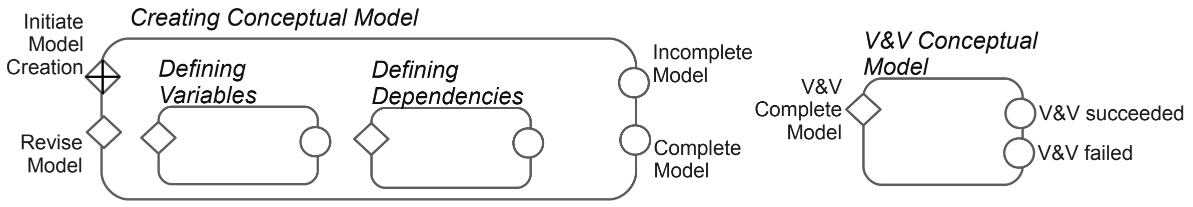


Figure 4.2: The stages of the *Conceptual Model* artifact. A *guard* is represented as diamond and is attached to a stage. A *stage* is represented as rounded rectangle. Stages can be nested and stages on the same level of nesting can be activated simultaneously. *Milestones* are represented as circles and are used to summarize the results of a stage, e.g., successfully completed.
The artifact comprises two main stages, capturing *model creation* and *model verification and validation (V&V)*. The creation stage is divided further into *defining variables* and *defining dependencies* stages. Moreover, the creation stage has two guards allowing the activation of that stage and two milestones indicating whether a model is *complete* or *incomplete*. The V&V stage has only one guard allowing the activation only in case of the *Complete Model* milestone and also two milestones indicating whether V&V was *successful* or not.

Conceptual Model The *Conceptual Model* artifact represents instances of conceptual models during their creation, change, verification and validation (V&V). It is depicted in Figure 4.2 p. 76. It distinguishes between two main stages, the *creation stage* and the *V&V* stage. Whereas the creation stage is sub-divided into a *Defining Variables* and a *Defining Dependencies* stage. They are defined with empty guards and no specific milestones, meaning both are active simultaneously, as long as the model creation stage is active allowing the definition of variables and dependencies interchangeably.

Both the V&V stage and the creation stage provide two milestones they can finish with. The latter provides a milestone for an *Incomplete Model* or *Complete Model*. Herein a complete model is a model that can be verified and validated, meaning variables used in dependencies are defined and variable definition is sound and complete. However, a complete model milestone does by no means imply that the model is complete with respect to the question of interest and the system under study. Moreover, the stage has two guards, whereas either of which can activate it. Firstly, the *Initiate Model Creation* guard activates on artifact creation. Secondly, the *Revise Model* guard is active as long as the model is incomplete or V&V failed.

The V&V stage has a *V&V succeeded* and a *V&V failed* milestone, indicating whether the validation or verification of the conceptual model was successful or not. In case it failed it is expected that the model is revised and artifact milestones that require a successful validated and verified conceptual model need to be revoked. The stage features a guard that allows V&V to be active in case a complete model is available, meaning the *Complete Model* milestone is set.

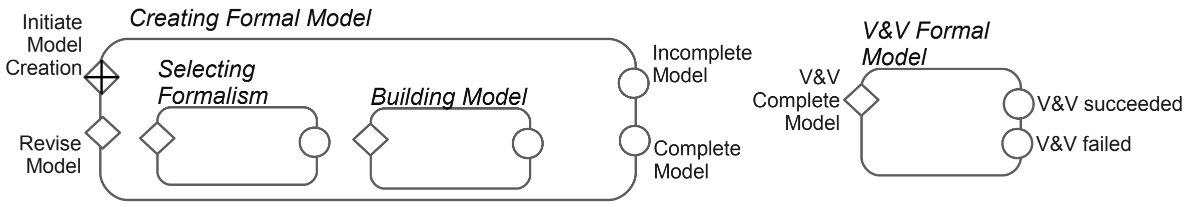


Figure 4.3: The stages of the *Formal Model* artifact. The artifact comprises two main stages, capturing *model creation* and *model verification and validation (V&V)*. The creation stage is divided further into *selecting a formalism* and *building model* stages. Moreover, the creation stage has two guards allowing the activation of that stage and two milestones indicating whether a model is *complete* or *incomplete*. The V&V stage has only one guard allowing the activation only in case of the *Complete Model* milestone and also two milestones indicating whether V&V was *successful* or not.

Formal Model The *Formal Model* artifact depicted in Figure 4.3 p. 77 captures the process of creating, revising, verifying and validating of a formal model.

Analogously to the *Conceptual Model* artifact, it distinguishes between two main stages, the *creation stage* and the *V&V* stage. Whereas the creation stage is also sub-divided into a two sub-stages, namely *Selecting Formalism* and a *Building Model* stage. The former is defined with an empty guard and a milestone indicating that a formalism is selected. The latter defines a guard that expects the selected formalism milestone of the previous stage before it can be activated.

Again, similar to the *Conceptual Model* both the V&V stage and the creation stage provide two milestones. The creation stage provides a milestone for an

Incomplete Model or *Complete Model*. A complete model is a model that can be verified and validated, i.e., no syntactic errors exist. Though, a complete model milestone does not imply that the model is complete or semantically sound with respect to conceptual model it resembles. Additionally, the stage has two guards, either of which can activate it. On the one hand there is the *Initiate Model Creation* guard activates on artifact creation. On the other hand there is the *Revise Model* guard which active as long as the model is incomplete or V&V failed.

The V&V stage has a *V&V succeeded* and a *V&V failed* milestone, indicating whether the validation or verification of the formal model was successful or not. In case it was not successful, it is expected that the model is revised. The stage features a guard that allows V&V to be active in case a complete model is available, meaning the *Complete Model* milestone is set.

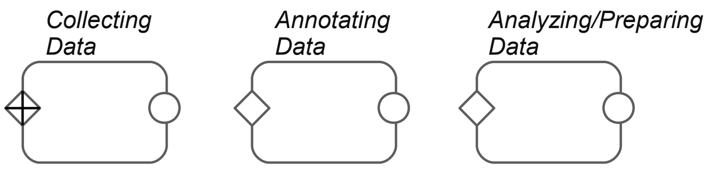


Figure 4.4: Stages of the *Data* artifact. The artifact comprises three stages, capturing *the collection of data*, *the annotation of collected data* and *the analysis and preparation of data*. The collecting data stage maps the process of data collecting, e.g., from literature, wet-lab experiments, etc. . Whereas the annotation stage and analysis stage base on collected data. Those stages are used to provide data that can be used in simulation experiments, verification or validation.

Data The *Data* artifact, shown in Figure 4.4 p. 78, is responsible for handling data that is associated with and its stages map any kind of external data that is important for the creation of *Conceptual Model* or *Formal Model*. It refers to heterogeneous information sources, whereas such sources can be, e.g.,:

- specific literature, for instance about the system to be modeled
- other already existing models, for instance self-created, retrieved from liter-

ature or model repositories, e.g., the BioModels Database (Le Novere et al., 2006),

- experiment description, for instance retrieved from literature, self-created or from experiment repositories, e.g., the myExperiment Project (Roure et al., 2008; Goble et al., 2010),
- data obtained in experiments, wet-lab or in silico experiments, such as simulation experiments,
- notes and media such as images and video, e.g., taken during conducting experiments

Each instance of those types becomes its own *Data* artifact instance.

The *Data* artifact consists of three stages, the *Collecting Data*, *Annotating Data* and *Analyzing/Preparing Data* stage. The first is meant for retrieving and collection data, by e.g., studying literature, querying a model repository or conducting a wet-lab experiment. Its guard activates this stage directly on artifact instantiation. Once data is collected the collected milestone of that stage is set and the other two stages can be activated. Data can be annotated, analyzed and prepared interchangeably and multiple times. Prepared data can be used by other artifact stages such as the V&V stages of the *Formal Model* or *Conceptual Model* artifact, for instance wet-lab data can be used in the V&V stage of the *Formal Model* artifact in order to validate simulation trajectories with the real system.

4.2.1.2 Artifacts and their interaction

Each artifact has a life-cycle of its own. However, artifacts can influence each others' life-cycle as well as interact with each other.

Figure 4.5 p. 80 illustrates a selection of interaction and dependencies between artifacts. Herein, different interaction and dependencies are marked using different colors. Brown arrows indicate a local interaction between stages of one artifact, while green arrows indicate an interaction between stages of different artifacts. Lastly, blue arrows indicate the possible use of an artifact that has

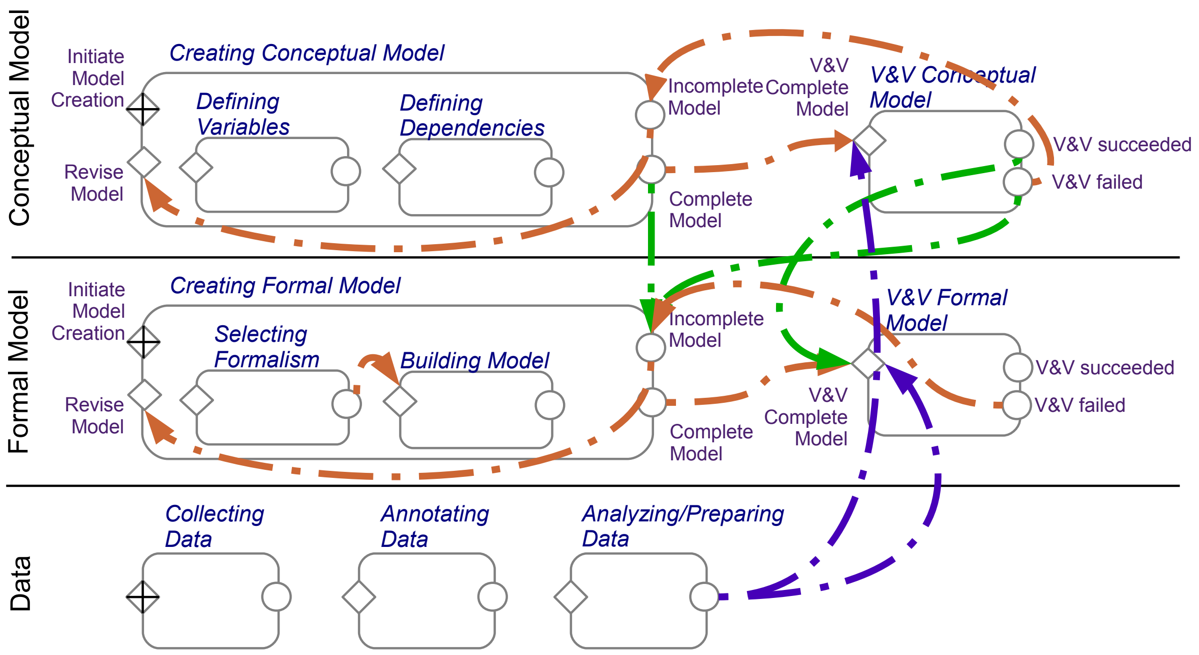


Figure 4.5: Some interactions between artifacts are shown. Brown arrows indicate a local interaction between stages of one artifact, e.g., a guard relying on a specific milestone. Green arrows indicate an interaction between stages of different artifacts, here it is shown for the *Conceptual Model* and *Formal Model* artifact and depicting the influence of a failed and succeeded V&V stage of the *Conceptual Model* stage on milestones and guards of the *Formal Model* stage. Blue arrows indicate the use of an artifact in a specific milestone state by other artifacts, for instance *Prepared Data* as input to the V&V stages of the *Conceptual Model* and *Formal Model* artifact.

reached a specific milestone by other artifacts. The illustration is not exhaustive to avoid cluttering. For instance, the local interaction between the milestone of *Collecting Data* and the guards of *Annotating Data* and *Analyzing/Preparing Data* of the *Data* artifact is not shown.

Local interactions, i.e., indicated by brown arrows are realized using guards and milestones. In the *Conceptual Model* artifact, the *Creating Conceptual Model* stage is activated on artifact instantiation. It allows tasks able to define a conceptual model, e.g., interactively using a graph based editor, to be executed. Also, when this stage activates the two sub-stages, *Defining Variables* and *Defining Dependencies* are also active. The task that is executed during one of those stages will return an incomplete or complete model eventually. This will trigger the corresponding milestone to be set. As long as the *Incomplete Model* milestone is set, the V&V stage's guard prevents the activation of that stage. The only option left is to revise the *Conceptual Model* further, reentering the first stage until a complete model emerges.

Once the *Complete Model* milestone is set the V&V stage's guard releases that stage to be activated. There can be multiple tasks that are applicable to the active V&V stage, depending on the model and additional *Data* artifacts. Each of those tasks can be selected to be executed simultaneously. The tasks will report on the outcome of the V&V and result in the *V&V succeeded* or *V&V failed* milestone accordingly.

In case of a failed V&V stage an incomplete milestone of the creation stage is triggered, automatically triggering the revise model guard allowing the option of revising the *Conceptual Model*. Note that the triggers and dependencies like this are defined using sentries.

The *Formal Model* artifact exhibits similar local interaction. It automatically activates the *Creating Formal Model* stage on artifact instantiation. However, only one of the two sub-stages will be also activated automatically, namely the *Selecting Formalism* stage. The other stage, *Building Model*, has a guard that allows activation only after a formalism is selected in the other stage. Once a formalism is selected, hence the *Building Model* stage is active, tasks that support the selected formalism and creation of a formal model can be used to build it. Those tasks will result in an *Incomplete* or *Complete Model*, setting the corresponding milestone of the model creation stage.

Analogously to the *Conceptual Model* artifact, the V&V of the *Formal Model* artifact is only activated once the *Complete Model* milestone is set, leaving only the revision of the model until the complete model emerges. Consequently, once the complete model milestone is set the V&V stage is activated. V&V tasks can be executed during this stage and will result in either successful or failed verification or validation. Milestones of the V&V stage will be set accordingly to those results. If the V&V failed, the incomplete model milestone is set and results in the revision of the model. The last interaction is again handled using a sentry.

Sentries can be used for local interactions but are also used when artifacts need to interact with each other, as indicated using green arrows in Figure 4.5 p. 80. As an example, the *Formal Model* artifact depends substantially on the *Conceptual Model* artifact. That is, the V&V stage of the *Formal Model* artifact can only be active if the V&V stage of the *Conceptual Model* succeeded. Consequently, the *Incomplete Model* milestone of the *Formal Model* artifact is triggered by a failed V&V stage of the *Conceptual Model.* The same happens if the *Conceptual Model* is incomplete the *Formal Model* is so too. Moreover, the *Formal Model* is also incomplete if the *Conceptual Model* reaches a complete milestone after the *Formal Model* was completed, indicating a revision of the *Conceptual Model* which should automatically lead to a revision of the *Formal Model*, too.

Finally, there are the dependencies between artifacts that indicate the use of an artifact by another, represented as blue arrows. In Figure 3.7 p. 60 such a dependency can be observed between the *Data* artifact, which can be used, by each of the V&V stages of the *Formal Model* and *Conceptual Model* artifacts. This can happen, e.g., when a *Data* artifact captures data obtained in a wet-lab experiment and this data is used as comparison data set for validation of the developed *Formal Model.* However, this is only one use case of using a *Data* artifact. Since the *Data* artifact can also represent another formal model described in literature, it could be used as starting point for deriving an extended formal model created by the *Formal Model* artifact.

4.2.1.3 Tasks

Until now the artifact-based workflow description uses Guards, Stages and Milestones and acts as a meta-model to the process described. However, the life-cycle of an artifact is driven by tasks that can take place during specific stages and trigger specific milestones. Execution order and executability is controlled by the GSM meta-model whereas tasks can interact (read, change and add) with the artifact's data.

Tasks can be categorized as local or global tasks. The former are only executed in the context of one artifact only relying only on information and stages from that artifact. Local tasks only affect that artifact in terms of milestones it triggers and data. On the other hand, global tasks are able to interact with one or more artifacts, i.e., requiring specific stages of different artifacts to be active. Global tasks can in return also affect milestones in multiple artifacts.

In general, there can be any number of tasks, each of which relying on conditions under which it can be executed. Those conditions can be:

- a specific stage is active
- a specific milestone of a specific artifact is reached or not yet reached
- specific data in one or more artifact is available
- a combination of the above (i.e., one ore more artifacts with multiple active stages, multiple reached milestone and specific data available)

The integration of the workflow model presented here into JAMES II comprises the following possible tasks:

- a graph editor/mind map editor for building the conceptual model
- Editor for building the formal model, e.g., the MLRules model editor
- V&V techniques as presented by Balci (1998) and in particular the ones provided by JAMES II using the FAMVal framework (Leye and Uhrmacher, 2010)
- V&V techniques, e.g., statistical model checking, face-validation and cross-validation for the formal model (Leye and Uhrmacher, 2010; Leye et al., 2010)
- analysis techniques for data preparation
- model retrieval techniques to access model repositories

4.2.2 Layer Two — The Simulation Experiment

4.2.2.1 The WorMS Workflow Management Framework

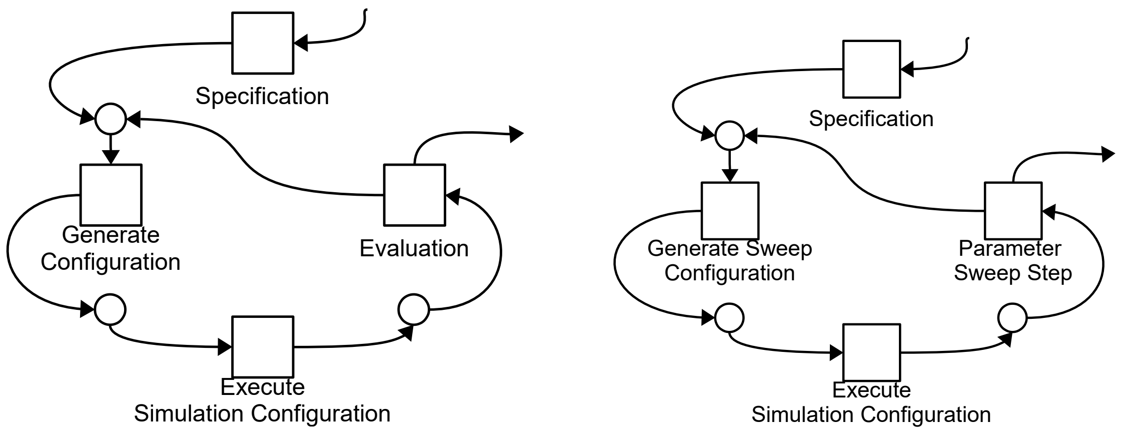
WorMS (Workflows for Modeling & Simulation) is a framework that allows the integration of workflows in modeling and simulation software (Rybacki et al., 2011). It uses techniques and concepts from business process modeling as well as scientific workflow. For example the description of workflows is more control-flow oriented than it is data-flow oriented, however it provides means to handle data-flow as well. The underlying workflow representation bases on workflow nets (Van Der Aalst and Van Hee, 2004), allowing easy verification and workflow analysis.

WorMS is built using exchangeable components (see Figure 5.3 p. 108), allowing the extension and adaptation of the system. In a nutshell the framework consists of a *Workflow Engine*, different *Monitoring* components, a *Workflow Executor*, a *Data Store* and a workflow model represented in the *Intermediate Representation*. Herein the *Workflow Engine* manages and controls the scheduling and execution of workflows, while the *Workflow Executor* in orchestration with a *Data Store* are responsible for the actual execution of a workflow. Essential to provide documentation and provenance for the execution of a workflow are the *Monitoring* components that are responsible for recording necessary information, such as system information, software used, work item execution order, work item input/output data and so on.

Additionally, WorMS provides components and extension points to provide *Security and User/Role Management*, *Converter*, *Administration*, *Analysis* and *Workflow Repository* functionality. WorMS uses plug-ins to integrate and exchange those components similarly to JAMES II. However, the plug-in mechanism is also exchangeable and abstracted by the *Plug-in Provider*. This mechanism allows actual plug-in systems, e.g., the registry of JAMES II can be integrated, keeping the framework flexible within different Modeling & Simulation systems.

WorMS is used for integrating workflows into JAMES II in order to provide documentation, reproducibility and provenance to simulation studies conducted with JAMES II. A more technical and thorough description of WorMS can be found in Section 5.2.1 p. 107.

4.2.2.2 Simulation Workflow with Template and Frames


Specification
Generate Configuration
Evaluation
Execute Simulation Configuration
Specification
Generate Sweep Configuration
Parameter Sweep Step
Execute Simulation Configuration

(a) Example of a workflow describing an experiment for evaluating a simulation algorithm (b) Example of Workflow describing a plain parameter sweep experiment

Figure 4.6: Experiment workflows for (a) evaluating a simulation algorithm and (b) a Workflow describing a plain parameter sweep experiment

As already mentioned, JAMES II provides implicitly defined workflows hidden inside its experimentation layer implementation. In general, there are two different options for integrating workflows into the experimentation layer of JAMES II. Firstly, specific parts of the existing experimentation layer could be replaced by workflows, e.g., the calculation of experiment variables or the orchestration of experiment execution (Rybacki et al., 2012a). Secondly, the entire experimentation layer could be replaced by a workflow-based experimentation layer implementation, making the experimentation process explicitly representable as workflow. A workflow based experimentation layer would exhibit flexibility, ease of changability, adaptation and extension. Additionally, it would also lead to an automatically documentable simulation experiment.

Herein the latter is chosen because it closes the gap between layer one and layer two, making guidance and documentation seamless. It also provides a more detailed and more complete documentation trace throughout an entire simulation study. In order to achieve this, WorMS is introduced as it is specifically tailored to support workflows in Modeling & Simulation software.

The presented workflow description is oriented towards the Six Tasks of a Simulation Experiment presented in Section 3.5.2 p. 63 and bases on the implicit experimentation process that is already present in the existing experimentation layer (see Figure 4.1 p. 70 for the different levels and interacting components).

Specification Having a workflow covering the Six Tasks of a Simulation Experiment, i.e., specification, configuration, data collection, model execution, analysis and evaluation allows its reuse in any experiment workflow.

For instance, assuming an experiment for evaluating a simulation algorithm the workflow shown in Figure 4.6(a) p. 85 would be used, whereas for a simple parameter sweep experiment the workflow shown in Figure 4.6(b) p. 85 is applied. An apparent observation is the similarity of the shown workflows for two different experiments. It clearly exhibits three of the six tasks, i.e., specification, configuration and evaluation at the top level of the workflows covering the *experiment* and *simulation configuration* level in the current experimentation layer. They also show the *Execute Simulation Configuration* task as a crucial part of the entire experiment workflow, comprising the remaining three tasks, model execution, data collection and analysis.

However, an essential part of any simulation experiment is the simulation experiment specification describing the objectives and goals of the actual experiment to conduct, without which a simulation experiment cannot be conducted. Which makes it a perfect entry point for the experimentation layer workflow.

Configuration Task Based on the specification different simulation configuration can be derived and executed, e.g., configurations evaluating a simulation algorithm by testing different executable models for accuracy or different simulation algorithms for performance or sweeping over a model's parameter space. Hence, the configuration task is placed right after the specification in the top-level experimentation workflow. It is also connected to the workflow task responsible for the evaluation of already executed simulation runs in order to provide more or adjusted configurations for further simulation runs.

Evaluation Task Also, at the top-level workflow, the evaluation workflow task, responsible for mapping the evaluation task of the Six Tasks of a Simulation Experiment onto the experimentation layer workflow. It evaluates simulation data from one for multiple runs and configurations according to the experiment's specification, that is the objectives and goals. It directly influences the generation of subsequent simulation configurations, hence works in concert with the configuration workflow task.

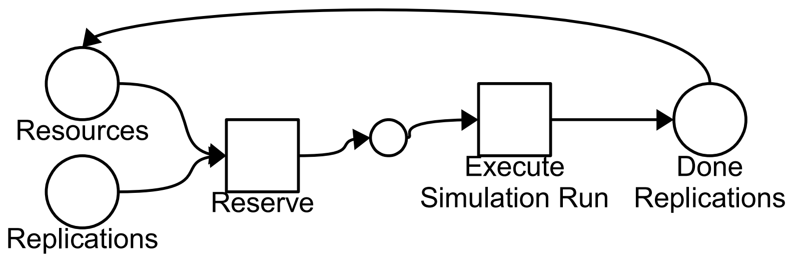


Figure 4.7: The workflow handling the execution of a specific simulation configuration as presented in (Rybacki et al., 2012a). Execution resources, which can be, e.g., machines on a grid, are explicitly modeled.

Model Execution Task Part of the Six Tasks of a Simulation Experiment is the model execution, which is represented within the *Execute Simulation Configuration* task in concert with Data Collection and Analysis. It comprises the experimentation layer level *Simulation Run.* A workflow description to represent the level of *Simulation Run* was already been proposed in (Rybacki et al., 2012a). It features the explicit handling of available compute resources within the workflow and is depicted in Figure 4.7 p. 87. Compute resource can be, e.g., another computer or a CPU-core on the current machine. This workflow is used as starting point for extending it further to also include the remaining tasks of a simulation experiment.

However, resource handling such as allocation, scheduling and so on is pulled out from this workflow. Resource management is instead explicitly handled by the workflow management system WorMS. In particular this is handled by the *Workflow Executor* (see Section 5.2.1.3 p. 122 and Section 5.2.2 p. 147). Which results in the simplified workflow shown in Figure 4.8 p. 88 as starting point for

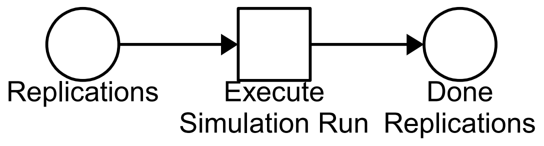


Figure 4.8: The same workflow as shown in Figure 4.7 p. 87 with explicit execution resource handling removed.

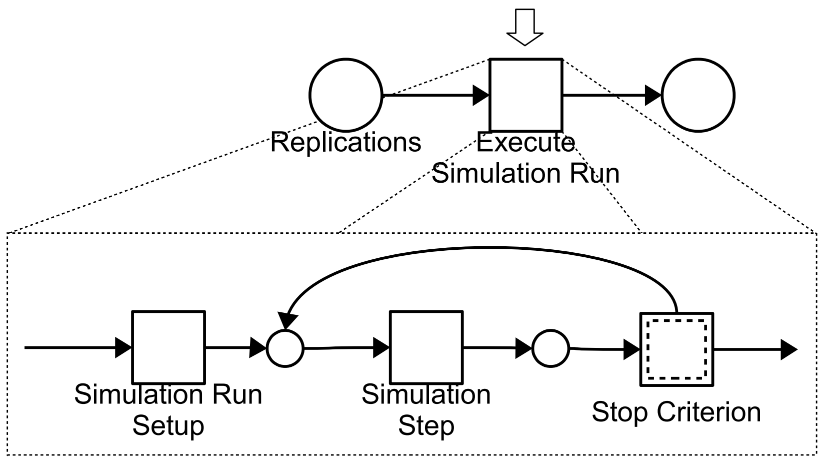


Figure 4.9: The *Execute Simulation Run* activity refined as a more detailed workflow.

further extension.

The workflow can further be extended using a sub-workflow for the *Execute Simulation Run* task as depicted in Figure 4.9 p. 88. The sub-workflow replaces the *Execute Simulation Run* by another workflow that describes the simulation experiment at the level of detail of a *Simulation Step*. It also incorporates additional tasks such as checks whether more simulation steps need to be taken using a simulation *Stop Criterion* and the *Simulation Run Setup* task, which is responsible for instantiating the executable model and for providing a suitable simulation algorithm used by the *Simulation Step* task. A simulation algorithm can be chosen using different strategies from the set of simulation algorithms applicable for the execution of the model at hand.

Firstly, an algorithm can be selected randomly from the set of simulation algorithms. Secondly, an algorithm is specified in advance in the Specification Task.

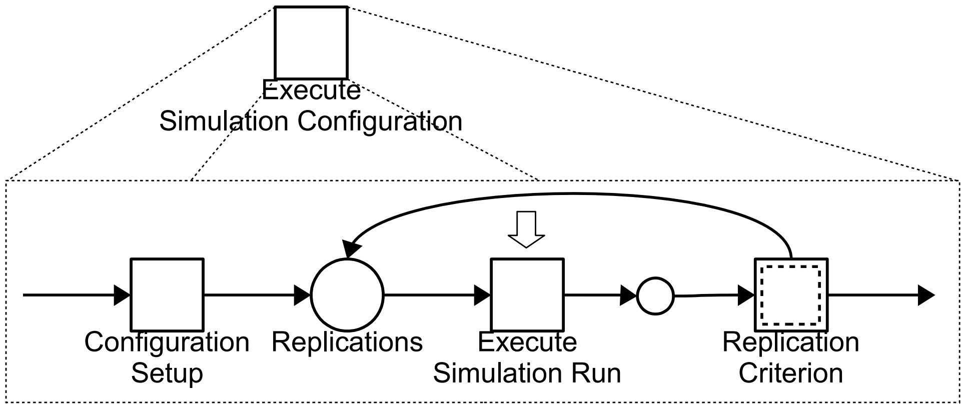


Figure 4.10: Extending the workflow from Figure 4.8 p. 88 with the tasks *Configuration Setup* and *Replication Criterion.* Thereby, a new task for the given sub-workflow is defined, i.e., *Execute Simulation Configuration.*

Lastly, an algorithm can be selected based on performance. This can happen statically using a performance index specified by developer of the simulation algorithm, dynamically using previous simulation runs as reference (Ewald, 2012) or adaptively during runtime (Helms et al., 2013, 2015).

The previous step refined the *Execute Simulation Run.* Next the workflow is extended by additional tasks, i.e., the *Configuration Setup* task and the *Replication Criterion* task (see Figure 4.10 p. 89). The latter is responsible for determining whether enough replications (i.e., simulation runs for one configuration) are already executed or whether more are needed. The first is responsible for configuring the simulation run, which includes, e.g., selecting model parameters based on the experiments specification and evaluation tasks.

Data Collection Task In order to cope with the responsibilities of the Data Collection Task the experimentation workflow is extended further. In particular the simulation run sub-workflow is extended by a data collection task which is inserted between *Simulation Step* and *Stop Criterion* as shown in Figure 4.11 p. 90. It is responsible for instrumenting the current simulation run, observing model and simulation behavior or data according to the experiment's specification.

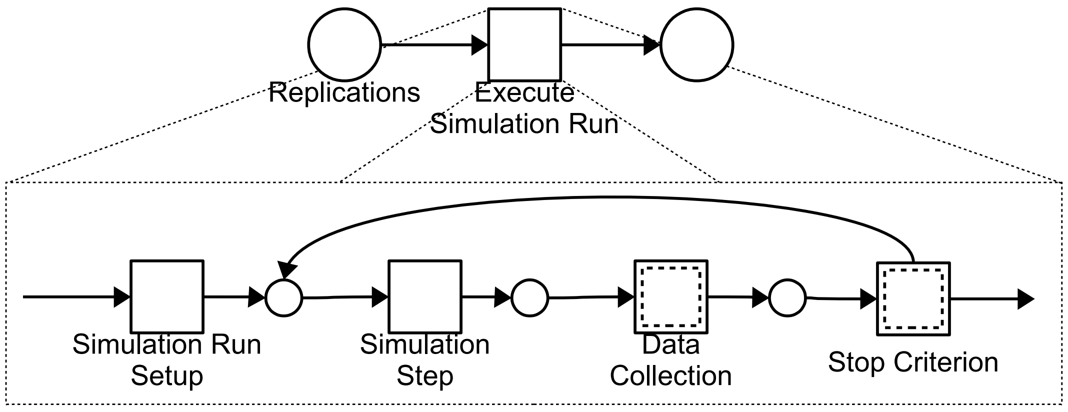


Figure 4.11: Extending the workflow further by incorporating a Data Collection template task

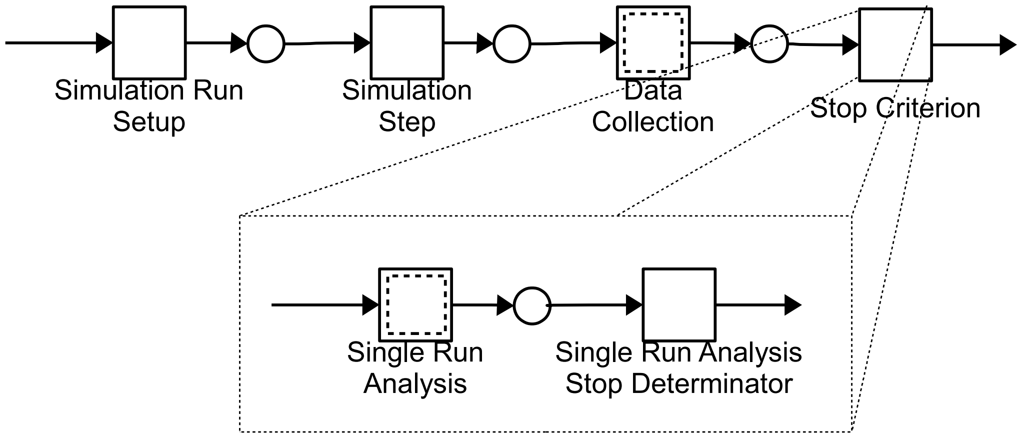


Figure 4.12: Replacing the *Stop Criterion* task by a sub-workflow incorporating a Single Run Analysis template task

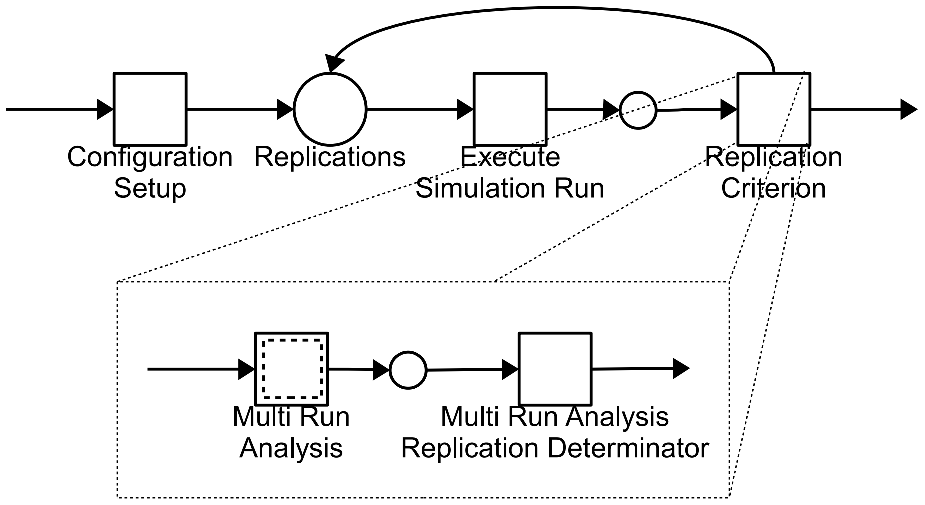


Figure 4.13: Replacing the *Replication Criterion* task with a sub-workflow incorporating a Multi Run Analysis template task

Analysis Task The analysis task is addressed by replacing the *Stop Criterion* and *Replication Criterion* workflow tasks by sub-workflows as shown in Figure 4.12 p. 90 and Figure 4.13 p. 91. The sub-workflow replacing the *Stop Criterion* task handles the analysis for single simulation runs, meaning that for analysis purposes only the data from the current simulation run is used. Results from the analysis can then in return be used by the *Single Run Analysis Stop Determinator*, which functions as *Stop Criterion*. The sub-workflow replacing the *Replication Criterion* task is responsible for handling analysis that may use data from multiple simulation runs. The *Multi Run Analysis Replication Determinator* replaces the *Replication Criterion* and may make use of results provided by the previous multi run analysis.

Putting it all together leads to the overall experimentation workflow depicted in Figure 4.14 p. 93. It exhibits all necessary constructs to implement the Six Tasks of a Simulation Experiment represented as a WorMS workflow.

To avoid the creation of a dedicated workflow for each possible experiment conductible in JAMES II and to cope with different algorithms for the same experiments, without redefining the experiments workflow, the concept of templates and frames is employed (Bowers et al., 2006). The presented workflow depicts template tasks with an additional dashed inside border. Using templates and frames allows to create one experiment workflow template covering a broad range of experiments. Further information about templates and frames and its implementation in WorMS can be found in 5.2.1.1 p. 117.

4.3 Summary

In this chapter the Modeling & Simulation framework JAMES II was introduced and determined how well it supports the process of creating valid simulation models as well as conducting simulation experiments with regard to providing assistance, documentation and provenance information. It also dealt with the extension of JAMES II by proposing workflows for layer one and layer two of a simulation study.

JAMES II is a framework for conducting simulation studies, ranging from creating models to executing simulation experiments and is written in Java and can be used standalone or as library. It supports different modeling formalisms and simulation algorithms employing a plug-in-based architecture, rendering JAMES II highly extensible and customizable.

The model creation process (layer one) is supported in a highly unconstrained fashion in JAMES II. In fact, JAMES II does not restrict model development to any specific pattern. It however provides different options to create models. There are textural and graphical editors as well as a Java-based model API for specifying simulation models. Assistance during model creation is provided by each of those options independently, e.g., by highlighting syntax and syntax or semantic checks. Besides that, there is no further assistance or predefined process defined in JAMES II for developing simulation models. Nevertheless, it provides all the necessary tools to create valid and verified simulation models. For instance, JAMES II supports a wide range of validation experiments and methods.

For layer two, which deals with the execution of a simulation experiment given a simulation model, JAMES II ships with a flexible experimentation layer. Which enables the execution of different types of experiments, e.g., validation and optimization experiments. The experimentation layer follows the six tasks of a

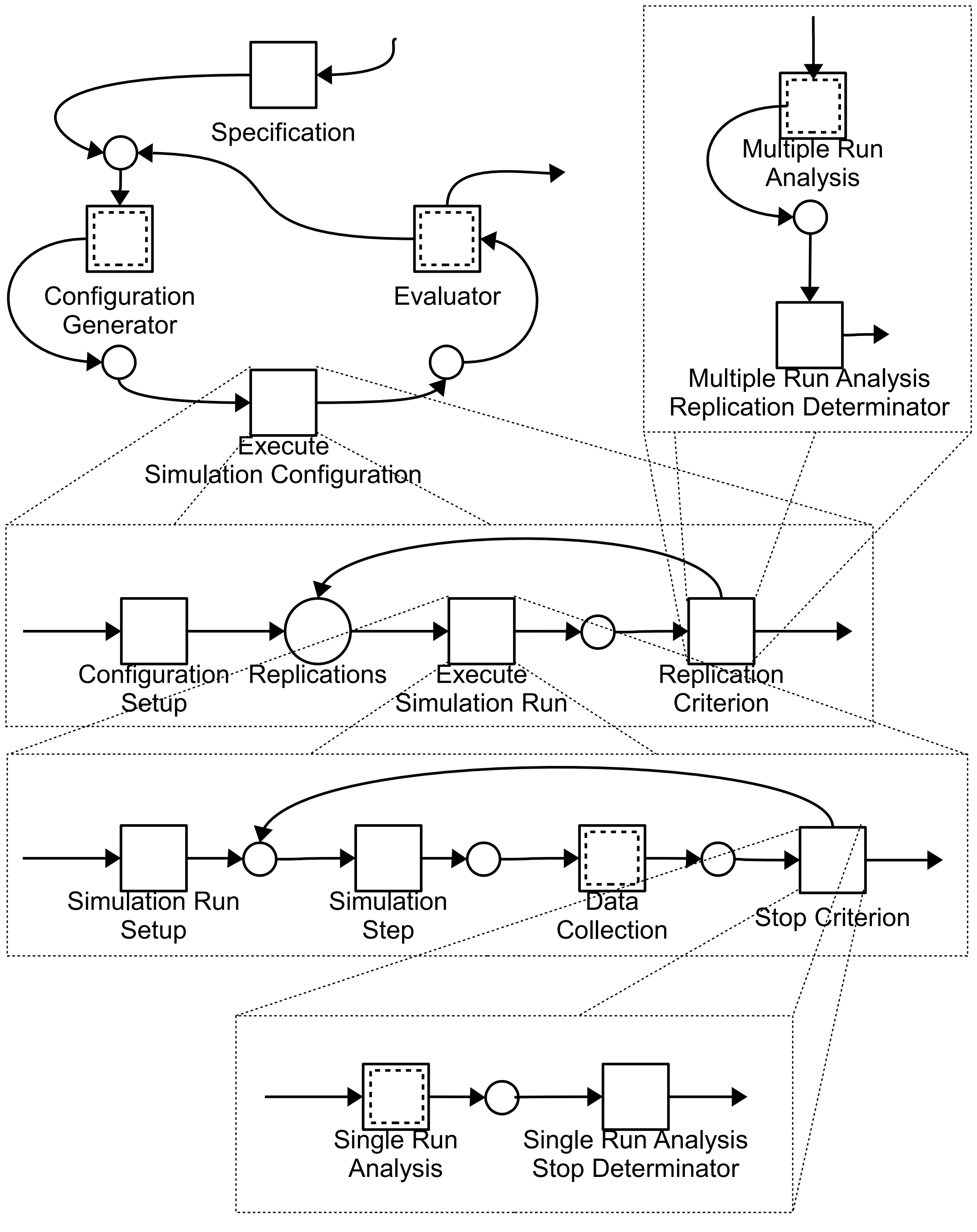


Figure 4.14: Overall workflow template including sub-workflows for a simulation experiment in JAMES II.

simulation experiment and represents an experiment on three different levels. On the top level the experiment with its specification and evaluation is handled, before it is divided into a number of different simulation configurations that then again are divided into a number of replications or simulation runs on the third level.

Experiments can be defined programmatically or declaratively using a dedicated domain specific language (SESSL) in JAMES II. Albeit the flexibility of the existing experimentation layer of JAMES II, the involved process is only implicitly defined and only depends on the used plug-ins and their dependencies. This makes it hard to control, document, monitor and provide provenance information automatically and completely.

To cope with those drawbacks, processes for layer one and layer two explicit defined as workflows are proposed to be integrated into JAMES II. For layer one dealing with the process of creating a simulation model, which is a highly flexible process with a lot of degrees of freedom and highly human interaction driven, a declarative workflow approach is employed. In fact, the GSM (guards, stages and milestones) artifact-based notation is used to describe a workflow for layer one oriented towards established life-cycle models. GSM focuses on involved artifacts and their life-cycle, which is constrained using milestones and guards. An artifact itself represents an entity in the described process as a whole including data and different stages throughout its life-cycle. During its life-cycle, an artifact can be in different stages and fulfill multiple milestones, where the latter summarize results of the stages they belong to.

In order to drive the life-cycle forward tasks are executed which depend on active milestones, stages or data of an artifact and can lead to milestones after execution, affecting the life-cycle of an artifact. Tasks are loosely coupled to the life-cycle of an artifact and are responsible for the actual data processing, data generation or data alteration. For instance, such tasks could be the editing, verifying and validating of a model or collecting and preparing data for further use in simulation experiments or for validation purposes.

The herein proposed artifact-based workflow for layer one bases on established life-cycle models from the domain of Modeling & Simulation and involves the

Conceptual Model artifact, the *Formal Model* artifact and the *Data* artifact. The *Conceptual Model* artifact captures the creation of a valid and verified conceptual model, distinguishing between model creation, model alteration and model validation stages. The *Formal Model* artifact represents the creation of a valid and verified formal model based on the associated conceptual model, distinguishing between model creation, model alteration and also model validation. Lastly, the *Data* artifact deals with the collection, preparation and validation of data. Data can be collected, e.g., from literature, wet-lab experiments or other simulation experiments.

The process of executing a simulation experiment (layer two) involves in contrast to the model creation process little to no human interaction and the execution pattern is rather strict. Thus, a task-based workflow description is more suited than an artifact-based workflow description. In order to integrate such workflows into JAMES II, WorMS was developed.

WorMS is a framework allowing the integration of task-based workflows into Modeling & Simulation software. In particular WorMS builds on an extension of workflow nets internally, which combines techniques from business process modeling and scientific workflows, providing means to manage control and data flow. Similar to JAMES II, it builds on plug-ins and consists of components such as, e.g., *Workflow Engine*, *Workflow Executor*, *Data Store* and components for *Security and User/Role Management*, *Converter* and *Analysis*. Using WorMS enables the creation of documentation and provenance information for executed workflows automatically.

In order to extend JAMES II to capture the execution of a simulation experiment as workflow, a workflow based on workflow nets supporting templates and frames is proposed. Templates and frames allow for a definition of workflow tasks that do not need to be defined during workflow definition but can be filled with actual tasks during execution, based on e.g., parameters, a specification or an objective, allowing to define one workflow covering a multitude of different experimentation scenarios. The herein proposed workflow resembles the entire functionality of the existing experimentation layer of JAMES II making the process clearly defined and explicit. Nevertheless, leaving room for easy adaptation and extension and making guidance and documentation seamless between layer one and layer two. The experiment workflow comprises tasks, templates, frames and sub-workflows for specification, configuration, evaluation, model execution, data collection and analysis. Thus orienting itself towards the six tasks of a simulation experiment which is also already used as basis in the current experimentation layer.

5 Implementation

Unlightening: (v) learning
something that makes you dumber

Daniel Dalton

5.1 Layer One — Artifact-based workflow implementation

The workflow for layer one is described using artifacts. An implementation managing such workflows and supporting the execution has to deal with the declarative nature of the artifact-based approach. It also has to cope with the interaction with human entities which perform activities of the workflow and therefore are not controllable by the management system.

An interesting option for implementing such a system poses the use of rule based systems. In Java the Java Specification Request (JSR) 94 (Toussaint, 2003) lends itself as starting point for using rule based engines in Java. JSR 94 provides a unified API for accessing and using rule based engines allowing different engines supporting JSR 94 to interoperate with each other.

Using a rule based engine allows declarative programming, which fits the declarative nature of an artifact-based workflow well. It helps to specify what to do instead of how to do it, leaving the execution up to the used rule engine. This in return leads to solutions for difficult problems that are easy to express, easy to understand and easy to read by others.

Moreover, rule based engine separate logic and data, putting the logic into rules and data into domain objects or facts. This allows for easier maintaining of logic and makes it easy to define cross domain logic, which would be harder if the logic was integrated directly into the domain object. Furthermore, by separating logic and data, logic is centralized and bundled at one point, typically in a rules repository, rather than hidden within domain objects and distributed across different source files.

Typically using the Rete algorithm or a variation of it rule engines are very efficient and scalable in matching rule patterns in domain objects or facts (Forgy, 1982).

Ultimately, a rule based engine is perfectly equipped for collecting provenance information by providing an explanation facility. It monitors all decisions it made and why it made them during execution, basically logging which rules were applied for a given set of facts and why in which order.

There exists a number of rule engines that support the JSR 94, e.g., JESS, ILOG JRules, RuleML and Drools (Orchard, 2001; Friedman-Hill, 2003; Boyer and Mili, 2011; Boley et al., 2010; Proctor, 2012). They all support JSR 94 and share a common base of functionality only differing in specific areas, for a general comparison on rule engines and their performance please refer to Liang et al. (2009).

5.1.1 Architecture

From the list of available rule engines this implementation architecture uses Drools as it is also used by other implementations of artifact-based workflow systems (Ngamakeur et al., 2012). However, exchanging it for any of the other engines should be possible without too much effort, because of JSR 94.

When using a rule engine to create a framework for the management of artifact-based workflows that follow the Guard-Stages-Milestone paradigm, entities of the artifact-based workflow need to be understandable and usable by the rule engine. An artifact-based workflow consists of artifacts and tasks, where the first consist of guards (sentries), stages, milestones (see Section 4.2.1 p. 74). In order to use

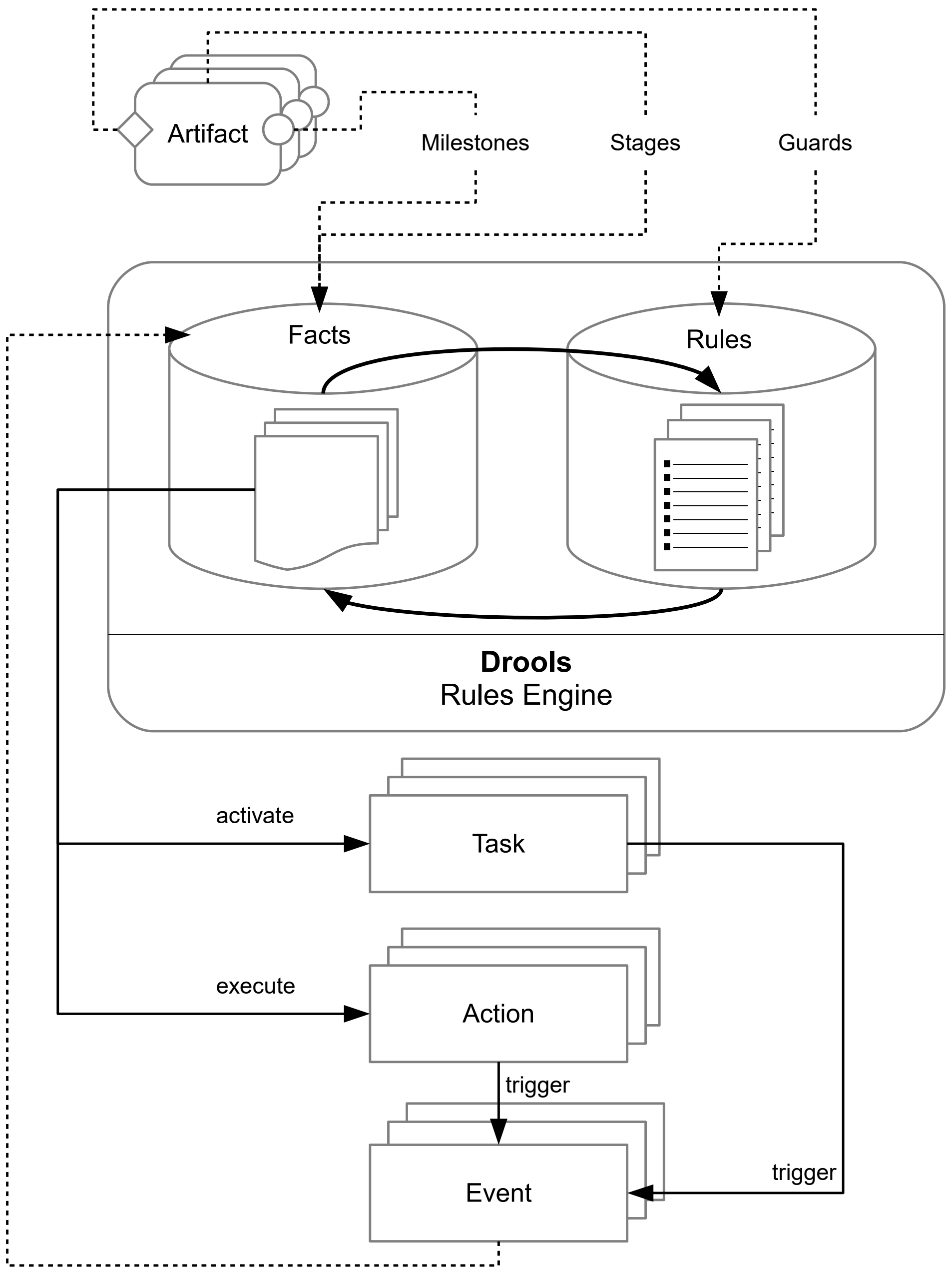


Figure 5.1: Overview of framework for executing artifact-based workflows

Listing 5.1: Drools Rule Examples for a guard enabling and disabling the stage of verification and validation for the conceptual model artifact using the Drools DSL.

```
rule "guard: verification validation conceptual model stage enable"
when
   // get the conceputal model artifact and bind it
   $artifact: ConceptualModel()
   // get the milestone "Complete Model" for the conceptual model artifact and bind it
   $milestone: Milestone( type == "Complete Model", artifact == $artifact )
   // check that the stage "Verifiaction Validation" is not enabled
   not(Stage( type == "Verification Validation", artifact == $artifact ))
then
   // enable stage
   insert(new Stage("Verification Validation", $artifact));
end

rule "guard: verification validation conceptual model stage disable"
when
   // get the conceptual model artifact and bind it
   $artifact: ConceptualModel()
   // check that the milestone "Complete Model" is not set for the conceptual model
   not(Milestone( type == "Complete Model", artifact == $artifact ))
   // get the "Verification Validation" stage of the conceptual model
   $stage: Stage( type == "Verification Validation", artifact == $artifact )
then
   // disable stage
   retract($stage);
end
```

this in a rule engine those entities need to be mapped to entities of the rule engine.

Drools works with a set of rules, which work on a set of facts called knowledge. It also supports events which essentially are translated into facts rules react on. Drools executes rules on facts generating or changing facts, which may trigger more rules until no more rule can be applied.

In Figure 5.1 p. 99 the architecture of the framework is depicted. It shows the mapping between entities with a dashed arrow and interaction with solid arrows. The main component poses the rule engine Drools and its entities *Facts* and *Rules*.

The components of an artifact-based workflow, i.e., artifacts, milestones and stages are represented as facts and guards are converted to rules. Tasks however are separated into two types of tasks. Firstly, there are tasks that are triggered and executed directly using facts or rules and are called *Actions* hereafter. Secondly, there are tasks that are only activated by facts or rules but their execution needs to be explicitly triggered, e.g., by a human. Albeit tasks and actions are part of the artifact-based workflow they are not mapped to entities of the Drools

Listing 5.2: Drools Rule Example for a guard enabling and disabling the stage of verification and validation for the formal model artifact across artifacts using the Drools DSL.

```
rule "guard: verification validation formal model stage enable"
when
   // get the conceptual model artifact
   $conceptualModel: ConceptualModel()
   // get the formal model artifact
   $formalModel: FormalModel()
   // get the "Complete Model" milestone for the formal model
   $milestoneFormalModel: Milestone( type == "Complete Model", artifact == $formalModel )
   // get the "V&V successful" milestone for the conceptual model
   $milestoneVVConceptualModel: Milestone( type == "V&V successful", artifact ==
       $conceptualModel )
   // check that the "Verification Validation" stage for the formal model is not enabled
   not(Stage( type == "Verification Validation", artifact == $formalModel ))
then
   insert(new Stage("Verification Validation", $formalModel));
end

rule "guard: verification validation formal model stage disable"
when
   // get the conceptual model artifact
   $conceptualModel: ConceptualModel()
   // get the formal model artifact
   $formalModel: FormalModel()
   // check that ther milestone "Complete Model" for the formal Model or "V&V successful" for
       the conceptual model is not set
   not(Milestone( type == "Complete Model", artifact == $formalModel )) or
      not(Milestone( type == "V&V successful", artifact == $conceptualModel ))
   // get the "Verification Validation" stage of the formal model
   $stage: Stage( type == "Verification Validation", artifact == $formalModel )
then
   // disable stage
   retract($stage)
end
```

rules engine. They are kept separately.

The transformation from milestones and stages to facts is straightforward, however guards need to be expressed using rules written in the Drools rules API, e.g., using the Drools DSL (domain specific language). Listings 5.1 p. 100 and 5.2 p. 101 illustrate how the guard that enables the *V&V* of the conceptual and formal model respectively is written using the Drools rules DSL. A rule has a name and consists of a *when* and a *then* part. Firstly, the *when* part specifies the pre-condition, that is built using facts, under which the rule can be applied. Secondly, the *then* part specifies what happens when the pre-condition is met, i.e., changing or adding facts. For instance, in Listing 5.1 p. 100 two rules are defined. The first rule manages the activation of the *V&V* stage, while the second rule is responsible for disabling that stage. A stage or milestone is defined as enabled if the matching fact is present and is disabled if the fact is missing. Rule one expects the following facts, i.e., the conceptual model artifact, the *Complete Model* artifact associated to that artifact and that the *V&V* stage is not already enabled (designated by the absence of a matching stage fact). If all conditions hold, a new stage fact is inserted. Rule two disables the *V&V* stage in case the milestone for *Complete Model* is missing (absent fact) but the *V&V* stage is enabled.

For each guard and sentry of the artifact-based workflow rules must be created and for each milestone, stage and artifact facts need to be defined. Drools will then apply rules based on facts and change facts, e.g., enabling and disabling stages and milestones, based on rules. Therefore, Drools is responsible for controlling the life-cycle of the artifacts.

However, it is not responsible for executing actual tasks or actions. This is handled outside of Drools. Here, tasks monitor specific facts, such as a stage or milestone which they require to be executable. For instance, the task involving the verification and validation of a conceptual model will only be executable if the *V&V* stage fact is available, which does not imply it will be executed if that stage is active. A user might decide to execute it later or not at all. However, if tasks are supposed to be executed on specific facts, an action needs to be used instead. Actions are tasks that depend on facts but are executed as soon as the fact holds.

In order to interact with facts tasks and actions can trigger events which are then turned into facts in Drools possibly triggering subsequent rules as a reaction to it. For instance, the V&V task might trigger a V&V-failed or success event, which will trigger the failed or success milestone facts. Additionally, events can also be triggered externally, e.g., as a reaction to changes to the environment.

Putting it all together leads to the process illustrated in Figure 5.2 p. 103. In a nutshell, for a given workflow there are a number of facts as well as a number of rules to begin with. The Drools rules engine will then apply rules on those facts, changing and creating new facts over time. While facts are changing and emerge, tasks and actions can become enabled or disabled depending on those facts. If an action is enabled it will be executed directly, while a task needs to

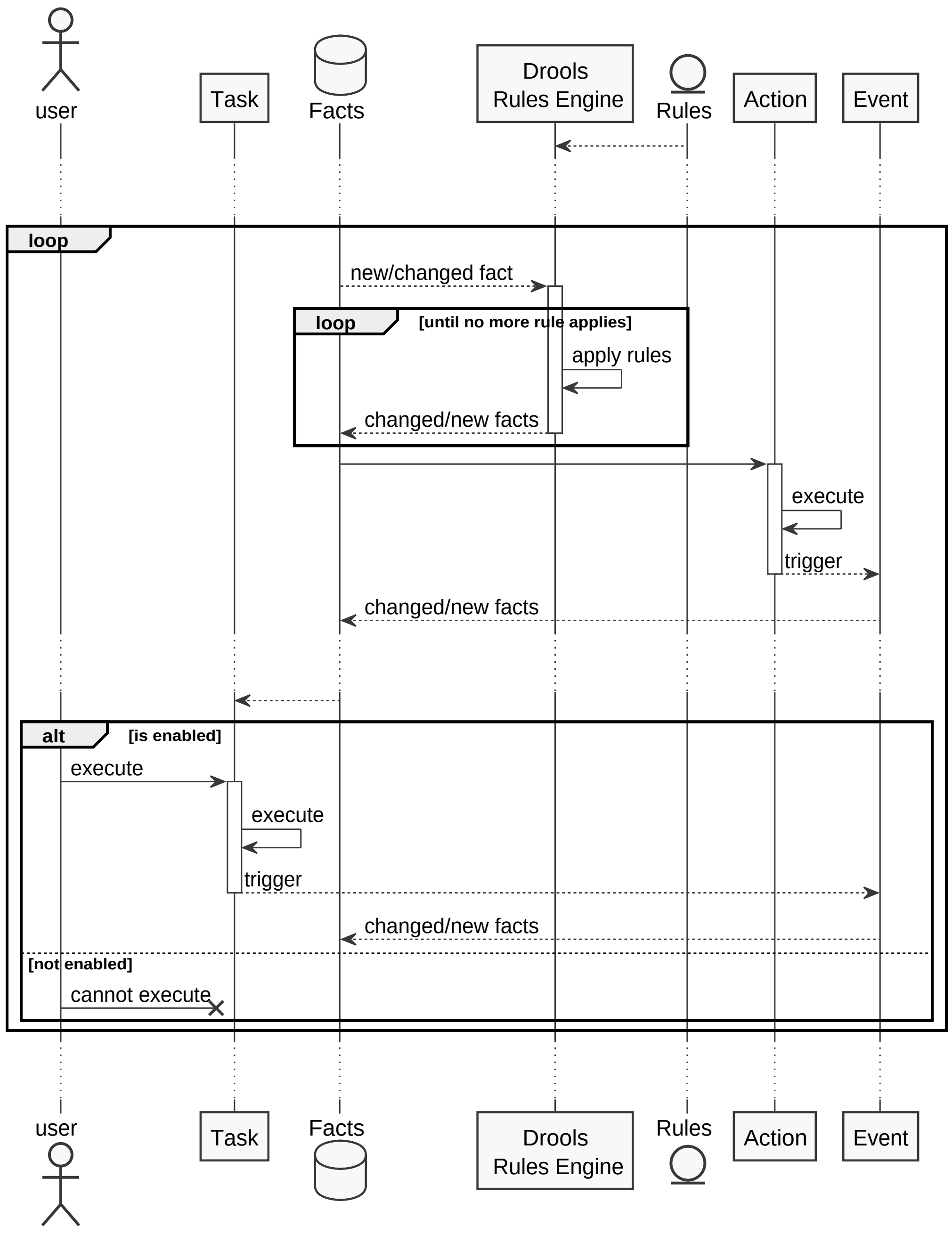


Figure 5.2: Sequence Diagram showing how facts and rules are applied and evaluated and how they interact with *Tasks, Actions* and *Events* in the artifact-based workflow management system

be explicitly executed, e.g., by a user clicking a button in case of a computerized task or executed directly as human task by the user. Both, action and task, might trigger events, such as an event *finished*, which will end up as new or changed facts in the Drools system, leading to a reevaluation of applicable rules and their application.

5.1.2 Integration into JAMES II

When integrating the artifact-based workflow framework into JAMES II, a couple of changes need to be made to JAMES II. There are two areas that need adjusting. Firstly, plug-ins of JAMES II need to be made aware of restrictions that may occur to them based on facts available in the workflow system. This can be achieved by hooking into the plug-in registry and proxying all plug-in requests passing them to an additional filter pass based on the availability of facts. This can be automated. However which plug-in depends on what facts needs to be defined before-hand.

Secondly, the graphical user interface (GUI) of JAMES II needs to be adapted to enable and disable buttons, menus, menu options and so on for specific tasks based on available facts. For instance, a *Validate Experiment* button should only be enabled, if the model artifact that represents the model to execute has the *V&V* stage enabled. Moreover, combining this will lead to an enhanced user experience, directly showing what options of interaction exists at any point in time. Even further, the rules engine and facts can be used to create an execution plan that in return can be used to guide the user through a series of tasks leading, e.g., to a valid and verified model.

Nevertheless, a challenging part of the integration is the actual translation of artifacts, milestones and stages to facts and more importantly guards and sentries to rules. This can either be done automatically or manually, whereas the manual translation can be tedious and error prone. When creating rules, it is important that the plausibility and consistency of artifact guards and sentries is reflected and hold when defining the rules. It needs to be ensured, e.g., that stages cannot be enabled given the defined rules if they cannot be enabled using the artifact based description, thus not violating established contracts.

Moreover, once rules and facts are derived, preferably automatically from the artifact-based workflow, tasks and actions have to be identified and adapted to work with facts and events rather than with the artifacts directly.

5.2 Layer Two — Imperative task-based workflow implementation

In order to support workflows from the domain of Modeling & Simulation standards suggested for general workflow systems need to be considered (Hollingsworth et al., 2004). Also requirements specific to workflows as well as scientific workflows and Modeling & Simulation workflows in particular need to be taken into account, when designing and implementing a workflow system for the use in Modeling & Simulation software (Rybacki et al., 2010).

On the one hand, adapting and using existing systems supporting workflows, e.g., Project Trident (Barga et al., 2008a), Taverna (Hull et al., 2006; Oinn et al., 2006), Kepler (Altintas et al., 2004; Ludäscher et al., 2006) is one way to integrate workflows into Modeling & Simulation software. The benefits are that many functionalities are already provided, although with a focus on features that reflect the original motivation in developing that system. However, this results in the user being restricted to a particular system and its functionality and constraints. This makes it desirable to abstract from a particular system and allow different ones to be used.

On the other hand, a system for supporting Modeling & Simulation workflows can be designed and implemented from scratch. The advantage is that the system can be tailored specifically to the requirements of the domain of Modeling & Simulation and its workflows. However, as argued by Rybacki et al. (2010), supporting workflows in Modeling & Simulation is still in an early stage, leading to a possible refinement, adaptation and extension of those requirements in the future. Thus, the software design should be flexible and extensible. A plug-in-based design, e.g., as used in JAMES II, has already shown that it is suitable for supporting Modeling & Simulation experiments, seamlessly integrating different modeling formalisms, calculation algorithms, and validation methods in a flexible and extensible way (Ewald et al., 2010a).

To provide a comprehensive support for different systems the representation of workflows is of high importance, as it governs the interaction between components. In order to be able to abstract from a specific system a formalism with clear semantics is required to ensure that each system knows exactly how the workflow is to be interpreted. There exist a number of different workflow description options (see Section 2.3 p. 26), that fit this profile. In the area of task-based workflows, High-level Petri Nets such as Workflow Nets constitute as such an option, providing a workflow description which has been well established in the workflow community (Ellis and Nutt, 1993; van der Aalst, 1996a; Van Der Aalst and Van Hee, 2004).

WorMS (Workflows for Modeling and Simulation) follows the second idea of creating a workflow management system from scratch basing managed workflows on workflow nets. It provides a framework to integrate Modeling & Simulation processes defined as workflows into Modeling & Simulation software such as the Modeling & Simulation framework JAMES II (see Section 5.2.3 p. 168) (Rybacki et al., 2010, 2011).

Orchestrating parts of the Modeling & Simulation software using workflows, WorMS is able to provide user guidance based on the workflow at hand as well as enables the automated creation of provenance data for the workflow execution. During workflow execution every step is well-defined and encapsulated into a task. This includes input and output data of tasks as well as the partial order in which tasks can be executed and under which conditions. Having this information available, allows for recording the workflow execution providing provenance data and therefore means to reproduce the workflow execution. The recording itself can be at different levels of detail yielding different levels of reproducibility (see Section 6.1 p. 181).

The advantage of using WorMS and explicitly defined workflows is that the mapped process does not need to be hard coded in the Modeling & Simulation software — it can be adapted and changed to current needs independently without needing to directly change the code of the Modeling & Simulation software. Typically, the workflow is defined at a higher level of process abstraction, however WorMS does not dictate at which level of abstraction a workflow is defined. Processes at a rather low level, like the execution of an experiment (with configurations and replications), are perfectly describable and integrable using WorMS in Modeling & Simulation software (Rybacki et al., 2012a,b).

Integrating WorMS into the backend of Modeling & Simulation software separates the internal control logic of the application from the specialized application functions, leading to a separation of concerns.

In the following the components of WorMS, their role, responsibility, interaction and interplay are presented.

5.2.1 WorMS — A Framework for Workflows in Modeling & Simulation

The basic architecture of WorMS is plug-in-based. Technically this means that it follows the strategy pattern (Gamma et al., 1995). The components present in WorMS are depicted in Figure 5.3 p. 108. It provides extension points where custom components (*strategies*) can be plugged in as needed (colored in blue). Not all components are exchangeable through plug-ins, there are also internal parts responsible for managing, organizing and orchestrating the plug-ins. Additionally, there is the central component of *Workflow Engine* and its internally used workflow format, the *Intermediate Representation* responsible for orchestrating workflow execution and for providing a workflow description with clear semantics. The *Workflow Engine* itself provides extension points for components it interacts with internally, such as the actual *Workflow Executor*, *Data Store* or *Monitoring* are exchangeable by plug-ins which in turn rely on the *Intermediate Representation* to provide concise implementation.

Generally speaking the framework comprises two parts. On the one hand, there are the components responsible for providing plug-ins, workflow repositories as well as security and user management (*Security and User/Role Management*, *Plug-in Provider* and *Workflow Repository*). On the other hand, there are components that are dedicated to the execution a workflow and its documentation (*Workflow Engine* with *Intermediate Representation*, *Administration*, *Analysis*,

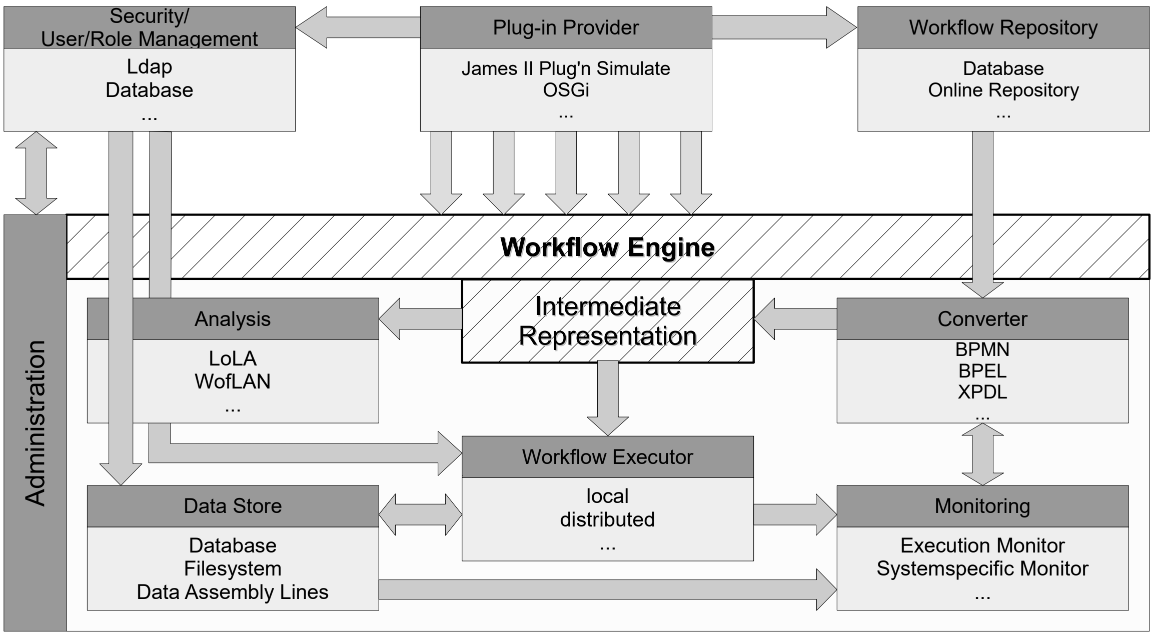


Figure 5.3: Framework Overview. Fixed components are diagonally hatched, exchangeable components are of solid gray (Rybacki et al., 2011).

Data Store, *Workflow Executor*, *Converter* and *Monitoring*).

The components in the first part are not *Workflow Engine* specific and could be reused with different systems. However, the *Workflow Engine* and the *Intermediate Representation*, including its extension points can only be replaced as a whole and there is no guarantee that another system provides the same extensions points or uses the same *Intermediate Representation* for describing workflows. Nevertheless, different system can also be integrated at the level of the *Workflow Executor*.

This implies two strategies when using WorMS in Modeling & Simulation software. Firstly, the extension of the provided *Workflow Engine* by adding, removing and/or exchanging components (plug-ins), such as different *Analysis* Tools or different *Converter* s. Secondly, reusing another existing workflow engine or system and replace the provided *Workflow Engine* altogether.

However, WorMS comes already with a *Workflow Engine* implementation that supports Workflow Nets based workflow descriptions (see Section 5.2.1.1 p. 109), supports different *Data Stores* (Memory-based, persistent, distributed, etc.) , provides *Workflow Executor* implementations that can execute a workflow parallel and distributed (see Section 5.2.1.3 p. 122) as well as different *Monitoring* (performance or documentation) capabilities.

In the following an overview those components, including responsibilities, functionalities and implementation details is presented.

5.2.1.1 Intermediate Representation

As already mentioned, in order to work together seamlessly, the components of the workflow engine operate on a common internal representation of workflows.

Ideally, the workflow representation enables the execution of workflows with clear semantics, the analysis of them as well as the conversion from other representations, such as BPMN or BPEL, as they provide sophisticated workflow modeling and reengineering tools, that could be leveraged this way.

As already stated in Section 2 p. 9, van der Aalst motivates in (van der Aalst et al., 1994; van der Aalst, 1996b) and (Van Der Aalst and Van Hee, 2004) the use of Petri Nets or Workflow Nets respectively for the description of workflows. Other authors also promote the use of Petri Nets and its derivatives to represent workflows (Merz et al., 1995; Oberweis et al., 1997; Russell et al., 2009).

Consequently, WorMS uses workflow nets, which are in fact derived from Petri Nets, allowing using a well known and researched formalism with clear semantics as well as many analysis techniques, that checks for certain properties, e.g., soundness or dead locks (Murata, 1989; Schmidt, 2000; Jensen et al., 2007).

Using workflow nets as *Intermediate Representation* in WorMS does not mean workflows have to be modeled as workflow nets. If there is a viable transformation from another workflow representation to a workflow net, e.g., as shown for BPEL in (Hinz et al., 2005), and there is a *Converter* implementation available, that other representation can be used to model workflows usable in WorMS. This means the modeler can automatically benefit from the corresponding tools for other workflow representation.

The workflow net implementation of WorMS is inspired by Van Der Aalst and Van Hee (2004), but might slightly differ technically. It features the explicit definition of *Input* and *Output* ports for each task, it supports the use of *Sub-*

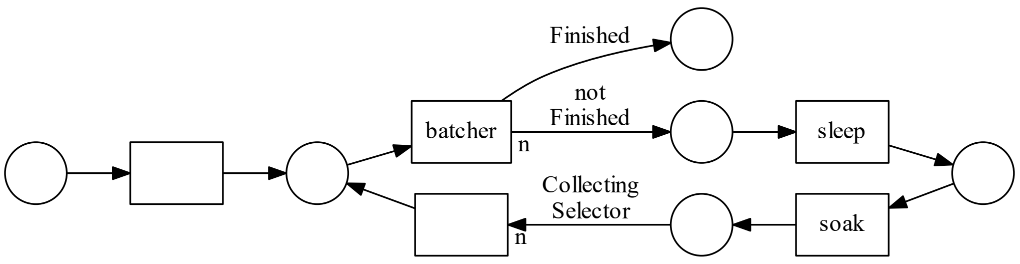


Figure 5.4: Visual Representation of a simple benchmark workflow that has two main tasks, *SleepTask* and *SoakTask*. *SleepTask* takes a time as input specifying how much time this tasks consumes and *SoakTask* stresses one core of a CPU using the PDP-11 soak test ($tan(tan(tan(tan(tan(tan(tan(tan(tan(tan(tan(x)))))))))))$) in combination with a prime number check. Additionally, this workflow allows the execution of a specific number of *SoakTask* and *SleepTask* in parallel as well as iterate over a specific number of batches of parallel executions, all specified using an input token with appropriate colors (data) (see Figure 5.3 p. 111 for how to specify this workflow in WorMS)

Workflows as well as *Template & Frames* (Rybacki et al., 2012b). Furthermore, tokens can be filtered using *Edge conditions* and can be clustered fed to tasks using *Token selectors*. Those features are explained in more detail in the following paragraphs using the sample workflow depicted in Figure 5.4 p. 110 for illustration purposes. At the same time the source code that is needed to create this workflow is presented in Listing 5.3 p. 111 showing how workflows can be described fairly straight forward in WorMS using the internal WorMS Java API.

Input/Output Values WorMS requires the explicit definition of input and output ports for each task in the workflow. An input port is needed for each separate data value (also known as color of a token) Without a specific input port a task cannot access any data from previous tasks or initial data. This is due to the security and efficiency concept employed by WorMS and is handled by the *Data Store* (see Section 5.2.1.4 p. 131).

Listing 5.3: Specification of the benchmark workflow shown in Figure 5.4 p. 110 in WorMS.

```java
public class SimpleBenchmarkModel extends DefaultWorkflowModel {

   public SimpleBenchmarkModel() {
      start = new DefaultPlace("start");
      end = new DefaultPlace("end");
      IPlace tmp = new DefaultPlace("");
      IPlace tmp2 = new DefaultPlace("");
      ITask sleep = new SleepTask(1, true);
      ITask soak = new SoakTask(1, false);
      ITask batcher = new BatchTask();
      ITask collect = new NullTask();

      //adding places to net
      addPlace(start, end, tmp, tmp2);
      //adding Tasks to net
      addTask(sleep, batcher, collect, soak);

      //connecting places and tasks with edges
      addEdge(start, tmp2);
      addEdge(tmp2, batcher);
      //adds an edge only passing tokens that contain finished property
      addEdge(batcher, end, 0, new FinishedCondition());
      //adds an edge only passing tokens that don't have the finished property
      addEdge(batcher, sleep, new NotCondition(new FinishedCondition()));
      addEdge(sleep, soak);
      addEdge(soak, tmp);
      //here an edge is created that has a special selector letting only letting all tokens
          of one batch through at once
      addEdge(tmp, collect, new CollectingTokenSelector());
      addEdge(collect, tmp2);
   }
}
```

Analogously, output ports must be specified for each data value a task produces and should be exchanged with other tasks. Without an output port no data can be injected into the workflow.

If data is produced, there is the option of producing multiple records using the same output values, multiple records each having different output values as well as a combination of both. Records in return are transformed into tokens in the workflow net by the *Workflow Executor* (see Section 5.2.1.3 p. 122). Interestingly, records can be produced asynchronously during the execution of a work item rather than once after the execution is finished. This allows for execution schemes similar to streaming approaches or data-centric approaches such as process networks. Producing records asynchronously, allows processing data as soon as it is available rather than waiting for a task to finish before proceeding with the execution (see Section 2.2.3 p. 19 specifically Figure 2.4 p. 22 for why this is helpful).

Albeit, input and output ports need to be specified when data processing takes place in a task, they do not need to be specified for pass-through purposes. Assuming a scenario involving $Task1$, $Task2$ and $Task3$ consecutively executed ($Task1 \rightarrow Task2 \rightarrow Task3$). $Task1$ produces a data value on output port X and $Task3$ requires a data value at its input port X. However, $Task2$ does no data processing on X, so X basically passes through $Task2$. Now, $Task2$ could define an input and output port for X but does not have to in WorMS, as data passthrough is handled automatically by the *Data Store*. This makes modeling workflows and reusing of tasks a lot easier.

Also, as already mentioned for efficiency and security purposes input data is not directly provided to the task (or work item for that matter) on execution, but is rather provided lazily. Access to data is handled using access tokens that are issued by the *Data Store* for each execution of a task. With an access token a work item can request data specified using input ports directly from the *Data Store*. This has the advantage that data that is not needed (requested) does not have to be loaded from the *Data Store*. It also allows the *Data Store* to control which data is accessible via the access token and if necessary can deny access to wrongly or unauthorized requested data, e.g., data no input port is specified for or by a role that has no privilege to access that data (see Section 5.2.1.4 p. 131 and Section 5.2.1.6 p. 140).

Tasks In workflow nets tasks equalize the transitions in the underlying petri net. They build the foundation of each defined workflow in WorMS, while places and edges (arcs) are connecting tasks with each other semantically (parallelly, sequentially, iteratively and alternatively).

Contrary to other WfMSs WorMS is not web-service oriented and does not require task to be represented by web-services, however they can represent web-service invocations if desired. An advantage of not relying on web-services, is that workflows are still executable in a local non-connected environment and by being able to avoid the web-service overhead (connection as well as invocation) the execution can be more efficient, particularly when the workflow is executed completely locally with no distribution (see Section 5.2.1.3 p. 124) within the same Virtual Machine (VM). Additionally, workflow defintions suffer from decay over time which is amplified when depending on external resources such as web-services (Zhao et al., 2012; Hettne et al., 2012).

Consequently, WorMS comes with a Java API for the definition of tasks implementing the task to execute directly in Java. Listing 5.4 p. 114 shows a very basic task implementation, using an `AbstractTask` base class provided by WorMS, which makes defining tasks very easy. Basically, a task provides input and output ports (which are omitted in Listing 5.4 p. 114) as well as an implementation that provides an `IWorkItem` instance. The work item does the actual work of the task in the context of the calling running workflow (see Section 2.1 p. 9). Listing 5.5 p. 114 shows the associated work item implementation for `DummyTask`. The `DummyTask` also relies on a base class provided by WorMS named `AbstractWorkItem`, which already implements most of the administrative functionally of a work item, such as data listener management, event handling and notification of progress and data to the *Workflow Executor*. The things that need to be provided by an actual work item implementation is the actual work to be performed (`internalRun`), cancellation handling (`cancel`, `isCancelled`) as well as providing data to each output port on demand (`getOutput`).

In order to showcase input/output port, including data output as well as asynchronous data output, the `DummyTask` and `DummyWorkItem` are extended and shown in Listing 5.6 p. 115 and 5.7 p. 115. The task was extended by the declaration of an output port named `COUNT` of type `int`. The work item now fire additionally to the progress event a data generated event (`fireDataAvailable`) during execution. Consequently, it now also provides an implementation for providing data values on the output port `COUNT`.

This shows that it is fairly straight forward to specify tasks and their work items in WorMS using its Java API.

Sub-Workflows Another feature present in workflow nets is the concept of sub-workflows. Sub-workflows can be used to structure complex workflows into reusable parts, easier to maintain and extend, separate concerns by isolating

Listing 5.4: Sample Task implementation. This task does not specify any input or output ports and simply provides a work item implementation showin in Listing 5.5 p. 114

```java
public class DummyTask extends AbstractTask {

   @Override
   protected IWorkItem create(List<Map<String, Object>> input) {
      return new DummyWorkItem(this);
   }
}
```

Listing 5.5: Work Item implementation for the sample task shown in Listing 5.4 p. 114. The work item's job is to count to 10 and report its progress after each count.

```java
class DummyWorkItem extends AbstractWorkItem {

   public DummyWorkItem(ITask parent) {
      super(parent, "Dummy Name");
   }

   @Override
   protected void internalRun() {
      final int maxCount=10;
      for (int i=0;i<maxCount;i++) {
         //update progress information
         fireProgress((float)i/maxCount, String.format("%d of %d", i, maxCount));
      }
   }

   @Override
   public void cancel() {
      //this is called when the current execution should be cancelled
   }

   @Override
   public boolean isCanceled() {
      //cancellation is not implemented for this work item so return false
      return false;
   }

   @Override
   public <E> E getOutputValue(long datasetId, String name, Class<E> type) {
      //no actual data is produced from this work item
      return null;
   }
}
```

Listing 5.6: Adapting the Sample Task (see Listing 5.4 p. 114) to support streamed data output.

```java
public class DummyTask extends AbstractTask {
   public static final String COUNT="COUNT";
   public static final Class<?> COUNT_TYPE=Integer.class;

   public DummyTask() {
      Map<String, Class<?>> outputMap=new HashMap<>();
      //adding COUNT data to output
      outputMap.put(COUNT, COUNT_TYPE);
      setOutput(outputMap);
   }

   @Override
   protected IWorkItem create(List<Map<String, Object>> input) {
      return new DummyWorkItem(this);
   }
}
```

Listing 5.7: Work Item implementation for the adapted sample task shown in Listing 5.6 p. 115

```java
class DummyWorkItem extends AbstractWorkItem {

   public DummyWorkItem(ITask parent) {
      super(parent, "Dummy Name");
   }

   @Override
   protected void internalRun() {
      final int maxCount=10;
      for (int i=0;i<maxCount;i++) {
         //update progress information
         fireProgress((float)i/maxCount, String.format("%d of %d", i, maxCount));
         //let the data listener know that there is 1 new data token identified by i
         fireDataAvailable(i,1);
      }
   }

   @Override
   public void cancel() {
      //this is called when the current execution should be cancelled
   }

   @Override
   public boolean isCanceled() {
      //cancellation is not implemented for this work item so return false
      return false;
   }

   @Override
   public <E> E getOutputValue(long datasetId, String name, Class<E> type) {
      //check which data is requested and if it is type compatible
      if (DummyTask.COUNT.equals("name") && type!=null &&
         type.isAssignableFrom(DummyTask.COUNT_TYPE)) {
         //for simplicity we return the id as it represents the count index in this example
         return (E) Integer.valueOf((int)datasetId);
      }
      return null;
   }
}
```

business logic in extra sub-workflows. For instance, the simple benchmark workflow shown in Figure 5.4 p. 110 should be integrated into another workflow, it can simply be added as sub-workflow, acting as ordinary task within the parent workflow, as shown in Figure 5.5 p. 116.

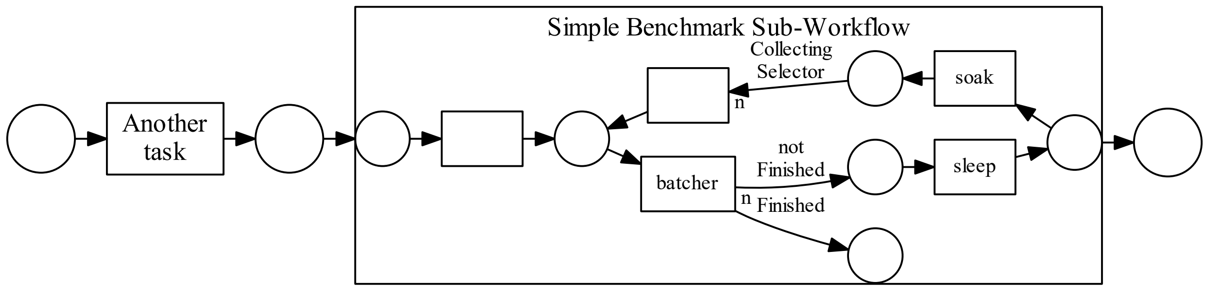


Figure 5.5: Workflow integrating the simple benchmark workflow from Figure 5.4 p. 110 as sub-workflow

Sub-workflows are also fairly easily set up using the WorMS Java API, e.g., in Listing 5.8 p. 117 the sample workflow shown in Figure 5.5 p. 116 is described. Please refer to line 9 p. 117, where a task is added that is represented by a sub-workflow (`SimpleBenchmarkModel`) which are defined earlier (see Listing 5.3 p. 111) by simply calling `addSubWorkflowTask`. After that, the return `ITask` encapsulating the sub-workflow can be handled as it was an ordinary task when constructing the workflow further (see line 18 and 19). It automatically inherits the input and output ports derived from the sub-workflow and handles token hand-over from and to parent workflow including data requests from and to the *Data Store*.

Interestingly, while it seems the sub-workflow concept is only used to structure a complex workflow it also serves as means to simplify workflows that need to describe a sub-process that can be performed multiple times in parallel. This is achieved by the fact, that sub-workflows are executed in isolation in WorMS, meaning that they are treated the same way as ordinary tasks when they are executed multiple times in parallel. An ordinary task creates a new work item instance whenever executed, which allows for a parallel execution of the same task without interference. The same semantic is applied to the execution of a sub-

Listing 5.8: Description of workflow example shown in Figure 5.5 p. 116

```java
public class SampleModel extends DefaultWorkflowModel {

   public SampleModel() {
      start = new DefaultPlace();
      end = new DefaultPlace();
      IPlace tmp = new DefaultPlace("");
      ITask anotherTask = new AnotherTask();
      //adding a template task represented by another workflow model as sub-workflow frame
      ITask benchmarkTask = addSubWorkflowTask(new SimpleBenchmarkModel());

      //adding places to net
      addPlace(start, end, tmp);
      //adding Tasks to net
      addTask(anotherTask);

      //connecting places and tasks with edges
      addEdge(start, anotherTask);
      addEdge(anotherTask, benchmarkTask);
      addEdge(benchmarkTask, end);
   }
}
```

workflow task, which means the state of a sub-workflow is not shared between executions.

For instance, assuming the workflow shown in Figure 5.6 p. 117 is executed and having the shown token state. This means, that the simple benchmark sub-workflow task will be executed twice in parallel. What happens is that there will be two instances of that benchmark workflow each of which handling one of the input tokens.

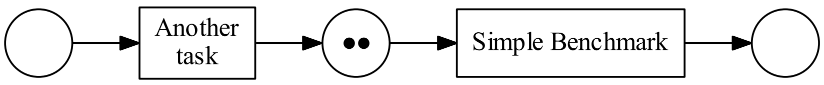
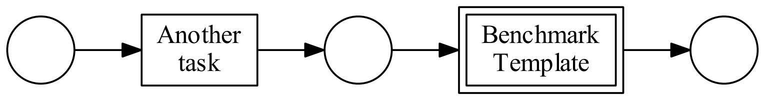


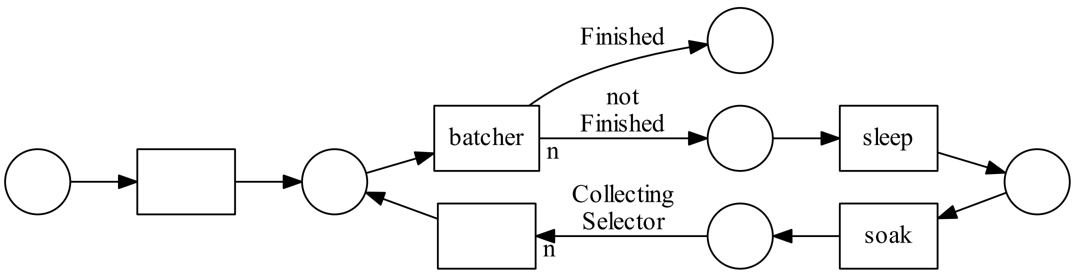
Figure 5.6: Workflow integrating the simple benchmark workflow from Figure 5.4 p. 110 as sub-workflow

Template and Frames Template and frames are introduced as mean to add flexibility to imperative workflows (see Section 2.4.2.1 p. 37). They are not part of the workflow net definition. Other than the sub-workflow approach which changes the workflow structurally, the Template and Frames approach introduces template (placeholder) tasks, which are evaluated and exchanged by actual tasks (wrapped in Frames) during runtime.

Exchange can happen based on execution history of the process until the template task is reached, as well as based on initial configuration, which might define the actual task to be used for a specific template task, or it can also be selected by a user if there are multiple matching tasks for a template task.



(a) Workflow integrating a template task



(b) Simple Benchmark Workflow usable as Frame

Figure 5.7: Example template-based workflow

Templates are typically used in situations where there are multiple options for a task which might change over time in a process which otherwise would have to be modeled explicitly (see Figure 2.14(a) p. 38).

Another example is shown in Figure 5.7 p. 118 which shows the sample workflow integrating a benchmark workflow as sub-workflow, changed to use a template task for the benchmark task instead. The simple benchmark workflow is then provided as Frame. Listing 5.9 p. 119 shows the necessary source code to create such a workflow including the frame representing the simple workflow benchmark using the WorMS Java API. As it can be seen, the code is not much different from using the sub-workflow approach. However, the advantage is that there can be a multitude of frames per template without the need to change the

Listing 5.9: Description of workflow example shown in Figure 5.5 p. 116

```
 1  public class SampleModel extends DefaultWorkflowModel {
 2
 3     public SampleModel() {
 4        start = new DefaultPlace();
 5        end = new DefaultPlace();
 6        IPlace tmp = new DefaultPlace("");
 7        ITask anotherTask = new AnotherTask();
 8        //adding a task represented by another workflow model as sub-workflow
 9        ITask benchmarkTask = addTemplateTask(SimpleFrameType.INSTANCE,
              GUIFrameSelector.INSTANCE, new SubworkflowFrame(SimpleFrameType.INSTANCE,new
              SimpleBenchmarkModel()));
10
11        //adding places to net
12        addPlace(start, end, tmp);
13        //adding Tasks to net
14        addTask(anotherTask);
15
16        //connecting places and tasks with edges
17        addEdge(start, anotherTask);
18        addEdge(anotherTask, benchmarkTask);
19        addEdge(benchmarkTask, end);
20     }
21  }
```

workflow model, because frames can be added and removed independently from the workflow model.

Technically, a template task defines a `IFrameType` which defines the interface (input and output ports) of the template task, which in turn must be met by the provided frames. Frames are automatically matched based on the interface (input and output port-wise) they provide. Additionally, a frame can filter itself based on actual values on input ports. This allows for instance to provide a number of analysis methods as frames for an analysis template task, without the need to specify under which conditions which frame is available beforehand as a frame can determine applicability e.g., by checking input distributions or initial specifications.

Frames are provided by an `IFrameProvider` and an actual frame is selected during runtime using an `IFrameSelector`, which can be a simple pop-up dialog asking a user to select a frame or can be automatic selectors based on a policy, e.g., a machine learning policy.

Once a frame type for a template and a set of frames for that template is defined it can simply be added using `addTemplateTask` providing the frame type, a selector and the available frames (see Listing 5.9 p. 119 line 9). Similar to a sub-workflow task, a template task can after being added used like regular task within the workflow model. In this example, a static set based internal frame provider is used fixing the available frames to the frames used during definition in line 9. However, a separate frame provider can be specified instead in order to e.g., provide a dynamic supply of frames based on plug-ins.

Edge Conditions and Token Selectors Edge conditions are used on outgoing edges from tasks in order to control flow of produced tokens, e.g., implementing a (X)OR-fork. Tokens only pass if the edge condition holds. Token selectors are used on incoming edges for tasks and provide the ability to consume more than one token at once even selecting specific token combinations, e.g., select all tokens having the same value for field *name*. The security aspect already discussed for tasks is also followed for conditions and selectors. Both need to define input ports and only values from those input ports can be accessed from the *Data Store*.

For instance, in the simple benchmark workflow (see Figure 5.4 p. 110) there are two edge condition and one token selector applied. One edge condition is found on the edge between *batcher* task and end place and the other between *batcher* task and place leading to *sleep* task. Both conditions complement each other, meaning if one condition is false for a token (not letting it pass) the other is true (letting it pass). They are used to controlling the flow between the batch iteration and finished paths taking a *finished* input value as condition. See Listing 5.3 p. 111 line 22 for the condition used on the path to the end place and line 24 used on the alternative path, simply negating the finished condition (see Listing 5.10 p. 121 for an example implementation).

The token selector is found on the edge labeled *Collecting Selector* and does exactly what it says (see Listing 5.3 p. 111 line 28). As can be seen the batcher task produced n tokens (parallel tokens) that pass through *sleep* and *soak* and the collection selector refuses to forward any token as long as there are all n tokens available and then forwards them at once. It functions as a AND-join in this case.

Listing 5.10: Implementaiton example of the `FinishedCondition` used in Listing 5.3 p. 111

```java
public class FinishedCondition extends AbstractCondition implements ICondition {

  public FinishedCondition() {
    //add input port in order to access the number of batches left provided by the output port
        of the batcher task
    addInput(BatchTask.BATCHES_LEFT, Number.class, -1);
  }

  @Override
  public boolean check(Map<String, Object> input) {
    //if there are no more batches left return true
    return ((Number) input.get(BatchTask.BATCHES_LEFT)).longValue() < 0;
  }
}
```

5.2.1.2 Workflow Engine

The *Workflow Engine* is the central component orchestrating the execution of a workflows in WorMS. The organization includes the scheduling of workflows, triggering and managing the execution of workflows. It also presents the central place when it comes to attaching monitors or other workflow observers and provides information about workflow states, such as currently running, finished or scheduled.

However, the *Workflow Engine* does not execute a workflow it uses a *Workflow Executor*, either specified when scheduling a workflow or the standard executor, in order to execute a workflow. Nevertheless, it monitors, observes the workflow execution and handles *Workflow Executor* and associated workflow, *Data Store* and *Security and User/Role Management* policy.

Listing 5.11 p. 122 shows how a workflow is scheduled using the default security policy (which does not restrict any action). The returned `workflowID` is used to identify the scheduled workflow throughout the workflows life-cycle. For instance, as seen in Listing 5.12 p. 122 line 21 the `workflowID` is used to start the previously scheduled workflow using the `BasicMemoryDataStore` and the initial token mapping defined in `map`.

Once a workflow is finished executing its end state will be provided by the *Workflow Engine*, allowing access to tokens, in e.g., the end place, and the *Data Store* in order to access the token's associated data.

Listing 5.11: Adding of a workflow to the *Workflow Engine* for later execution.

```java

//create the simple benchmark workflow model
IWorkflowModel benchmarkModel = new SimpleBenchmarkModel();

//add the workflow to the engine and get a workflow ID for later reference
String workflowID = WorkflowEngine.scheduleWorkflow(benchmarkModel,
   DefaultSecurityModel.ALLALLOWEDINSTANCE, false);
```

Listing 5.12: Executing the previously added workflow using a specific *Data Store*, *Security and User/Role Management* policy and initial token mapping

```java
//map represents a token and its data
Map<String, Object> map = new LinkedHashMap<>();

//set data of start token
map.put(BatchTask.BATCHES_LEFT, 5);
map.put(BatchTask.PARALLEL, 20);
map.put(SleepTask.SLEEP, 250);
map.put(SleepTask.DATA, "..."); //some big data to be added here
map.put(SoakTask.SOAK, 25);

//using the basic memory backed data store implementation
IDataStore dataStore = new BasicMemoryDataStore();

Pair<Map<IPlace, List<IToken>>, IReadOnlyDataStore> resultTokenMapping =
    WorkflowEngine.start(workflowID, new SimpleDataProvider(DefaultRole.STANDARD, workflowID,
    new Pair<>(benchmarkModel.getStart(), map)), dataStore);
```

5.2.1.3 Workflow Executor

The *Workflow Executor* is another central component of WorMS, responsible for the actual execution of a workflow. It orchestrates the flow of data (tokens) through the workflow involving activities and places as well as other components such as *Security and User/Role Management* and *Data Store*.

Ultimately, the core of the *Workflow Executor* is a workflow net simulator, which determines activities that are executeable at a given point in time, also handling token consumption and production respectively. The execution of activities responds to the firing of transitions in the workflow net, which triggers the instanciation of an activity into a work item which in return is then executed. Before a transition can fire all token selectors (see Paragraph 5.2.1.1 p. 120) must be fulfilled.

Figure 5.8 p. 123 shows a simplified procedure the *Workflow Executor* applies, omitting the determination of firable transitions and starts with one selected firable transition and associated activity. It then invokes `create` on the asso-

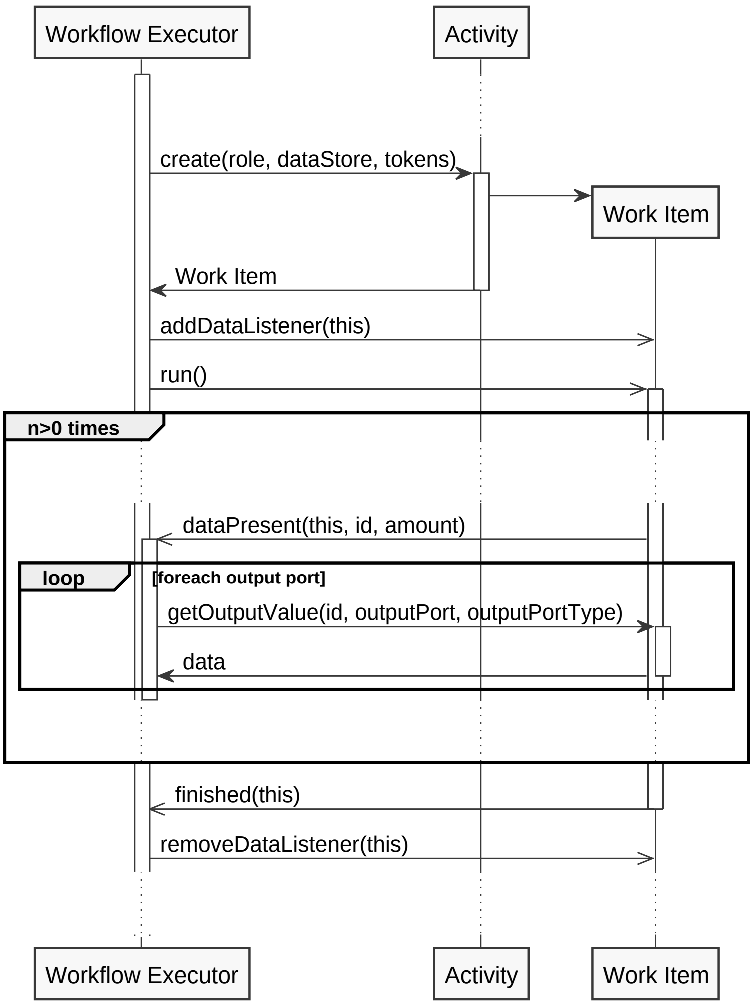


Figure 5.8: Execution of an Activity by the *Workflow Executor* assuming the default security policy, which does not restrict the execution of activities by a specific role.

ciated activity resulting in a work item which is executed using its `run` method after the *Workflow Executor* registers itself as data listener. During its execution, the work item may produce data, which is received by the *Workflow Executor* and is handed over to the *Data Store* associated with the current execution. The *Data Store* in return manages the data and captures it in a token which is then placed into the workflow net for further processing. Data might be produced multiple times at any time during a single execution of a work item but at least once, ensuring the production of at least one token after execution of a work item. After the work item is finished executing it sends out a notification which is processed by the *Workflow Executor*.

The special feature of the workflow net simulator used by the *Workflow Executor* is that transitions and therefore activities can fire multiple times concurrently. This is can happen if transitions are still fireable, after the consumption of tokens from the previous transition firing or become fireable during the execution of the activity associated with the transition (see Figure 5.9 p. 125). The simulation of the workflow net does not pause during the firing of a transition. This ensures that as long as there are resources fireable transitions are executed leading to an automatic parallel execution schema.

Parallel and Distributed Execution of Activities in WorMS Technically the *Workflow Executor* does not invoke the `create`, `addDataListener`, `removeDataListener`, `run` or `getOutputValue` methods of an activity as shown in the simplified Figure 5.8 p. 123. They are invoked, by one of the *Workflow Executor*'s workers. Workers allow the *Workflow Executor* to process the workflow in parallel by delegating the actual execution, data monitoring and work item instantiation.

There is the distinction between parallelism achieved within one system and VM (local) and parallel and distributed execution across systems and VMs (distributed). From a technical point of view different techniques exist to achieve parallel and distributed execution, e.g., threads, Message Passing Interface (MPI), Remote Method Invocation (RMI) and Akka (Walker, 1994; Downing, 1998; Gupta, 2012).

Threads Threads can be used to enable concurrent execution locally within a Java VM. Distributed execution (across machines) cannot be achieved using threads. However, since everything is run within the same VM interaction, communication and access of data between threads is directly supported and fast.

If data needs to be exchanged between threads no serialization and deserialization is needed making it fast but also error-prone as the concurrent implementation using threads needs to take care of synchronization and

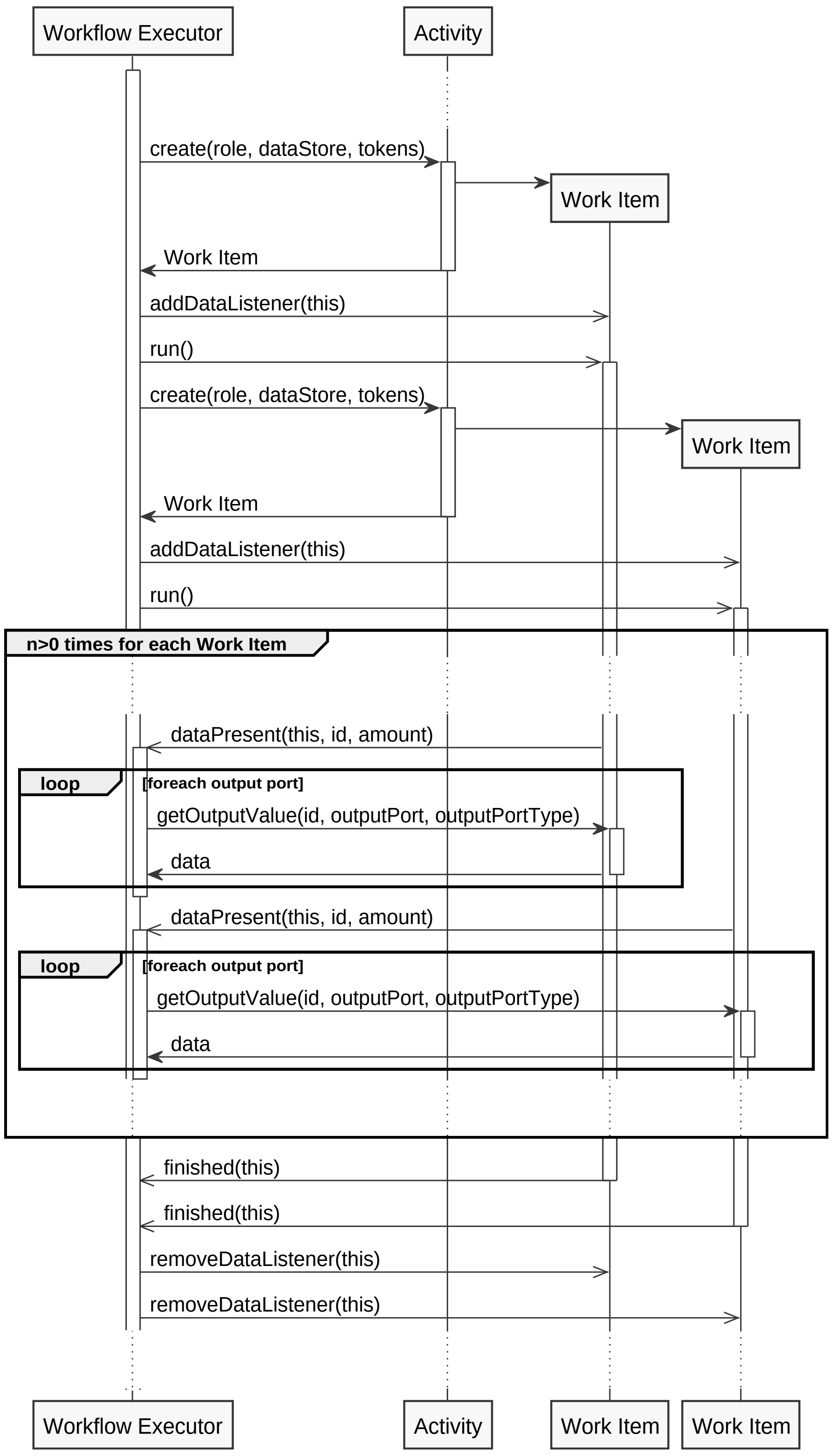
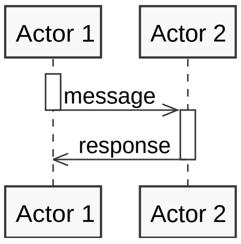
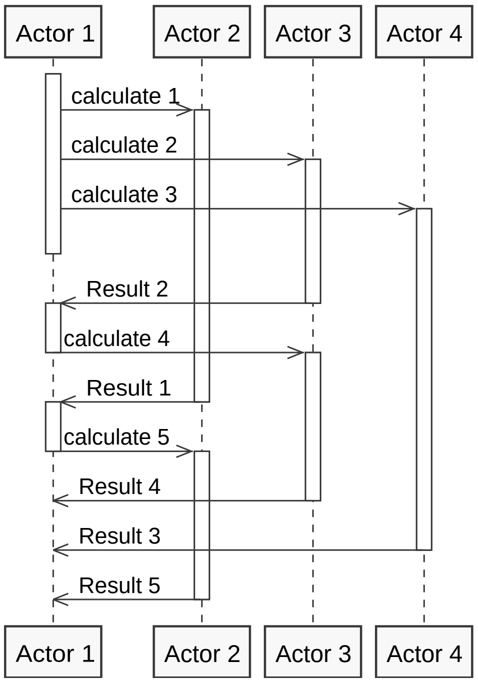


Figure 5.9: Execution of an Activity twice simultaneously by the *Workflow Executor*



(a) Principle of the Actor Concurrency Model



(b) Example calculation using actor concurrency model

Figure 5.10: Actor Concurrency Model

parallel data access. Moreover, a thread-based implementation is memory-bound and also bound by the maximum threads that can be created on a single machine.

RMI In contrast to threads Remote Method Invocation or RMI can be used to create a distributed execution environment. It allows the invocation of methods and the retrieval of data from distant distributed instances running on different machines.

However, parallelism is not directly supported but can be achieved using threads locally on each of the connected machines. RMI simplifies the management, creation and deletion of data across multiple connected machines providing a distributed garbage collection. Nevertheless, the same restrictions apply when only using threads, synchronization of threads and data access needs to be handled and might also run into memory and thread limits. On top, data needs to be serialized and deserialized when accessed across machines.

MPI description MPI or Message Passing Interface is a portable language-independent protocol used to program parallel computers. It is designed to support the creation of parallel and distributed execution environments. Communication between machines is handled using messages, hence Message Passing Interface. Here MPI is focused on using global messages only when necessary, falling back to internal communication if possible. MPI can be used to implement concurrency schemes such as the actor concurrency model used by Akka, however a lot of special cases that are already abstracted by Akka, such as synchronization and threads need to be provided for MPI.

Akka Akka is an open source library for building concurrent, distributed and fault-tolerant systems. Akka is written in Scala, but also offers a Java API. Getting concurrency right is intrinsically difficult and concurrency with threads is even more difficult. Akka addresses this by using the actor concurrency model introduced in Erlang (Agha, 1986). In principle it is based on asynchronous message passing (similarly to MPI) and immutable state,

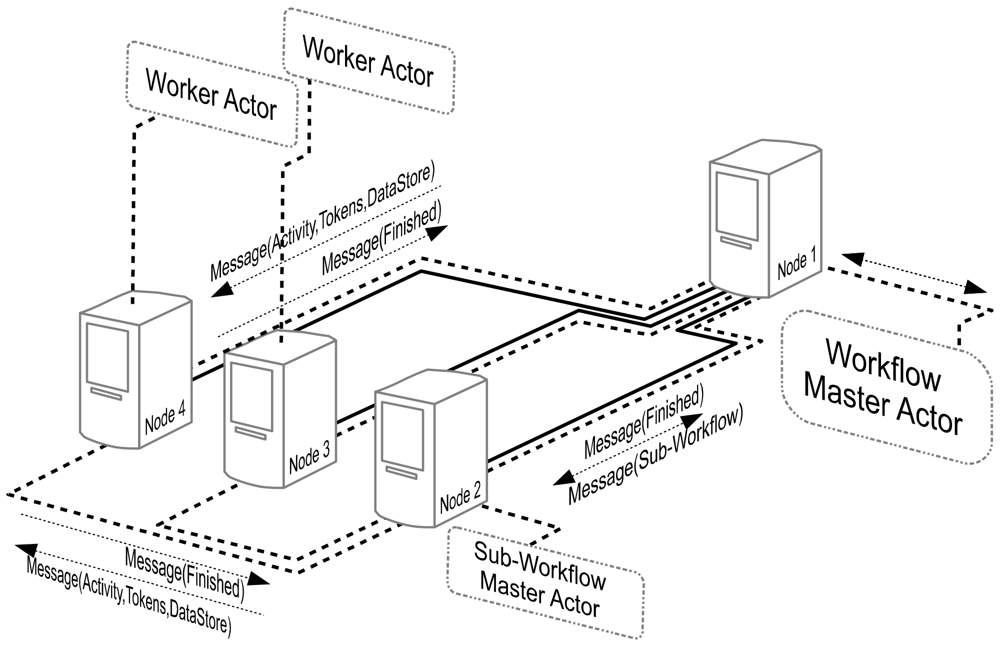
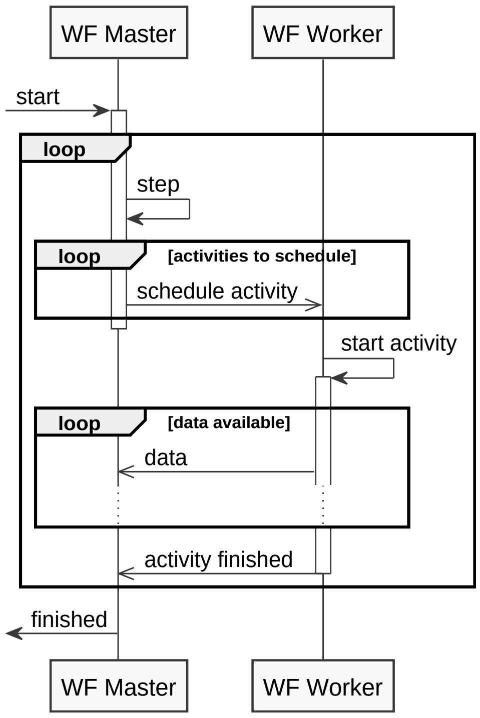


Figure 5.11: Parallel and distributed execution of a workflow in WorMS using Akka.Messages are used to transfer data between nodes (workers). For each workflow instance there exists a *Workflow Master Actor* and a number of *Worker Actors* and one *Sub-Workflow Master Actor* per sub-workflow.

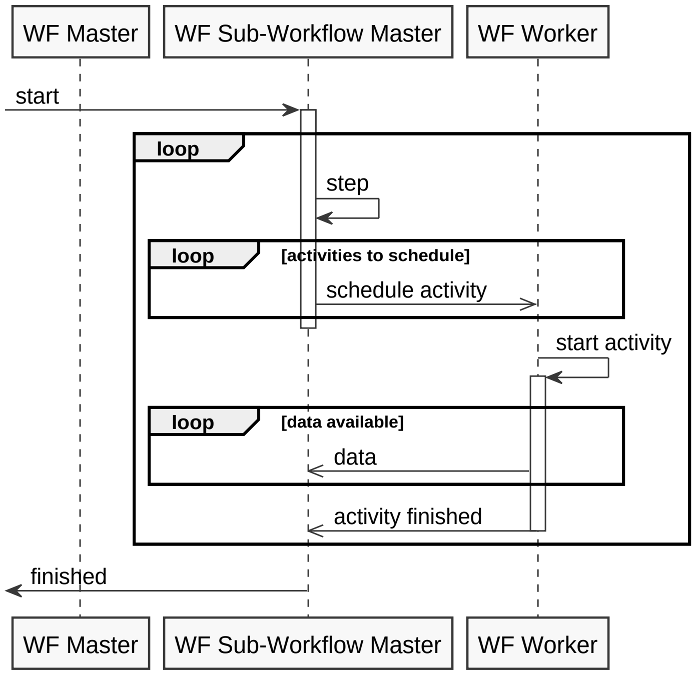
providing a higher level of abstraction than threads. Messages are passed between actors and only one message is handled at a time by each actor. New messages are delivered into a mailbox. After a message is processed an answer message may be sent and the next message from the mailbox is processed (see Figure 5.10(a) p. 126).

However, not all concurrent problems can be easily transformed to the actor model. To address this, Akka also provides Software Transactional Memory (STM), allowing the modification of many objects at once transactionally. Moreover, Akka provides other concurrency abstractions, e.g., Agents and Dataflow Concurrency.

Akka is used in WorMS and is responsible for the actual execution of activities in a concurrent and distributed manner. In fact it is also used for distributed *Data Stores* (see Section 5.2.1.4 p. 131). For the implementation the



(a) without work items that represent sub-workflows



(b) with work items representing sub-workflows

Figure 5.12: Interplay of actors responsible for workflow execution in WorMS

Workflow Executor uses three types of akka-actors are defined. Firstly, the Workflow Master Actor, which is responsible for executing or simulating the top-level workflow net. There is exactly one master actor for each workflow that is executed. Secondly, there is the Sub-Workflow Master Actor, which is responsible for handling a sub-workflow. Managing sub-workflows involves the simulation of the workflow net contained in the sub-workflow as well as communication (token exchange from and to the sub-workflow) with the parent workflow, i.e., another sub-workflow master actor or the workflow master actor. If there are sub-workflows defined in the workflow to execute there is at least one sub-workflow actor per workflow execution. And lastly, there is the Worker Actor, which sole responsibility is the actual execution of an activity including the instantiation of the associated work item which is then executed. The more worker actors are available to WorMS and therefore to the *Workflow Executor* the more activities can be performed concurrently. Except from the workflow master actor, the sub-workflow master and worker actors can be distributed across machines (see Figure 5.11 p. 128). Important to note is that there can be an arbitrary number of worker and sub-workflow master actors on the same machine and worker actors are optionally shared by workflow master and sub-workflow master actors across workflow executions.

The typical execution procedure of a workflow without involving sub-workflows is depicted in Figure 5.12(a) p. 129. The workflow master actor (WF Master) starts the workflow execution and simulates one step in the workflow net given the initial token markings and data. After this step a transition that can fire is determined and its associated activity and consumed tokens, which refer to the activities input data, are sent via an actor message to a worker actor (WF Worker). Which worker actor is chosen depends on the worker selection policy (see Section 5.2.2 p. 147 for more information on selection policies). This is repeated until no more transition is able to fire. Meanwhile, the receiving workflow worker starts to execute one scheduled activity at a time, by instantiating a work item which is then executed. Produced data during execution is send back to the master actor that scheduled the activity where it is turned into a token by the *Data Store* and put back into the master actor's workflow net simulation. Once the work item finishes the workflow master is also notified and the worker actor proceeds with the next scheduled activity. Whenever a master actor receives data or a finished message it performs another simulation step of the workflow net and starts over with scheduling firable transitions.

In case an activity is represented as a sub-workflow it will not be scheduled for execution by a worker actor but a sub-workflow master actor. The sub-workflow master actor then executes the sub-workflow similarily to the master actor with the difference that it also manages token and data flow from parent workflow to the sub-workflow as well as propargating sub-workflow events such as the finished event to the parent workflow (see Figure 5.12(b) p. 129).

5.2.1.4 Data Store

The execution of workflows involves the passing of tokens as well as possible data attached to them between work items. Each work item consumes at least one token and produces at least one token, with or without additional data attached. The number of tokens produced during a workflow execution and the data associated highly depends on the workflow and the work items involved. Thereby, the amount of data and number of tokens produced can be very high, albeit the number of active tokens (tokens that are not yet consumed) is usually smaller. For instance, referring to Figure 5.13 p. 134 during execution a total of ten tokens is produced, however at any given time there are at most two active tokens.

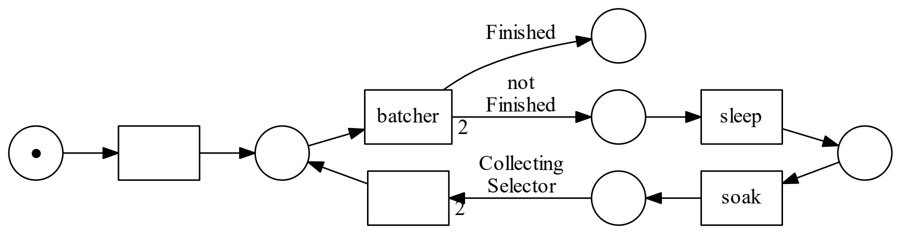
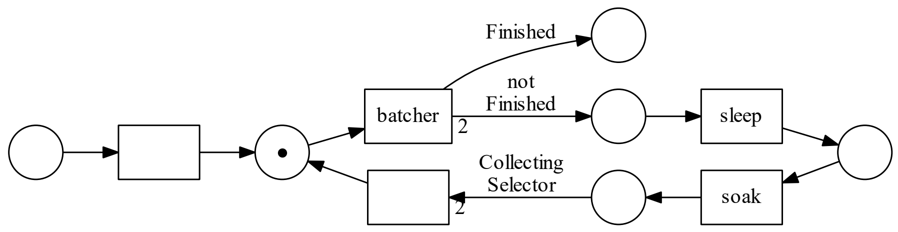
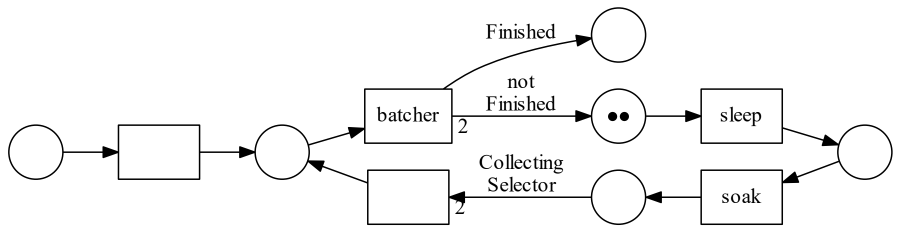
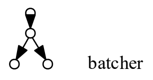
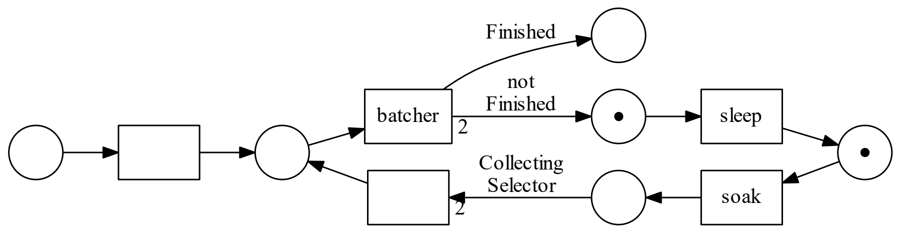
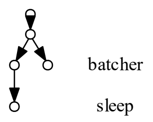
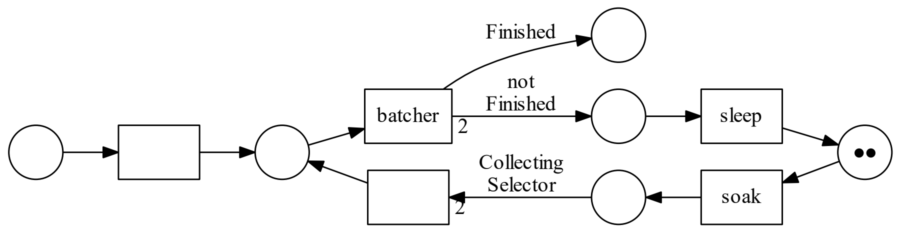
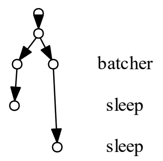
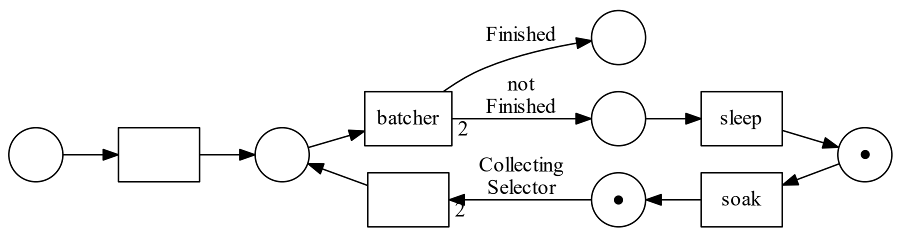
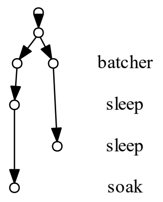
There are two places where tokens are managed. Firstly, as shown in Section 5.2.1.3 p. 122 the *Workflow Executor* orchestrates when tokens are created and when they are consumed as well as what data to attach to tokens. The position (place) of each active token is managed here, too. The *Workflow Executor* only manages active tokens. Secondly, tokens are also managed by the *Data Store*. It is responsible for creating tokens for the use by the *Workflow Executor*, storing and retrieving data for tokens and ensuring that security policies are followed when storing and retrieving data. The *Data Store*, in combination with the *Security and User/Role Management* component has to provide only the information which is actually accessible for a specific work item and a specific user to ensure consistency and security.

The amount and type of data to be stored varies from workflow to workflow, therefore data management must be generic and robust. For instance, implementation can be based on e.g., Data Assembly Lines (Zinn et al., 2009), databases (rational, document- or object-based) or be purely in-memory stores.

In general, the *Data Store* has three tasks. Firstly, create and manage tokens. Secondly, store data associated with a specific token. Thirdly, retrieve data associated with a specific token.

Herein, the first two tasks are required by the *Workflow Executor* and the third task is required by work items, edge conditions and token selectors. In order to ensure that only specific components can read or write data the *Data Store* introduces two proxies namely a `IReadOnlyDataStore` and a `IWriteOnlyDataStore`, whereas the read only data store is provided to, e.g., a work item and the write only data store provided to, e.g., the *Workflow Executor*.

Internally, whenever a token is created the tokens that were consumed by the work item that produced this token in the workflow net are linked as parent tokens. This is important firstly, to provide a token history allowing for each token to trace its creation (provenance). Secondly, this is also important to provide the ability of not having to specify pass-through input and output ports for work items as described in Section 5.2.1.1 p. 110. Instead of appending all data from previous tokens to next tokens using pass-through input and output ports, data can be traced back using the parent tokens information to the token that actually holds the requested data. Using parent tokens, a workflow execution creates an acyclic directed graph of tokens. For instance, taking the simple benchmark workflow shown in Figure 5.13 p. 134 and execute it using the initial setup. On the right hand side of the figure the token graph that is built by the *Data Store* over time is depicted. As can be seen there are a total of ten tokens created during execution, all leading back to the initial token.

	Workflow	Token Graph
Step 1	 Finished not Finished Collecting Selector batcher 2 2 sleep soak	
Step 2	 Finished not Finished Collecting Selector batcher 2 2 sleep soak	
Step 3	 Finished not Finished Collecting Selector batcher 2 2 sleep soak	
Step 4	 Finished not Finished Collecting Selector batcher 2 2 sleep soak	
Step 5	 Finished not Finished Collecting Selector batcher 2 2 sleep soak	
Step 6	 Finished not Finished Collecting Selector batcher 2 2 sleep soak	

Continued on next page ↩

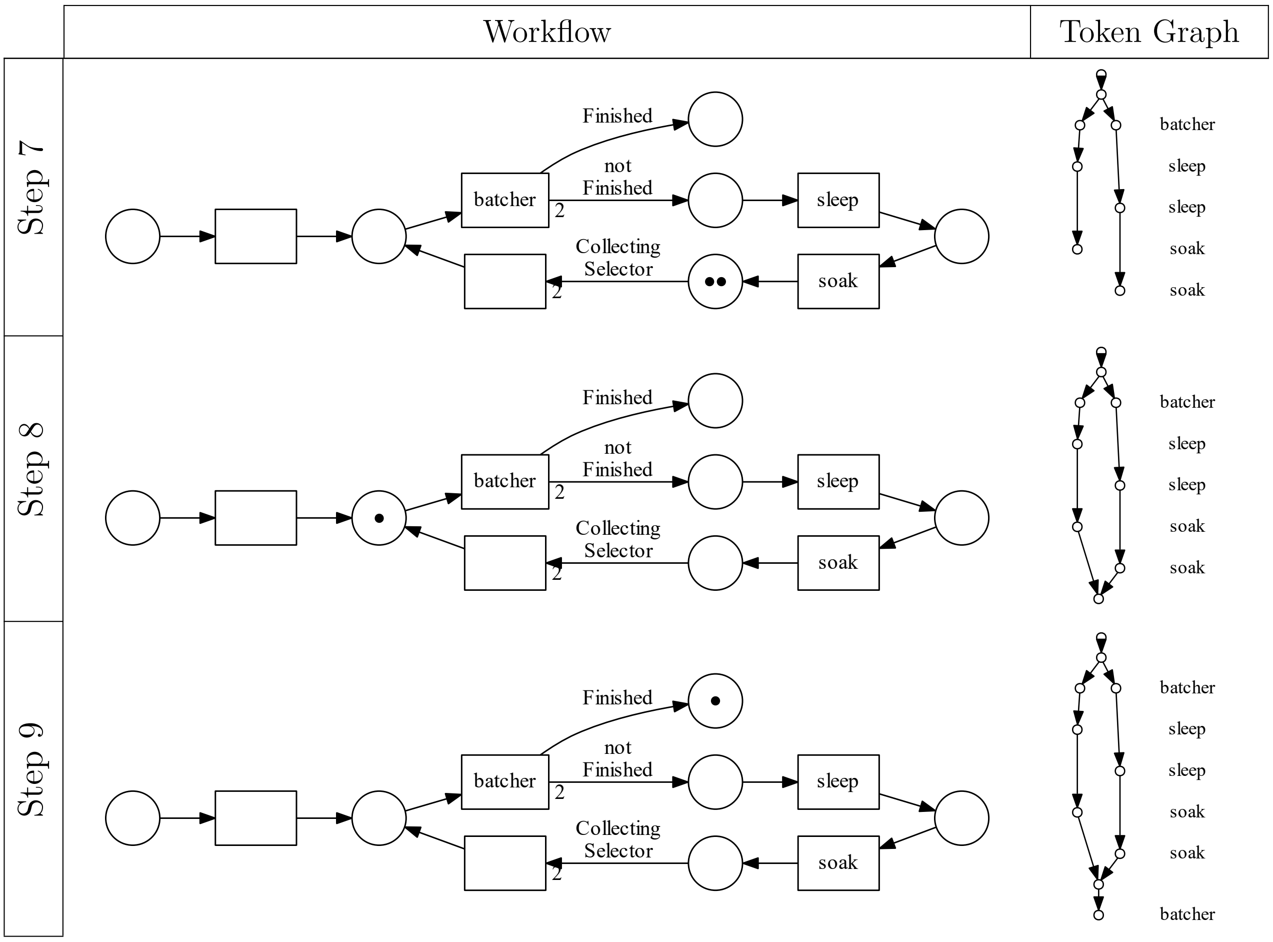


Figure 5.13: An example execution of the simple benchmark workflow used in Section 5.2.2.2 p. 153 using one batch and two parallel runs within the batch. The workflow execution consists of nine steps from start to finish. For illustration purposes work items are executed sequentially and whenever there are multiple active work items (work items with consumable tokens) a work item is selected at random. This means, the order of work items shown in this figure is one possible order for the given workflow. Each step shows the current configuration of the workflow net with tokens on the left and the corresponding token graph that is created within the *Data Store* on the right. The token graph also indicates which tokens were produced by which work item.

Figure 5.14 p. 135 shows the token graph and the associated data with each token. The *batcher* work item needs the information of the number of batches left as well as the number of parallel runs, which is provided by the initial token at

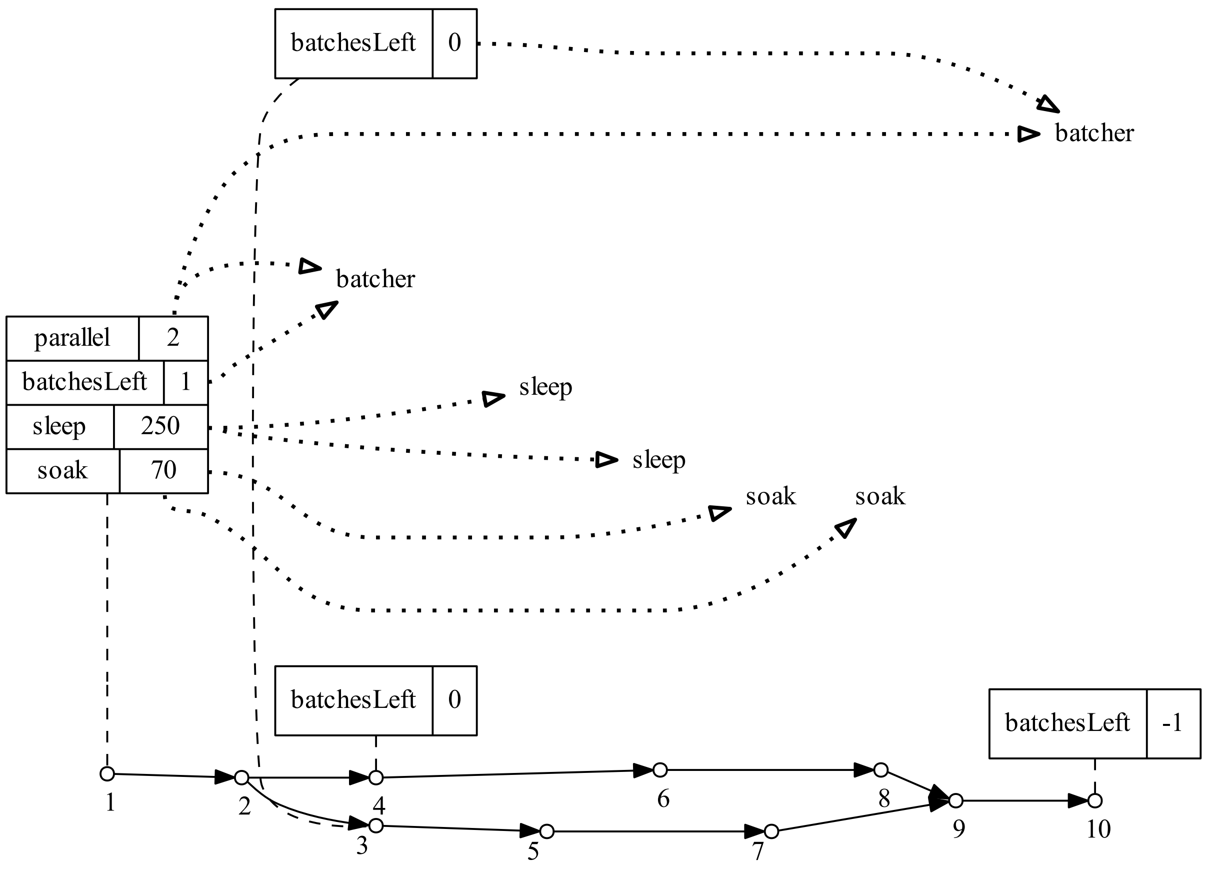


Figure 5.14: Token graph created from the workflow execution shown in Figure 5.13 p. 134. Additional information of data associated with tokens, if data was associated during execution, is shown along side the tokens (association is indicated with a dashed line). The work items are shown at the same position as the token(s) they produced. Also, the access of the data by the work items, is depicted using dotted arcs from the data to the work item accessing that data through a request to the *Data Store* handling the data and tokens. For instance, the first *batcher* work item accesses data (*batchesLeft* and *parallel*) from token 1, however on the second invocation it accesses only *parallel* from token 1 and *batchesLeft* from token 3.

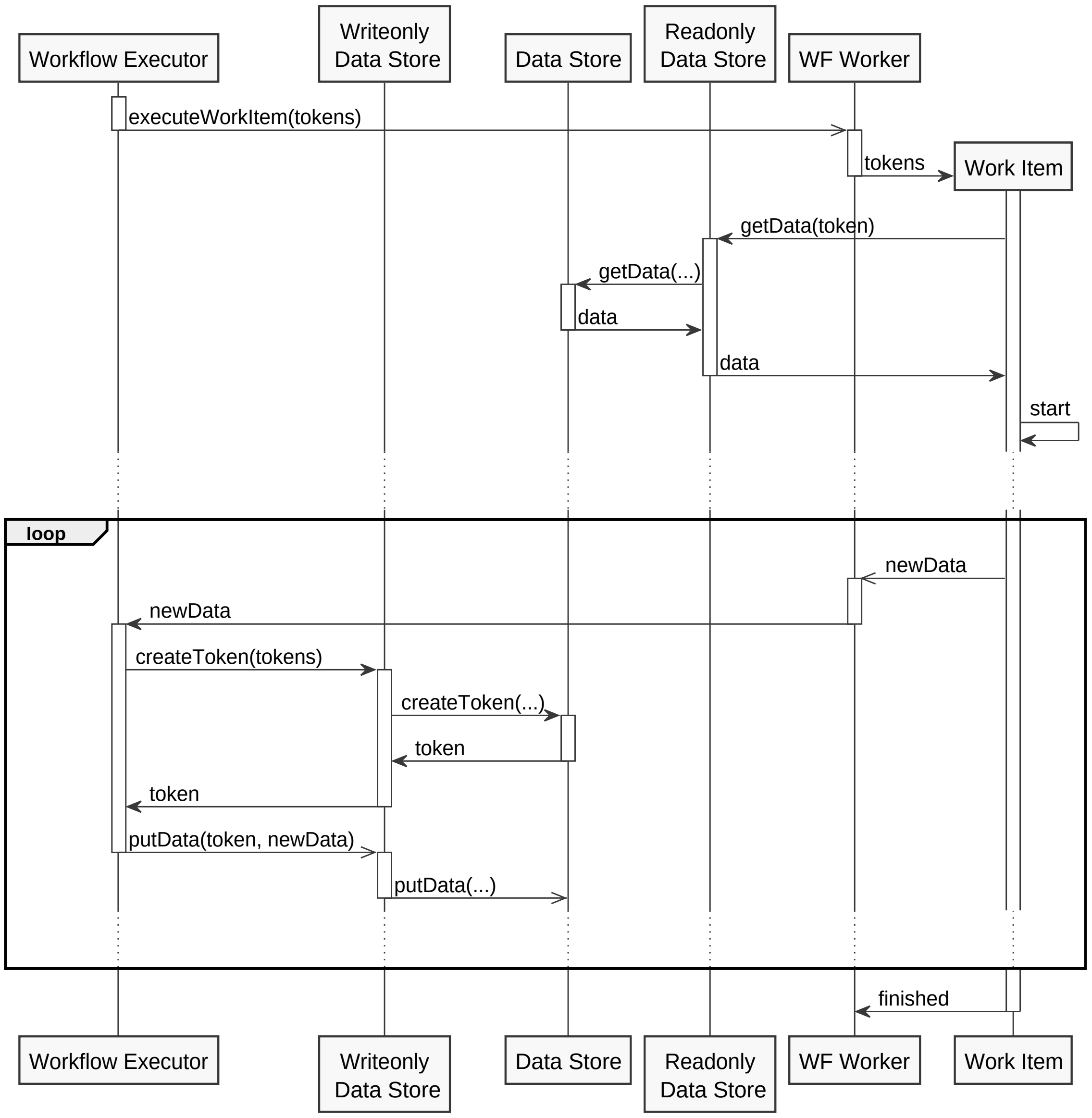


Figure 5.15: Sequence chart of how the *Data Store* is used in general during the execution of one work item of a workflow. It shows the indirect access to the *Data Store* using the *Readonly* and *Writeonly Data Store* respectively. For simplification the security component, more internal detail of the *Workflow Executor*, WF Worker, *Data Store* and Work Item are omitted.

first. In order to provide this data the *Data Store* would traverse the token graph backwards from token 2 (which is the input token for the first *batcher* work item invocation) back to the token providing the necessary data, which in this case is to token 1 for the *batchesLeft* and *parallel* data. For the second *batcher* work item invocation the *Data Store* would traverse back from token 9 back to token 1 to retrieve the *parallel* information as well as back to token 3 for the *batchesLeft* information.

This example shows that the *batcher* produces two tokens containing the same data, as specified by the *parallel* information. When the *Data Store* traces backwards to find a token providing the necessary data, it would find two tokens. In this case which one to take does not matter, as both tokens have the same data associated, however in other workflows this might not be the case. Multiple ways exist for how this can be handled in WorMS. On the one hand the first found token is used, which is the standard behavior. On the other hand an entity, e.g., a work item, can request all found tokens with that data and decide which one to use. This also allows requesting data produced, e.g., in a loop. And a third option is that an entity can explicitly specify from which work item the data should origin.

Storing data is fairly simple as the *Data Store* only has to store the data provided and associate it with the corresponding token.

During request as well as storage of data the *Data Store* works with the *Security and User/Role Management* component in order to restrict data access to specific roles and it also verifies whether the entity (work item, edge condition, etc.) requesting or storing data have corresponding input or output ports defined. Figure 5.15 p. 136 illustrates a simple request and store of data.

WorMS supports the execution of workflows in a distributed manner special data stores are needed that work in such a setup as well. Luckily, WorMS ships with a component that can turn any *Data Store* into one that supports distributed scenarios. In order to use a *Data Store* that does not support distributed executions, in a distributed environment the `DistributedActorDataStore` can be used. The usage is remarkably simple. For instance, using the memory based data store a changing something like `IDataStore`

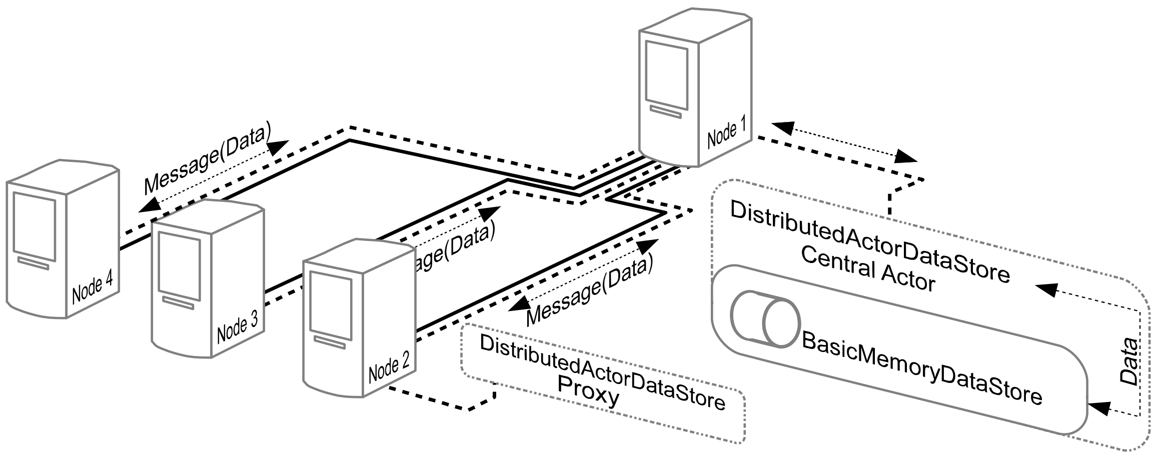


Figure 5.16: Distributed actor-based *Data Store* Wrapper. The distributed *Data Store* wrapper enables the use of any *Data Store*, in a distributed environment. In this example it wraps the `BasicMemoryDataStore`, which does not support the use in a distributed environment. The distributed data store wrapper uses the actor based paradigm also employed by the *Workflow Executor*. On the initiating node (Node 1) it creates a central actor, that talks internally to the `BasicMemoryDataStore` in order to requesting and storing data. On the other distributed nodes (Node 2–4) a proxy actor is deployed that connects back to the central actor, requesting and storing data using actor messages. It basically works like a client/server scheme leveraging the actor system that is already provided by the *Workflow Executor*.

`store=new BasicMemoryDataStore();` to `IDataStore store=new DistributedActorDataStore(new BasicMemoryDataStore());` is sufficient to make the memory based data store work in a distributed scenario.

The distributed data store wrapper is a lightweight component employing the Akka actor system and messaging between distributed store instances. Figure 5.16 p. 138 illustrates how this worker operates. It keeps the wrapped *Data Store* on the initiating node only (Node 1), while distributing an actor based proxy to the distributed nodes (Nodes 2–4) that connect back to the central node. Data is then transfered using actor messages and only stored in the initiating node using the wrapped *Data Store*.

Other implementations directly supporting distributed scenarios might make use of a distributed database, such as Cassandra/Dynamo (DeCandia et al., 2007; Lakshman and Malik, 2010) or Project Voldemort (Feinberg, 2011). They provide means of replicating data efficiently across nodes which could improve performance as data might be available locally on a node instead of only in central node.

WorMS comes with a couple of different *Data Store* implementations. Firstly, there is the `BasicMemoryDataStore` that stores data in memory providing very good read and write performance, however it does not persist data, which makes it less robust when it comes to memory errors or system crashes as the data cannot be restored in those cases. Due to the pure use of memory the amount of data that can be stored is limited to the available system memory. Secondly, `OutOfCoreDataStore` is a memory based *Data Store* which is backed by disk storage removing some restrictions, such as robustness when it comes to crash recovery as well as the memory limit, and is based on MapDB (Kotek, 2015). By persisting data, the *Data Store* is not just more robust, it also helps to support workflow reruns based on previous workflow executions. Thirdly, a distributed *Data Store* `DistributedHazelcastDataStore` based on Hazelcast (Johns, 2013) is available for the use in distributed scenarios. Furthermore, as already mentioned there is also the `DistributedActorDataStore` wrapper available that turns any *Data Store* into a distributed *Data Store*.

5.2.1.5 Converter

The intermediate representation of workflows in WorMS is based on workflow nets. However, not always are workflows specified in workflow nets or using the WorMS Java API. In that case, a workflow needs to be converted or transformed into WorMS' intermediate workflow representation.

The *Converter* component of WorMS is responsible for converting workflows. There can be an arbitrary number of converters, each dealing with a possibly different input representation of workflows, such as XPDL, BPMN or BPEL, the output representation however is fixed. For instance, Zha et al. (2008) present conversion rules for XPDL, Lohmann (2008) provides them for BPEL and Decker et al. (2008) provide a method that could be used to derive transformation rules for BPMN.

Allowing other workflow representations enables the modeling and description of workflows in any way, leveraging existing tooling as well reusing existing workflows from other systems, as long as a suitable converter exists.

5.2.1.6 Security and User/Role Management

The *Security and User/Role Management* component of WorMS takes care of authorization and user/role management including authentication. It is reused in different situations and by different other components, such as the *Data Store* and *Workflow Executor*.

The components interface mainly comprises the `IRole`, `ISecurityModel` and `IPrivilege` interfaces. Herein the `IRole` interface is used to handle user/role management as well as authentication without restricting the implementation. This allows for instance the implementation of a role system based on Kerberos (Miller et al., 1987) for authentication and authorization and LDAP (Koutsonikola, 2004) for user management and therefore incorporate existing and widely adapted technologies. However, WorMS provides a standard role that does not need to be authenticated and can be used directly if no further roles are required.

Another implementation might allow the exchange of the components providing further extension points, which would make it easy to use a database-based user management in combination with the existing authentication and authorization elements. Listing 5.13 p. 141

The `ISecurityModel` implementations together with `IPrivilege` implementations manage the authorization within WorMS. Thus, the execution of a work item, the access of data and the execution of entire workflows can be privileged to specific `IRole` instances. For instance, the *Workflow Executor* uses the security model to check whether the current active role can execute a specific workflow work item. The *Data Store* uses the security model to manage access to data generated by work items, while the *Workflow Engine* uses the security model to check whether the current active role is privileged to execute a specific workflow.

Listing 5.13: `ISecurityModel` implementation exemplified by the default implementation in WorMS. The default security model in WorMS does not restrict any data access, work item or workflow execution. However, data access is still restricted by the *Data Store* as only data advertised by input ports can be accessed by a work item, edge condition or token selector

```java
public class DefaultSecurityModel<T extends IInfo> implements ISecurityModel<T> {
  public static final ISecurityModel<IInfo> ALLALLOWEDINSTANCE = new
      DefaultSecurityModel<>(new DefaultPrivilege<IInfo>(true, new DefaultInfo("")));

  private final IPrivilege<T> privilege;

  public DefaultSecurityModel(IPrivilege<T> privilege) {
    this.privilege = privilege;
  }

  @Override
  public IPrivilege<T> canExecute(IRole role, ITask task) {
    return privilege;
  }

  @Override
  public IPrivilege<T> canAccessDataFrom(IRole role, ITask task, String id) {
    return privilege;
  }

  @Override
  public IPrivilege<T> canExecuteWorkflow(IRole role, IWorkflowModel model, String id) {
    return privilege;
  }
}
```

Listing 5.14: `IPrivilege` implementation exemplified by the default implementation in WorMS. The default privilege implementation either allows or denies all access depending on initialization (see line 5).

```java
 1  public final class DefaultPrivilege<T extends IInfo> implements IPrivilege<T> {
 2    private final boolean allowed;
 3    private final T info;
 4
 5    public DefaultPrivilege(boolean allowed, T info) {
 6      this.allowed = allowed;
 7      this.info = info;
 8    }
 9
10    @Override
11    public boolean isAllowed() {
12      return allowed;
13    }
14
15    @Override
16    public T getInformation() {
17      return info;
18    }
19  }
```

For an easier start with WorMS, it ships in concordance with the standard user implementation with a default security model and privilege implementation exemplifying a simple authorize all security model (see Listing 5.13 p. 141 and Listing 5.14 p. 141).

5.2.1.7 Analysis

A benefit of using a formal *Intermediate Representation* as internal workflow representation, is the availability of analysis methods for it. WorMS is prepared to integrate analysis capabilities so that the quality of workflows can be increased as well as their refinement can be supported. Since the framework uses a plug-in-based design and components work on the same *Intermediate Representation* it is possible for the analysis components to analyze a given workflow in any of the representations supported by the *Converter* components.

By providing the *Analysis* infrastructure WorMS can be extended by methods and metrics tailored to process analysis such as (Wynn et al., 2009; Mao, 2010; peng XIU et al., 2010; Weißbach and Zimmermann, 2010) and (Zha et al., 2011).

Model checking tools are also great to integrate into WorMS. A perfect candidate would be WofLAN (Verbeek and van der Aalst, 2000) as it focuses on workflow net analysis. Additionally, more general analyses focusing on the petri net aspect of the *Intermediate Representation* can be added to WorMS by integrating the model checking tool LoLA (Schmidt, 2000). With those tools properties such as the free-choice property as well as balanced AND/OR-splits and AND/OR-joins (typically an AND-split should not be complemented by an OR-join) can be checked. Especially the latter is important to ensure well-structuredness of a workflow.

By preparing analysis and model checking capabilities in WorMS the reliability and fault-tolerance as well as system stability of WorMS can be increased and ensured. Also, workflows can be enhanced, optimized and fixed based on results provided by the *Analysis* components.

Currently, WorMS is equipped with a well-structuredness analysis for workflows, ensuring exactly one start and one end place. This is a required property by WorMS in order to be able to reuse workflows as sub-workflows or frames. Otherwise, it would not be possible to determine where a workflow starts and where it ends which is crucial if embedded as sub-workflow or frame.

5.2.1.8 Monitoring

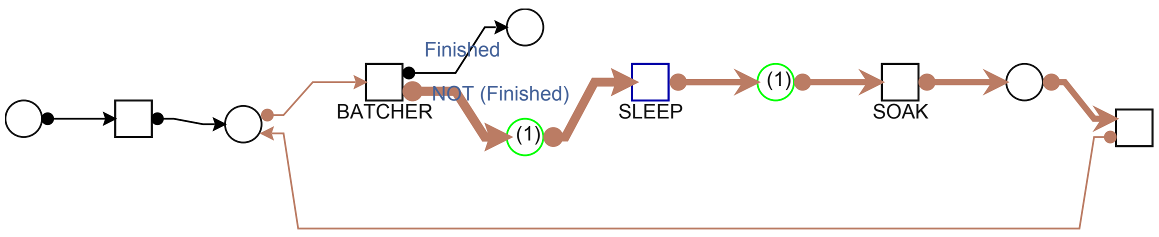
One of the key elements of the WorMS framework is the *Monitoring* extension point. It allows implementations to track, trace and document what, when, where, and how things are happening while executing a workflow. Thus, usage of the monitoring facilities of WorMS can range from auditing, reengineering and profiling (see Figure 5.17(b) p. 144 for a timing based profiling and Figure 5.17(c) p. 144 for a monitor profiling the throughput of different scheduling methods used in Section 5.2.2.2 p. 153) of workflows, for documentation, provenance generation and reproducibility of workflow executions as well as visualizing executions (see Figure 5.17(a) p. 144).

Important to note is, that although tracking the workflow execution is enough for auditing and profiling but for provenance and reproducibility also the used system, machine, and software components are needed to be tracked.

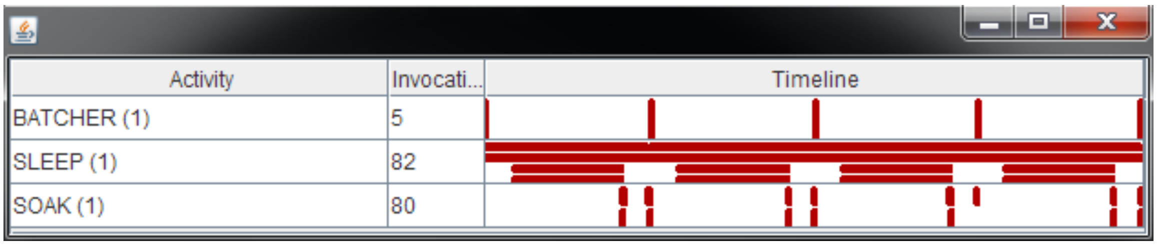
By providing a workflow monitor and additional system specific monitors (e.g., for monitoring architecture, software versions, operating system, machine specifications, etc.) provenance and documentation can be achieved at virtually any level of detail. For instance, by providing a dedicated monitoring component for a specific Modeling & Simulation system more information of the workflow run can be recorded, such as model or simulation parameters, events, or used software components, e.g., random number generators and event queues.

Also, not only can workflows be checked, reengineered, profiled and improved using monitors (Park and Kim, 2010; Accorsi and Wonnemann, 2010), the gathered information can also be used to profile, verify and improve the used *Workflow Executor*.

However, the most important responsibility of this extension point is the provenance and reproducibility aspect which helps to ensure result quality and credibility of the product generated by the monitored workflow (Wong et al., 2005; Davidson and Freire, 2008; Missier et al., 2008; Miles et al., 2008).

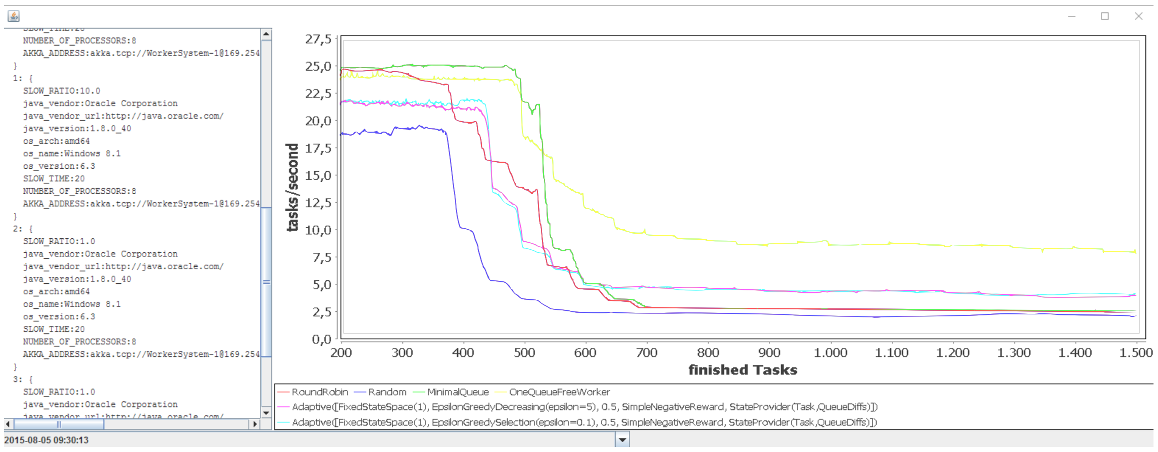


(a) Visualization monitor



Activity	Invocati...	Timeline
BATCHER (1)	5	
SLEEP (1)	82	
SOAK (1)	80	

(b) Work item timing monitor



```
  SLOW_TIME:20
  NUMBER_OF_PROCESSORS:8
  AKKA_ADDRESS:akka.tcp://WorkerSystem-1@169.254
 }
 1: {
  SLOW_RATIO:10.0
  java_vendor:Oracle Corporation
  java_vendor_url:http://java.oracle.com/
  java_version:1.8.0_40
  os_arch:amd64
  os_name:Windows 8.1
  os_version:6.3
  SLOW_TIME:20
  NUMBER_OF_PROCESSORS:8
  AKKA_ADDRESS:akka.tcp://WorkerSystem-1@169.254
 }
 2: {
  SLOW_RATIO:1.0
  java_vendor:Oracle Corporation
  java_vendor_url:http://java.oracle.com/
  java_version:1.8.0_40
  os_arch:amd64
  os_name:Windows 8.1
  os_version:6.3
  SLOW_TIME:20
  NUMBER_OF_PROCESSORS:8
  AKKA_ADDRESS:akka.tcp://WorkerSystem-1@169.254
 }
 3: {
  SLOW_RATIO:1.0
  java_vendor:Oracle Corporation
  java_vendor_url:http://java.oracle.com/
```

2015-08-05 09:30:13

tasks/second
27,5
25,0
22,5
20,0
17,5
15,0
12,5
10,0
7,5
5,0
2,5
0,0
200
300
400
500
600
700
800
900
1.000
1.100
1.200
1.300
1.400
1.500
finished Tasks
RoundRobin
Random
MinimalQueue
OneQueueFreeWorker
Adaptive([FixedStateSpace(1), EpsilonGreedyDecreasing(epsilon=5), 0.5, SimpleNegativeReward, StateProvider(Task,QueueDiffs)])
Adaptive([FixedStateSpace(1), EpsilonGreedySelection(epsilon=0.1), 0.5, SimpleNegativeReward, StateProvider(Task,QueueDiffs)])

(c) Workflow Throughput monitor

Figure 5.17: Monitoring Examples: (a) Real-time visualization of a workflow execution. Number of tokens per place and running work items are highlighted as well as the width of an edge indicates the number of tokens that passed over it over time normalized over all edges. (b) Monitor that shows the number of invocations per work item and the work items executions in a gantt chart. (c) Monitor that is used in the evaluation of work item scheduling policies in Section 5.2.2.2 p. 153, the monitor additionally collects data about machine setups, worker usage, etc.

Implementation-wise, a monitor implements the `IWorkflowEngineListener` interface and registers as listener for one or more workflows. Registering a monitor is as simple as calling `WorkflowEngine.addWorkflowEngineListener`. For instance, in order to use the visualization monitor shown in Figure 5.17(a) p. 144 a call to `WorkflowEngine.addWorkflowEngineListener(new VisualizationStarter())` is sufficient. This monitor will show a separate real-time workflow execution visualization for each workflow and sub-workflow.

The `IWorkflowEngineListener` combines methods for workflow started, finished, scheduled events and a `IWorkflowExecutorListener`. The `IWorkflowExecutorListener` provides methods for all the internal workflow execution events, such as `taskStarted`, `taskFinished`, `tokenAdded`, `tokenRemoved`, `edgeUsed` and more. Using those methods, it is possible to record the execution of a workflow in every little detail, even accessing and recording input and output data if needed.

5.2.1.9 Administration

The *Administration* extension point is meant for administrating different aspects of WorMS while it is executed. In general implementations should provide some kind of user interface, ideally a graphical user interface, to a running instance of WorMS. Over that interface it is supposed to at least provide means to control, e.g., pause, stop, and restart running and scheduled workflows as well as give an overview over currently running workflows and their state.

Furthermore, an administration component could provide means to manage security and user settings. For instance, it could manage, e.g., add, remove and edit users and their role. It could then manage privileges a specific role has, such as executing a workflow.

Another interesting task of an administration component is the management of the distributed infrastructure of workflow execution workers, such as adding, pausing or removing workers to a specific WorMS instance. Also, the start of workers on remote machines poses a useful task.

There are more responsibilities thinkable, e.g., since the administration component could also provide a list of monitor plugins that can be attached to workflow runs, scheduled or already running, removing the need to add them programmatically.

5.2.1.10 Workflow Repository

A *Workflow Repository* has two facets. Firstly, it can be used to provide workflows for execution. Such a workflow can be based on provenance or documentation data of a previously workflow execution or it is simply based on workflow newly added to WorMS.

Secondly, it can be used to store executed workflows possibly including documentation, data and provenance data. This is useful for increasing the credibility of published results, if the workflow that created the results and its provenance data is also available.

The repository holds meta-data along the workflow data, including version, documentation and provenance data, workflow tool, etc. if available. A repository allows for tracking evolution of workflows, gathering of feedback on results and used workflows to obtain them (Littauer et al., 2012), provide provenance information and documentation for published results, having a centralized storage for workflows for simply backing up or sharing of workflows and associated data and access to a workflow from anywhere, as long as the repository is accessible (e.g., interesting in distributed environments) (Goodman et al., 2014).

Herein, the idea of this extension point is to provide connections to established workflow repositories, such as the myExperiment repository (Roure et al., 2008; Goble et al., 2010).

5.2.1.11 Plug-in Provider

All the components in WorMS are extension points to be filled using actual implementations called plug-ins. WorMS already comes with implementations for most extension points, such as implementations for *Workflow Executor*, *Data Store*, *Monitoring*, *Security and User/Role Management* and *Analysis*. In order to manage available plug-ins the *Plug-in Provider* is used.

WorMS comes with an internally used implementation for the *Plug-*

in Provider organizing WorMS specific and included plug-ins. This allows it to be independent from external plug-in-system. Nevertheless, other providers can be implemented integrating other plug-in-systems from other Modeling & Simulation software such as the one used in JAMES II, connecting both system so that they can work together to integrate workflows, documentation, and provenance. To work with a Modeling & Simulation software using e.g., OSGi (OSGi Alliance, 2003) a specific *Plug-in Provider* based on OSGi would be needed.

Basically the *Plug-in Provider* acts as a plug-in-system abstraction. The *Plug-in Provider* is the main extension point which keeps the framework extensible and flexible within different Modeling & Simulation systems. It combines implementations with the internal mechanism to provide plug-ins that are not part of WorMS, e.g., plug-ins that are Modeling & Simulation system specific, such as workflow tasks based on actions within the Modeling & Simulation system. Additionally, the internal mechanisms can also be reused in Modeling & Simulation systems that do not have a plug-in system, allowing them to use the concept of plug-ins as well.

5.2.2 Adaptive Distributed Work Item Scheduling

In Paragraph 5.2.1.3 p. 124 the ability of executing work items in parallel or distributed, provided by WorMS is described. Performing work items in parallel or distributed can increase the execution speed of a workflow or in other words, reduce the total time needed to execute the workflow by leveraging computing resources, such as multi-core architectures or multi-machine environments. As already seen, from an implementation point of view it makes no difference whether a workflow is executed in parallel, having the workflow workers within the same VM or distributed across VMs or machines. Nevertheless, an important factor, particularly performance-wise, is the scheduling of a work item on a specific workflow worker, when it comes to the selection of that very worker. By scheduling the work item on the *right* worker performance can be increased, while selecting another worker for the same work item might lead to a considerable decrease in performance. For instance, assuming two workers and four work items to execute. For simplicity all work items are performing equally well on each of the worker.

However, a scheduling scheme where worker one is selected for all work items is the worst selection of workers that can be made in this case. Albeit, this is a very simple setup, as workers do not perform different on work items, this already shows how important it is to select the right worker for each work item (in the above scenario a uniform distribution of work items across the workers would be best).

Now, in reality not all work items behave the same on each worker, this is particularly true for distributed workers, with different hardware or in virtual environment. For the execution of a work item not only the computational aspect of a work item needs to be considered but also the data aspect, which deals with how much data needs to be transfered from and to a work item and therefore from and to the worker executing that work item, e.g., across a network. Scheduling a work item on th right worker is even more critical when there are different connection types between workers, such as internet-based (slow), gigabit local area network (fast) or direct connections using techniques such as IniniBand (InfiniBand Trade Association, 2000) (very fast) or when workers have different computational capabilities, such as a high-performance GPU.

So the question arises, how the scheduling of work items, hence the selection of workers can be handled. If worker infrastructure and its setup (such as connection speeds, hardware, etc.) is known as well as all computing and data bandwidth requirements of each work item, a hand optimized scheduling policy can work, even though it is hard to estimate the work item order of a workflow and the actual performance of a worker. However, in a dynamic environment, in which information is incomplete or where information changes over time, e.g., workers come and go, workers' performances changes due to contingent policies (others use the workers as well), unreliable connections between nodes and so on, a manual scheduling policy is not available and different approaches that can adapt to such dynamic environments are needed.

One option to address this problem is by employing adaptive systems that can adapt to such dynamic environments in order to optimize towards an objective, which in this case could be performance or energy usage. Adaptive systems can be found, e.g., in the domain of *Autonomous Computing*. In autonomous computing systems have to deal with the following problems: self-configuration, self-optimization, self-healing and self-protection, whereby the first two handle performance and the latter two handle reliability of an adaptive system.

An adaptive system can be defined as "Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible." (Laddaga and Veitch, 1997) and "Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation is meant." (Oreizy et al., 1999). The first definition assumes the system to only monitor itself and to adapt itself according to a specific objective such as performance. The second definition adds the observation of the environment the system operates in and takes that also into account when adapting itself towards an objective, it basically allows reacting on its environment.

When designing adaptive software, there are two approaches. On the one hand there is *Parameter Adaptation*, which changes parameters that influence the behavior of the software or system, however not allowing the providing of new algorithms after the adaptive system is developed. On the other hand, there is *Compositional Adaptation*, which is able to change entire algorithms or components of a software, allowing e.g., the creation and use of new algorithms after the adaptive system was developed.

Herein, for adaptation the system employs techniques of reflection. Reflection is decomposed into two parts: *Introspection* and *Intercession*. With introspection the system observes and examines itself and its behavior, while with intercession a system it modifies its behavior or state, reacting on information retrieved using introspection.

Nevertheless, an adaptive system needs to be able to adapt itself towards an objective. The problem is how this adaptation (intercession) determines how and what to adapt depending on the introspection so that the changes made actually lead to the desired objective. One approach is to use machine learning techniques, in particular techniques from the area of reinforcement learning (Kaelbling et al., 1996). Using reinforcement learning techniques allows to automatically find what to adapt and how to adapt it in order to optimize towards the desired objective.

5.2.2.1 Reinforcement Learning

Reinforcement learning is already used to solve adaptation problems in Modeling & Simulation (Ewald, 2012; Helms et al., 2013, 2015). The first tries to provide the best performing simulation algorithm (performance can be measured in computation time, memory consumption, etc.) for a specific experiment based on properties of that experiment, such as model and model parameters. Here, it uses reinforcement learning to learn from already made algorithm selections and their performance and adapts automatically in case a selection was sub-optimal. The second approach extends the simulation algorithm selection to support simulation algorithm adaptation during runtime, introspecting the algorithm's performance and adapting as needed on the fly.

The idea is to apply reinforcement learning in a similar fashion to schedule a work item on a specific workflow worker at a specific time in a specific execution environment and introspect its performance and derive knowledge (learn) a scheduling scheme.

Generally, reinforcement learning is an unsupervised machine learning approach which is iterative by nature and uses a reward system, where the learning algorithm tries to maximize the overall reward. Algorithms can be categorized into *Model-based* and *Model-free*. The first requires a model to operate on, which contains world knowledge such as state transition probabilities or the reinforcement function. A model-based algorithm basically learns a model beforehand (e.g., using training data) and uses the model to derive a controller from it. An example of such an algorithm is *Dynamic Programming*. Model-free algorithms however, do not need a model to operate, they learn a controller without learning a model, an example algorithm is *Q-Learning* (Watkins, 1989; Watkins and Dayan, 1992; Kaelbling et al., 1996).

Since it is difficult to derive training data for dynamically changing heterogeneous environment a model-free approach, in this case Q-Learning, is used to create an adaptive workflow worker selection policy.

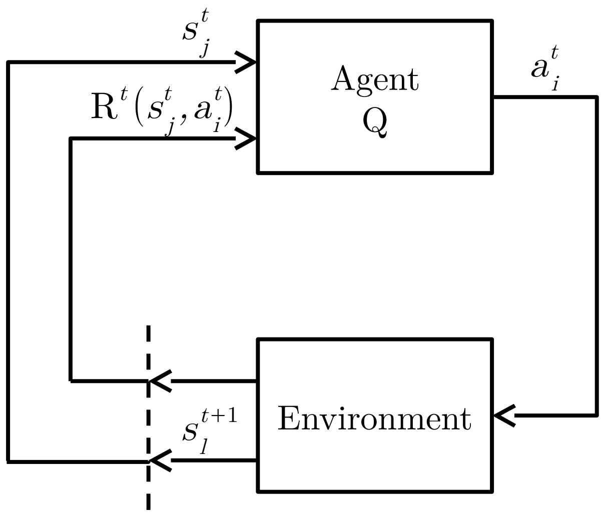


Figure 5.18: Q-Learning scheme

Q-Learning Q-Learning belongs to the model-free reinforcement learning algorithms. It consists of a set of Actions $\overline{A} = \{a_1, a_2, \ldots, a_n\}$, a set of states $\overline{S} = \{s_1, s_2, \ldots, s_m\}$, a rewards function $R\colon \overline{S} \times \overline{A} \to \mathbb{R}, (s_j, a_i) \mapsto R\,(s_j, a_i)$, and the learned knowledge function $Q\colon \overline{S} \times \overline{A} \to \mathbb{R}, (s_j, a_i) \mapsto Q(s_j, a_i)$. Additionally, a state, an action, a reward as well as the knowledge at time t is denoted by s_j^t, a_i^t, $R^t(s_j^t, a_i^t)$ and $Q^t(s_j^t, a_i^t)$ respectively.

The learning procedure is as follows:

(i) Agent has knowledge at time t represented as Q^t and the environment is in a state s_j^t and there are the Actions a_1 to a_n available.

(ii) Agent selects an Action a_i^t based on an action selection policy, explained later. Action a_i^t influences the state of the environment triggering a state transition from s_j^t to s_l^{t+1}. At the same time a reward is granted $R^t(s_j^t, a_i^t)$ and the Agent's knowledge Q^t is updated using the following equation:

$$Q^{t+1}(s_j^t, a_i^t) = R^t(s_j^t, a_i^t) + \gamma \max\left\{Q^t(s_l^{t+1}, a_1), \ldots, Q^t(s_l^{t+1}, a_n)\right\},$$

where $\gamma \to [0, 1]$ controls the influence of previously gathered knowledge. With γ the learning rate or adaptation rate is influenced. A larger γ means learning rate is lower, but it is less prone to reward outliers or fluctuating rewards. Consequently, a lower γ leads to a quicker adaptation and higher learning rate, but is prone for reward outliers or fluctuating rewards.

(iii) Continue with step i

Now, if this procedure is continued over and over selecting actions randomly, over time Q-values for each combination (s_j, a_i) are accumulated and represent the Agent's learned knowledge. Q-Learning is proven to converge, being able to provide an optimal decision making (Watkins and Dayan, 1992). In order to use the learned solution a selection policy based on Q-values, e.g., $greedy(Q)$ instead of a random action selection must be employed.

Action selection policies — Using Knowledge while learning As already stated, there are different ways for an Agent to select an action based on environment state and Q-values. The following overview introduces some common action selection policies that can be used with Q-Learning.

Random This policy selects actions randomly. This is useful for exploring different actions in different states in order to acquire knowledge about those actions and their performance in specific environmental states. A random action selection policy can also be used to train an Agent. Doing the learning procedure until the following holds.

$$\forall (s_j, a_i) \left((s_j, a_i) \in \overline{S} \times \overline{A} \land \left| Q^{t+1}(s_j, a_i) - Q^t(s_j, a_i) \right| < \epsilon \right).$$

Which basically means that over all state-action combinations is iterated until the Q-values only change within an ϵ interval.

However, acquired knowledge is not used, to make informed decisions on which action to select, using this policy.

greedy(Q) This policy uses Q-values in order to decide which action to select in which environmental state. It makes sense to use this policy, e.g., after a random action selection policy was used to already acquire knowledge. An action a_x is selected purely on the highest Q-value for a specific state s_j

with $a_x = a_i$, where $Q(s_j, a_i) = \max_{k=1}^{n} Q(s_j, a_k)$. In case there are multiple actions to choose from, one can be selected, e.g., randomly or by taking the one with the lowest index. Exploration does not take place using this policy.

ϵ-greedy This policy combines the first two policies *greedy*(Q) and *random*. In most cases it acts like the greedy(Q) policy, selecting an action based on the highest Q-value for a given state. However, the ϵ parameter influences the exploration rate $p\colon \overline{S} \to [0, \infty), (s_j) \mapsto p(s_j)$. For this policy the exploration rate at time t is defined as $p^t(s_j) = 1/\epsilon$ and is used to determine when to use the random action selection policy rather than the *greedy*(Q) policy. Basically, this policy explores on average every $1/p^t(s_j)$ action selections for a specific state s_j and uses Q-values otherwise. It provides a fair amount of exploration while still using already gained knowledge in order to make informed decisions on actions.

ϵ-decreasing Similar to the ϵ greedy policy, but decreases the exploration rate and therefore exploration over time. It starts with a specific ϵ, typically $\epsilon > 1$ and an state occurrence counter $c\colon \overline{S} \to \mathbb{N}, (s_j) \mapsto c(s_j)$ with initially

$$\forall s_j \left(s_j \in \overline{S} \wedge c(s_j) = 0\right).$$

The exploration rate at time t is defined as $p^t(s_j) = \epsilon/c^t(s_j)$, depending on the current environmental state s_j and after each action selection the counter is increased with $c^{t+1}(s_j) = c^t(s_j) + 1$. Starting with an $\epsilon > 1$ results in an exploration only phase for the first ϵ iterations per state.

5.2.2.2 Evaluation

A scheduling policy based on Q-Learning For the actual implementation of a Q-Learning based scheduling policy, $\overline{S}$ (states), $\overline{A}$ (actions) and R (reward function) need to be defined. Additionally, for the evaluation an action selection policy needs to be selected, which in this case will be the ϵ-greedy policy, with $\gamma = 0.5$ and $\epsilon = 0.1$. This policy allows a fair amount of exploration while still leveraging already learned knowledge from the beginning. A medium γ allows for a quicker adaptation to changes in the environment, while not being too prone to exceptional behavior (outliers).

The definition of actions, states and reward used in the implementation is described in the following.

States Generally, states are built from features that represent the current problem, e.g., solving a maze. Herein a state can be made of the agent's position in that maze. The agent's environment can only be in one state and a state transition is triggered by an action.

For the scheduling policy the environmental state consists of the *work item* that is to be scheduled and the *length* of each workflow worker's work item queue. Assuming there are n workers ($\overline{W} = \{w_1, \ldots, w_n\}$) and the workflow executed consists of m work items ($\overline{T} = \{t_1, \ldots, t_m\}$, a state is defined as

$$s_j = (t_k, \mathrm{l}(w_1), \ldots, \mathrm{l}(w_n)) \wedge t_k \in \overline{T} \wedge w_1, \ldots, w_n \in \overline{W},$$

where $\mathrm{l}\colon \overline{W} \to \mathbb{N}, (w_k) \mapsto \mathrm{l}(w_k)$ and donates the length of the work item queue of worker w_k.

For the evaluation the following holds $n = 4$ and $m = 5$. In order to avoid combinatorial explosion of the state space, which is built as $\overline{T} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ for this scenario with four workers and five work items, a restricted queue length function (l') is used instead and defined as $\mathrm{l}'(w_k) = \min(3, \mathrm{l}(w_k))$. l' only allows four different values that can be interpreted as actual queue lengths for $\mathrm{l}'(w_k) < 3$ and as *more than three* queue length for $\mathrm{l}'(w_k) = 3$.

Using l' results in a state $s_j = \big(t_k, \mathrm{l}'(w_1), \ldots, \mathrm{l}'(w_2)\big)$, and reduces the state space dramatically.

Actions Actions are used to trigger state transitions of the environment. There can be multiple actions available.

For the evaluation the set of actions $\overline{A}$ is defined as

$$\overline{A} = \overline{W} = \{w_1, \ldots, w_2\}.$$

This means each available workflow worker represents one available action.

Q-Values and Rewards Knowledge is gathered in the agent's Q-values through rewards awarded for an action selected in a specific environmental state at a specific time.

For the evaluation the reward is based on the time needed from scheduling until work item completion and is calculated as

$$R(s_j, a_i) = -100\,000 - \frac{100\,000}{\max\left(-\operatorname{d}(a_i)/1000, -100\,000\right)},$$

with $\operatorname{d}\colon \overline{A} \to [0, \infty), (a_i) \mapsto d(a_i)$ returning the time needed (duration) to execute a_i from the time of scheduling until the time a_i is finished.

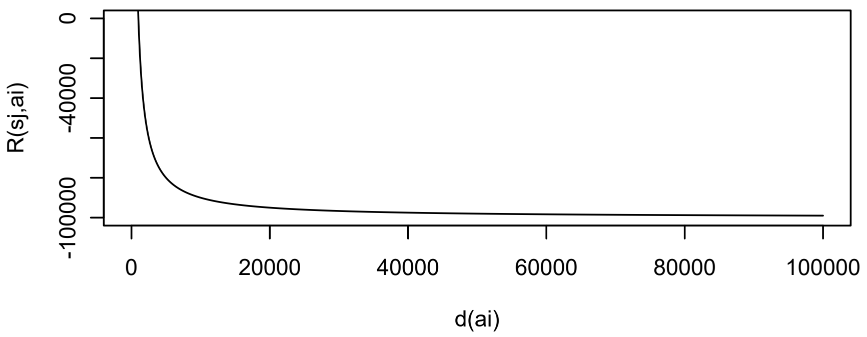


Figure 5.19: The Reward $R(s_j, a_i)$ over duration $\operatorname{d}(a_i)$

Consequently, the reward will be the lower the more time is needed for a work item to complete (see Figure 5.19 p. 155). However, the lowest reward is limited to $-100\,000$ which is also the reward earned on work item failure.

Setup In order to evaluate the Q-learning based adaptive scheduling policy introduced previously, it is compared against a selection of established scheduling policies, that can also be used to select a workflow worker to schedule a work item based on more or less complex internal rules. The policies the adaptive scheduling policy competes with are:

Random This policy is the simplest scheduling policy, selecting a worker randomly. This policy is to be considered the worst case here, and other policies should beat this one.

Round Robin Also a simple policy using a static selection scheme for scheduling work items on workers. Each worker is selected one after the other in a specific order, which does not change over time. Once each worker was selected, the procedure restarts at the first worker in order. This results in a uniform distribution of work items to workers. However, in a case where worker performance differs or changes this is a suboptimal scheduling scheme.

Minimal Queue This policy distributes work items to workers dynamically based on the number of work items currently in queue for each worker. It follows a simple idea which is also found, e.g., in situations where a queue is selected in a supermarket. If there are multiple counters in the supermarket, usually the counter with the shortest queue is picked. The same idea is used here. A work item is scheduled for the worker with the shortest queue.

One Queue Free Worker This is an alternative version of the minimal queue policy. Instead of having multiple queues per worker, there is only one global queue and each worker can only be in the state of executing a work item (being occupied) or not executing a work item (being free). A work item is put onto the queue when scheduled and then a work item is picked from the global queue if there is a free worker and is assigned to that free worker, changing the state of that worker to occupied. This is repeated for each work item to be scheduled.

The adaptive scheduling policy is evaluated in two scenarios and compared to the results of those policies applied in the same two scenarios.

For the evaluation of the benchmark workflow shown in Figure 5.20 p. 157 is used. As already mentioned, the evaluation comprises two scenarios. Firstly, a simple scenario is evaluated in which four distributed workflow workers are used to perform the shown workflow. The distribution however takes place on a

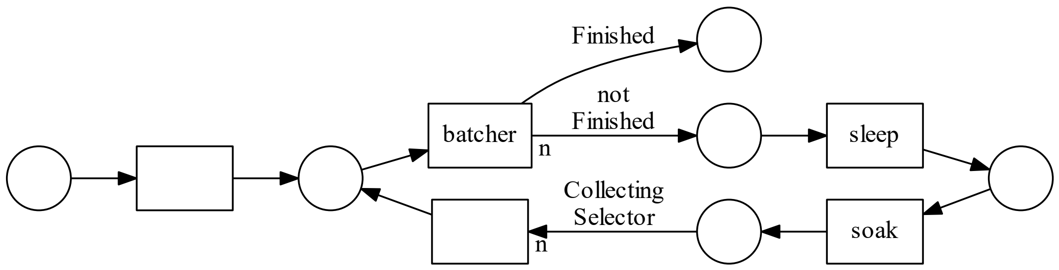


Figure 5.20: Benchmark Workflow Model used to evaluate different scheduling policies in different worker scenarios

multi-core machine where each worker runs in its own VM. All workers occupy one core each of a total of eight cores that are available on the test machine and for this scenario each worker is assumed to be equal computational-wise. Secondly, a scenario which simulates external load on two of the four workers, which for instance can be caused by resource sharing in a virtual environment such as cluster, or other activity on the node the worker runs on. Both workers will become ten times slower after 20 seconds and stay slow until the workflow execution is finished (see Table 5.1 p. 158 for a scenario overview).

All workflow executions are repeated ten times, the data transfered between work items is $1MB$, the *Batcher* creates 30 batches for each run and each batch contains 24 tokens to be processed allowing up two 24 work items to be executed in parallel simultaneously (depending on how many workers are present). Both scenarios are conducted with a sleep time of 250 ms for the *Sleep* work item and four workers are present for execution.

For the evaluation the following selection policies are compared: *Round Robin, Random, Minimal Queue, One Queue Free Worker* and *Adaptive* (see Section 5.2.2.2 p. 156)

		worker-0	worker-1	worker-2	worker-3
Scenario 1 (simple)					
	Slow Factor	x1	x1	x1	x1
	Slow Factor Time				
Scenario 2 (load)					
	Slow Factor	x10	x10	x1	x1
	Slow Factor Time	20s	20s		

Data Size	$1MB$
Replications	10
Batches	30
Parallel Work Items	24
Sleep Time	$250ms$
Selection Policies	*Round Robin, Random, One Queue Free Worker, Minimal Queue, Adaptive*
Action Selection Policy	ϵ-greedy with $\gamma = 0.5$ and $\epsilon = 0.1$
Reward function	$R(s_j, a_i) = -100\,000 - \frac{100\,000}{\max(-\,\mathrm{d}(a_i)/1000, -100\,000)}$

Table 5.1: The setup for evaluating of scheduling policies. All workflow executions are repeated 10 times, the data transferred between work items is $1MB$, the *Batcher* creates 30 batches for each run and each batch contains 24 tokens to be processed allowing up two 24 work items to be executed in parallel simultaneously (depending on how many workers are present). Both scenarios are conducted with a sleep time of 250 milliseconds for the *Sleep* work item and 4 workers are present for execution. In scenario 1 all workers have the same specification over time and none will be getting slower over time. In scenario 2, external load is simulated for *worker-0* and *worker-1*. Both will become 10 times slower after 20 seconds and stay slow until the workflow execution is finished.

Results

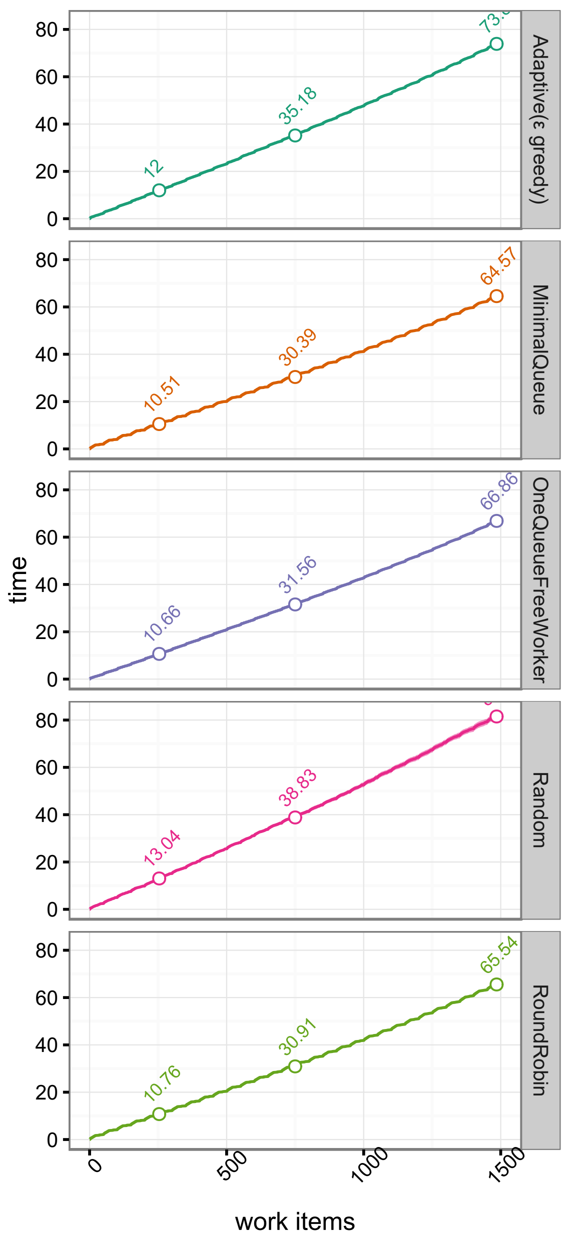
	work item	time	work items/second
Adaptive(ϵ greedy)			
	254	12.001 ± 0.163	21.92 ± 0.32
	750	35.178 ± 0.132	20.99 ± 0.38
	1485	73.892 ± 0.233	17.99 ± 0.26
MinimalQueue			
	254	10.505 ± 0.097	25.22 ± 0.20
	750	30.393 ± 0.120	24.05 ± 0.07
	1485	64.573 ± 0.264	20.38 ± 0.24
OneQueueFreeWorker			
	254	10.658 ± 0.027	24.70 ± 0.14
	750	31.558 ± 0.056	23.23 ± 0.05
	1485	66.862 ± 0.360	19.27 ± 0.46
Random			
	254	13.044 ± 0.228	19.79 ± 0.46
	750	38.826 ± 0.691	19.07 ± 0.53
	1485	81.507 ± 1.291	16.96 ± 0.42
RoundRobin			
	254	10.760 ± 0.244	24.77 ± 0.28
	750	30.911 ± 0.391	23.70 ± 0.15
	1485	65.540 ± 0.506	20.19 ± 0.17

Table 5.2: Evaluation Scenario 1: Overall time and Throughput

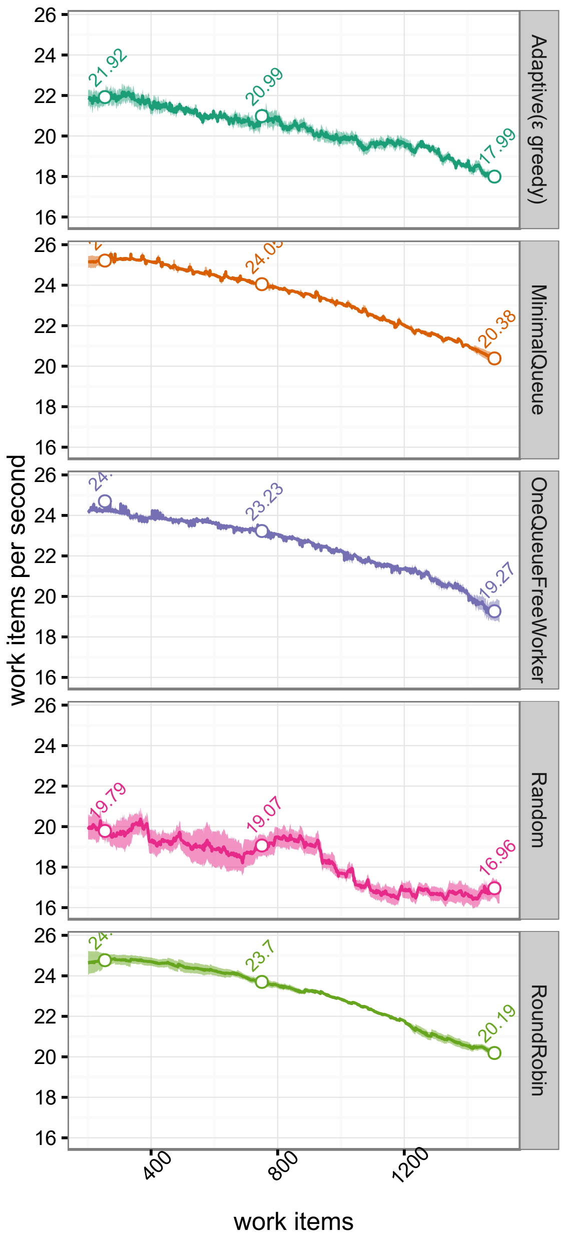
Scenario 1 — Homogeneous Environment In this scenario all workers behave the same performance-wise. They run each within a dedicated VM on the same machine using one CPU-core each. The scheduling policies which are compared with the adaptive scheduling policy are expected to behave similar, the random scheduling policy however is expected to perform a little worse than the rest.

Ideally, the adaptive scheduling method should learn over time to distribute work items evenly across available workers and exhibit similar performance values as the other policies.

Figure 5.21 p. 160 shows two charts comparing scheduling policies based on time needed to execute all work items as well as the throughput over time. Addi-



(a) Overall time needed Scenario 1



(b) Throughput Scenario 1

Figure 5.21: The overall time and throughput for each of the scheduling policies for Scenario 1. Both charts exhibit the average of time needed as well as of throughput over all replications and also shows the standard error (see Table 5.2 p. 159 for a tabular overview). (a): Shows the time needed to complete the work items over time. An almost linear incline in time needed for all scheduling policies is observed. (b): Shows the throughput (work items per second) for each scheduling policy over time. The moving average is calculated using a window of width 200, hence the first 200 work items are omitted. A nearly constant throughput over time for all scheduling policies can be observed.

tionally, Table 5.2 p. 159 shows the time and throughput including standard error values for three selected number of finished work items, which are also marked in the mentioned figure.

The time chart shows a linear increase of time needed with work items to be executed on all policies. *Minimal Queue, One Queue Free Worker* and *Round Robin* scheduling policy perform very similar time-wise as well as throughput wise. The *Random* scheduling policy performs the worst as expected. Interestingly, the *Adaptive* scheduling policy already performs better than the *Random* at an early stage needing 12.001 seconds vs. 13.044 seconds to execute 254 work items and an average of 21.92 work items per second vs. 19.79 work items per second after 254 work items. However, it does not reach the performance of the other scheduling policies during the execution of the workflow. Nevertheless, all policies perform very close to each other.

	work item	worker-0	worker-1	worker-2	worker-3
Adaptive(ϵ greedy)					
	254	23.25 ± 0.57	26.89 ± 0.48	25.27 ± 0.73	24.59 ± 0.98
	750	23.30 ± 0.35	27.14 ± 0.37	24.50 ± 0.33	25.05 ± 0.59
	1485	22.98 ± 0.25	27.60 ± 0.19	24.70 ± 0.20	24.72 ± 0.30
MinimalQueue					
	254	25.60 ± 0.11	23.81 ± 0.11	26.83 ± 0.25	23.75 ± 0.25
	750	26.67 ± 0.18	23.47 ± 0.10	26.56 ± 0.25	23.30 ± 0.17
	1485	26.93 ± 0.23	23.36 ± 0.19	26.65 ± 0.26	23.06 ± 0.12
OneQueueFreeWorker					
	254	24.09 ± 0.28	24.82 ± 0.42	25.10 ± 0.45	25.99 ± 0.34
	750	25.00 ± 0.15	24.79 ± 0.25	24.88 ± 0.32	25.34 ± 0.30
	1485	24.89 ± 0.17	24.97 ± 0.20	25.02 ± 0.24	25.12 ± 0.21
Random					
	254	24.54 ± 0.83	26.27 ± 1.39	23.70 ± 1.30	25.49 ± 0.92
	750	25.05 ± 0.61	25.28 ± 0.49	25.01 ± 0.66	24.65 ± 0.48
	1485	25.16 ± 0.54	25.16 ± 0.49	24.59 ± 0.59	25.10 ± 0.19
RoundRobin					
	254	25.10 ± 0.00	25.10 ± 0.00	25.04 ± 0.06	24.76 ± 0.06
	750	25.03 ± 0.00	25.03 ± 0.00	25.03 ± 0.00	24.90 ± 0.00
	1485	25.13 ± 0.02	25.20 ± 0.02	24.83 ± 0.00	24.83 ± 0.00

Table 5.3: Evaluation Scenario 1: Worker Usage

This is also shown in Figure 5.22 p. 162 and Table 5.3 p. 161 repectively. They give an overview of how often workers were used over time. For instance, for the *Round Robin* scheduling policy a ≈ 25% usage of each worker is exhibited, which

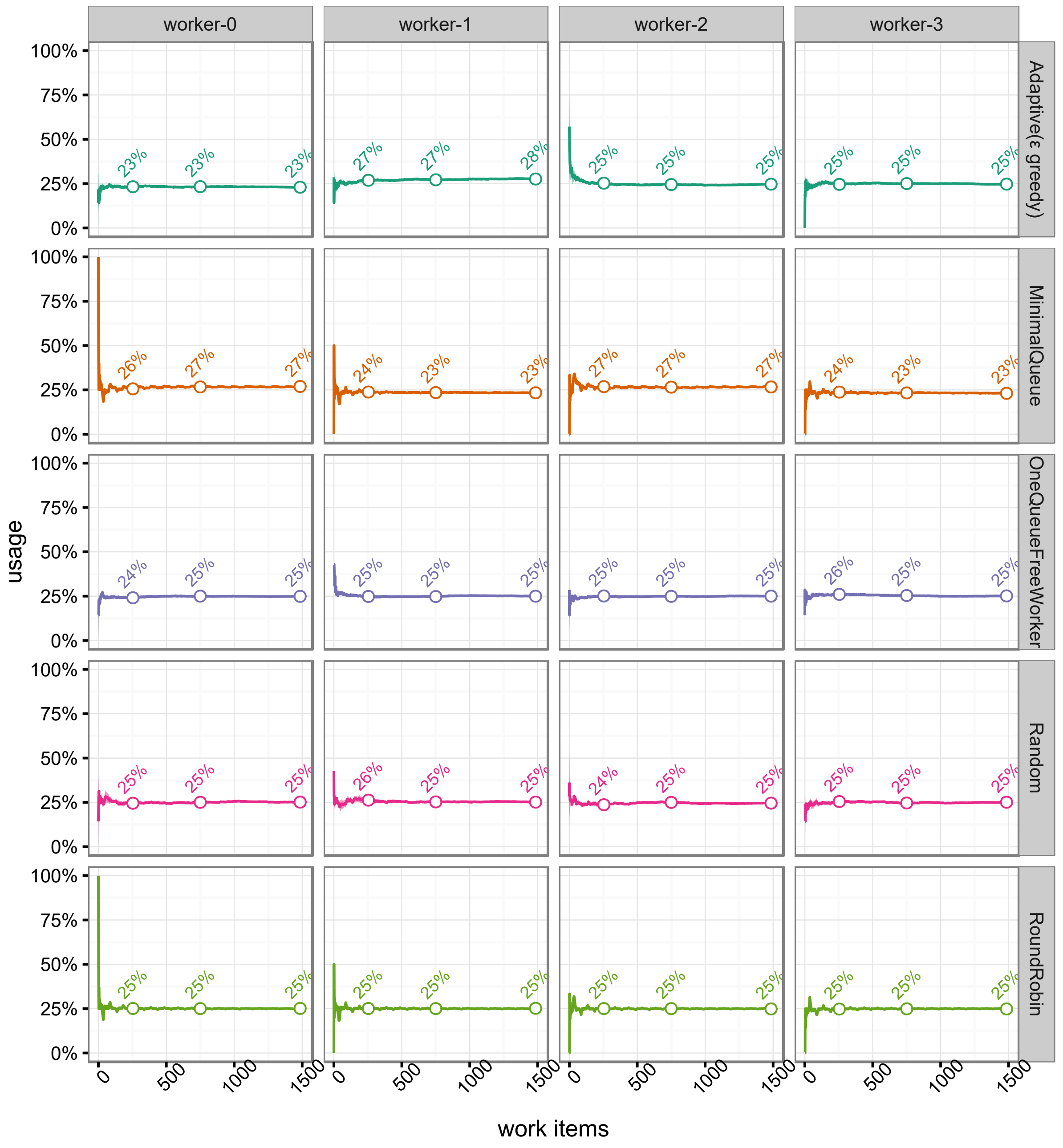


Figure 5.22: Worker Usage Scenario 1

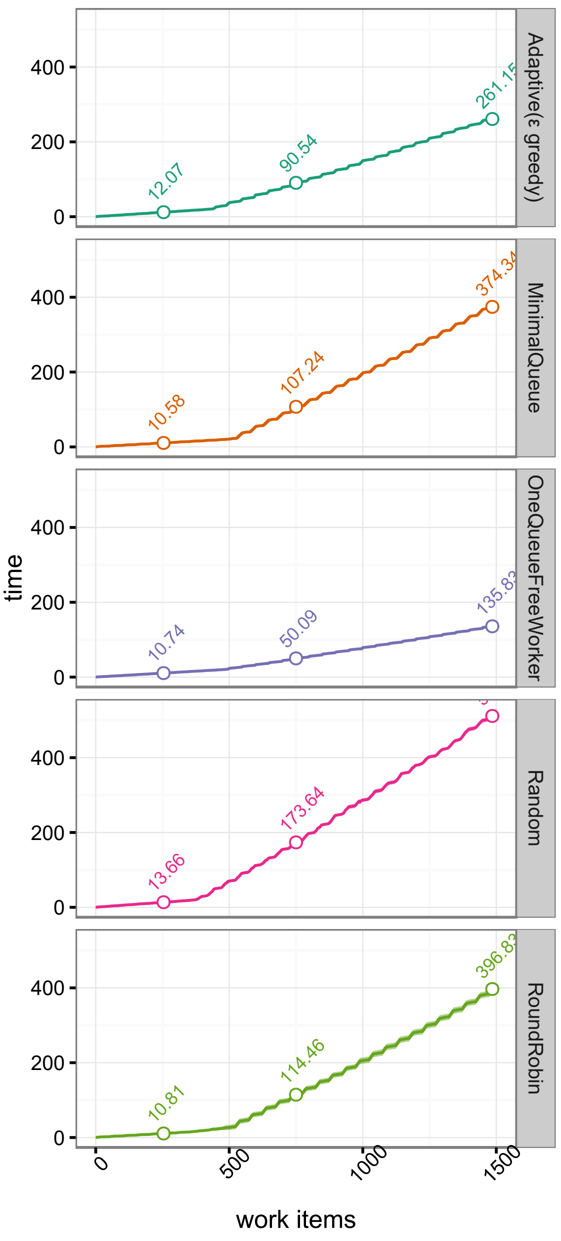
is expected since this policy distributes work items evenly across all workers. Interestingly, all other policies exhibit similar worker usage ratios, except the *Minimal Queue* policy, which nevertheless performs reasonably well. Especially noteworthy is the fact, that the *Adaptive* scheduling policy over time approaches the same worker usage ratio as the other policies.

Over all, it can be seen that the *Adaptive* scheduling policy is able to learn a good scheduling scheme based on the state information it has available and the reward function that is used. There are a few possible reasons for why it did not perform as well as the best performing policies in this scenario. Firstly, the number of work items was not enough to learn the optimal scheduling scheme. Secondly, the state does not contain enough information to learn a better scheduling scheme, there could be other factors influencing the performance not reflected in the current state implementation. Thirdly, a continuously exploring action selection policy is used, which might lead to sub-optimal action selections once in a while decreasing performance temporary on the one hand, however increasing knowledge on the other.

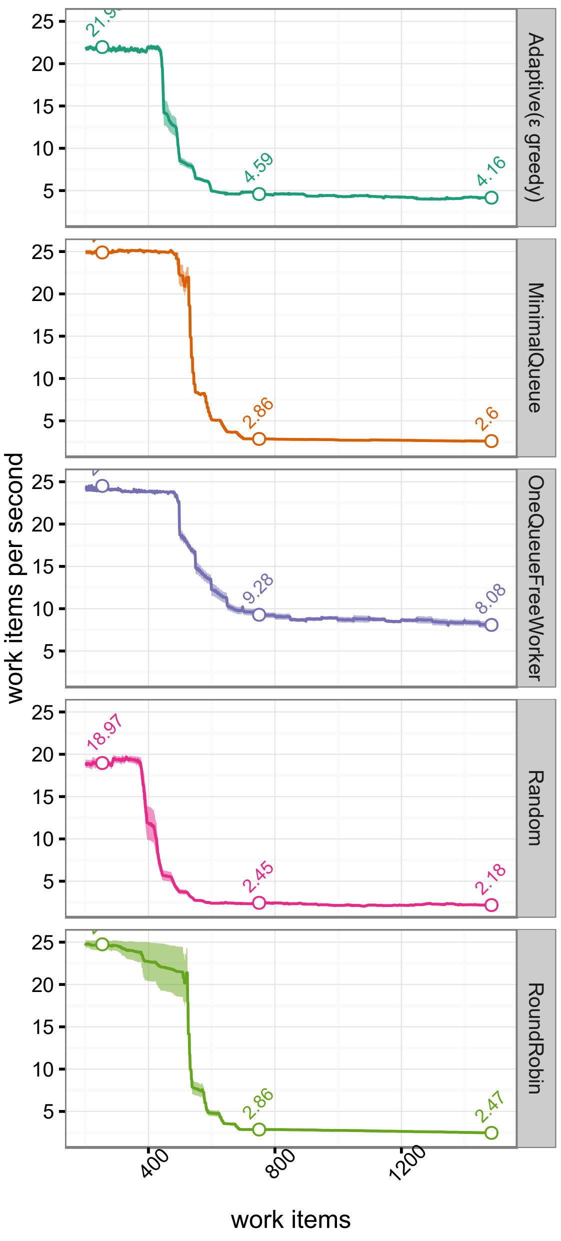
Scenario 2 — Heterogeneous dynamic Environment The previous scenario did not have a dynamic or heterogeneous environment the policies needed to adapt to, which made it easier for policies that do not use environmental information, such as *Random* or *Round Robin* to perform well.

This scenario starts with the same homogeneous environment as Scenario 1 does. However, the environment is changed into a heterogeneous worker environment after 20 seconds of workflow execution, by slowing down *worker-0* and *worker-1* by a factor of ten. This simulates a dynamic and heterogeneous environment at the same time. In reality this scenario is perfectly feasible in case workers are executed in virtual nodes, where virtual nodes share a physical node and a virtual node is using a lot of the capacity of the physical node. Another example of such an environmental behavior is in case a work item performs different on different workers.

Figure 5.23 p. 164 shows two charts comparing scheduling policies based on time needed to execute all workflow work items as well as the throughput over



(a) Overall time needed Scenario 2



(b) Throughput Scenario 2

Figure 5.23: The overall time and throughput for each of the scheduling policies for Scenario 2. Both charts exhibit the average of time needed as well as of throughput over all replications and also shows the standard error (see Table 5.4 p. 166 for a tabular overview). (a): Shows the time needed to complete the work items over time. For all scheduling policies the effect of the slow down of two workers after 20 seconds can be observed. However, the impact varies per scheduling policy. (b): Shows the throughput (work items per second) for each scheduling policy over time. The moving average is calculated using a window of width 200, hence the first 200 work items are omitted. The impact of the slow down of two workers after 20 seconds is clearly observable and affects each scheduling policy differently.

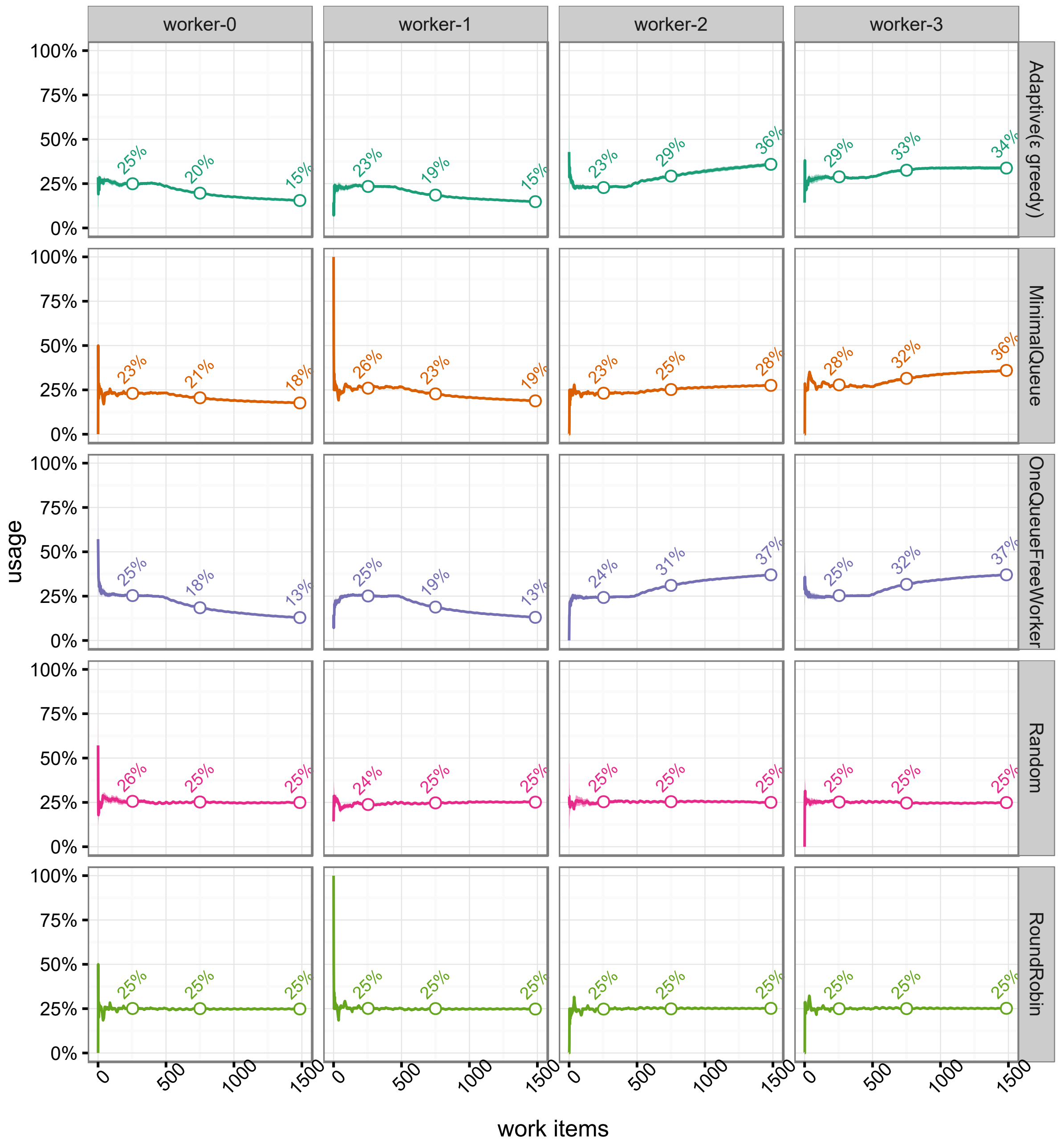


Figure 5.24: Worker Usage Scenario 2

	work item	time	work items/second
Adaptive(ϵ greedy)			
	254	12.074 ± 0.214	21.96 ± 0.28
	750	90.543 ± 2.139	4.59 ± 0.19
	1485	261.146 ± 2.784	4.16 ± 0.12
MinimalQueue			
	254	10.584 ± 0.082	24.89 ± 0.28
	750	107.240 ± 0.846	2.86 ± 0.02
	1485	374.336 ± 2.413	2.60 ± 0.03
OneQueueFreeWorker			
	254	10.737 ± 0.068	24.51 ± 0.24
	750	50.086 ± 0.991	9.28 ± 0.35
	1485	135.833 ± 2.486	8.08 ± 0.40
Random			
	254	13.656 ± 0.259	18.97 ± 0.36
	750	173.641 ± 3.876	2.45 ± 0.11
	1485	511.436 ± 6.828	2.18 ± 0.05
RoundRobin			
	254	10.805 ± 0.333	24.75 ± 0.46
	750	114.465 ± 7.128	2.86 ± 0.02
	1485	396.834 ± 8.526	2.47 ± 0.01

Table 5.4: Evaluation Scenario 2: Overall time and Throughput

time. Additionally, Table 5.4 p. 166 shows the time and throughput including standard error values for three selected number of finished work items, which are also marked in the mentioned figure.

Both charts clearly show the slow down of the two workers resulting in a drop in throughput and an increased incline of time needed to process work items. Interestingly, the drop of throughput as well as the increase of incline varies a lot between scheduling policies. The lease affected policy is the *One Queue Free Worker* scheduling policy. It only shows a slight increase of time needed to process work items and a smaller drop of throughput compared to the other policies. The worst performing policy is the *Random* scheduling policy, but again this is to be expected. However, the *Minimal Queue* and *Round Robin* scheduling policies follow closely behind *Random.* This is especially surprising for the *Minimal Queue* scheduling policy as it considers queue lengths of workflow workers when

	task	worker-0	worker-1	worker-2	worker-3
Adaptive(ϵ greedy)					
	254	24.93 ± 0.90	23.42 ± 0.51	22.80 ± 0.75	28.85 ± 0.85
	750	19.63 ± 0.37	18.55 ± 0.35	29.26 ± 0.92	32.57 ± 0.74
	1485	15.49 ± 0.29	14.84 ± 0.20	35.89 ± 1.03	33.78 ± 0.92
MinimalQueue					
	254	23.03 ± 0.27	25.99 ± 0.25	23.14 ± 0.27	27.84 ± 0.31
	750	20.54 ± 0.12	22.75 ± 0.20	25.20 ± 0.14	31.50 ± 0.16
	1485	17.64 ± 0.07	18.82 ± 0.10	27.56 ± 0.12	35.98 ± 0.18
OneQueueFreeWorker					
	254	25.32 ± 0.48	25.10 ± 0.21	24.26 ± 0.70	25.32 ± 0.75
	750	18.47 ± 0.15	18.83 ± 0.17	31.06 ± 0.50	31.63 ± 0.32
	1485	12.92 ± 0.14	13.06 ± 0.22	37.00 ± 0.50	37.02 ± 0.32
Random					
	254	25.60 ± 1.13	23.81 ± 0.57	25.38 ± 1.11	25.21 ± 0.94
	750	25.22 ± 0.63	24.73 ± 0.47	25.39 ± 0.31	24.65 ± 0.44
	1485	24.94 ± 0.53	25.13 ± 0.35	25.01 ± 0.19	24.91 ± 0.50
RoundRobin					
	254	25.10 ± 0.00	25.15 ± 0.06	24.76 ± 0.06	24.99 ± 0.07
	750	25.03 ± 0.00	25.03 ± 0.00	24.90 ± 0.00	25.03 ± 0.00
	1485	24.83 ± 0.00	24.81 ± 0.01	25.17 ± 0.00	25.19 ± 0.01

Table 5.5: Evaluation Scenario 2: Worker Usage

scheduling work items and it is expected that slower workers have longer queue lengths. The *Adaptive* scheduling policy performs relatively well and places itself between *One Queue Free Worker* and *Minimal Queue* policy, having a 1.6× higher throughput compared to *Minimal Queue*, however still having only half as much throughput as *One Queue Free Worker*.

When comparing worker usages for Scenario 2 as depicted in Figure 5.24 p. 165 and Table 5.5 p. 167, it can be seen that *Random* and *Round Robin* exhibit a very similar worker usage pattern to the pattern shown in Scenario 1. The pattern shows a close to uniformly distribution of work items across workers. The remaining scheduling policies, *Minimal Queue, One Queue Free Worker* and *Adaptive*, show a clear adaptation pattern at time of worker slow down. Each of the policies change the ratio of worker usage, reducing the amount of selections for *worker-0* and *worker-1* while increasing the amount of selections for *worker-2* and *worker-3*. The *One Queue Free Worker* policy performs best, almost halving the ratio for *worker-0* and *worker-1* from ≈ 25% to ≈ 13%. Followed by the

Adaptive scheduling policy reducing the ratio for the slow workers from ≈ 25% to ≈ 15%, placing it second performance-wise. The *Minimal Queue* policy reduces the ratio to only ≈ 18%, which results in an inferior performance compared to the other two policies.

Overall, in this case too, the *Adaptive* scheduling policy is able to adapt and learn a good scheduling policy, beating all other policies but the *One Queue Free Worker* policy. The performance of the *Adaptive* scheduling policy might be improved further to close the gap to the *One Queue Free Worker* policy by a different state definition using other environment information, more work items to learn over in case the learning rate is low, a different action selection policy as it affects learning rate and so on.

5.2.3 Integration into JAMES II

JAMES II exhibits a flexible experimentation layer that has been designed to conduct simulation experiments with models incorporating many different techniques (Himmelspach et al., 2008; Ewald et al., 2008). The experimentation layer can be described as an implementation of the skeleton pattern (Gamma et al., 1995). It defines all the possible experiments JAMES II can support, including parameter scans, optimization and validation experiments.

When integrating WorMS into JAMES II the goal is to support as much of the experiments as possible. Herein, the experimentation layer skeleton is replaced by the workflow presented in Section 4.2.2 p. 84. For this workflow to be integrated into JAMES II little effort is necessary.

Firstly, specific components have to be created in order to seamlessly connect WorMS and JAMES II. This includes a component that realizes an implementation of a *Plug-in Provider* built on top of the plug'n simulate architecture of JAMES II, which allows the use of JAMES II plug-ins within WorMS if necessary. Another component that needs to be created is a *Monitoring* and recording component that can monitor JAMES II specific information, such as available plug-ins and setup as well as plug-in selections and parameters within workflow tasks, e.g., used simulation algorithm, random number generator with which seed and event queue. This is important to provide provenance and documentation involving JAMES II internals. Provenance of the workflow execution and JAMES II internal provenance can then be combined to a more detailed and continuous overall provenance.

Secondly, a lot of what JAMES II already offers and uses in the current experimentation layer can be reused. However, it has to be encapsulated in either a WorMS task, WorMS template or WorMS frame. For instance, this affects the simulation run configuration, simulation run, single or multiple run analysis and so on. Special care has to be taken when turning existing JAMES II components into WorMS tasks, as workflow tasks require state less execution and immutable input and output data.

Thirdly, once all the JAMES II specific simulation and experimentation task, templates and frames are created and the workflows themselves have to be created and specified in WorMS.

Finally, in order to minimize migration overhead a `BasicExperiment` equivalent class exhibiting the same methods and semantics is created allowing to reuse already existing experiment setups by just changing from `BasicExperiment` to `BasicWFExperiment`.

Thus, by integrating WorMS in JAMES II and adapting the experimentation layer with workflows the following benefits emerge:

- JAMES II is easier extended to support unforeseen experiments by extending the experimentation workflow
- the experimentation is no longer a black box and provenance information is available
- documentation is automatically provided for the workflow-based experimentation layer
- detailed control options and state views can be provided using an *Administration* component
- resilience and robustness as well as automatic parallel and distributed execution of experiments are provided

- the experiment workflow executions can be stored using one of the emerging repositories for workflows such as myExperiment (Roure et al., 2008; Goble et al., 2010), assuming a corresponding *Workflow Repository* component is provided
- *Security and User/Role Management* support is added to experimentation

5.2.4 Challenges

During the development of WorMS a lot of challenging problems had to be solved. Those range from parallelizing and distributing workflow execution seamlessly over providing a flexible extensible software architecture to integrating flexible workflow descriptions based on templates and frames. However, there are interesting challenges and problems that are not completely solved yet. A selection and possible solutions is presented in the following. Moreover, they are associated to the WorMS component it is related to.

Intermediate Representation In order to consume tokens based on content or to consume more than one random token edges can define alternative token selectors. An interesting application is the use of a token selector that is shared across two or more edges leading to the same task selecting a token for one edge based on a selected token on another edge. The problem is however the explosion of token combinations to check for valid token combinations to consume. A possible solution is to provide a query like API for defining token selectors that can optimize the selection and reduce the complexity using techniques from other query languages such as SQL (Horng et al., 1994; Chaudhuri, 1998).

When designing workflows input and output port data types need to match in order to exchange data between tasks. There is a problem in case the data types do not match which results in an additionally needed transformation. The problem is also known as the shimming problem. There are approaches to deal with this problem, e.g., extending the descriptions of tasks providing information for automatic shimming (Lin et al., 2009).

Moreover, right now WorMS separates security model and workflow description. Nevertheless, in order to make informed decisions the security model needs knowledge about the workflow and its tasks. So it makes sense to integrate security constrains directly into the description of a workflow, deriving a security model for the *Security and User/Role Management* component automatically.

Workflow Executor The *Workflow Executor* is one of the main components of WorMS and a lot of challenging problems were already addressed during development. However, some enhancements and features are still open.

For instance, right now each token is handled as a separate entity by the *Workflow Executor*. A problem arises if there are a lot of tokens at a time in the workflow net during workflow execution, not including already consumed tokens which are only dealt with by the *Data Store*. Tokens can accumulate fast during execution, especially when executing the experiment workflows. For instance, assuming in a simulation experiment there are a 100 configuration with 1000 replications each to perform, could easily lead to 100 000 tokens present during workflow execution. Now 100 configurations can easily turn into 1000 or 10 000 and the number of replications can easily be 1 000 000. Handling this amount of data costs memory. An approach to reducing the memory usage could be to introduce population based tokens. What it does is to merge equal tokens present in a place, e.g., all the configuration tokens or replication tokens, storing only one actual token and how often this token is present in that place. However, once the tokens travel through the workflow net they are likely to diverge into individual tokens over time which only delays the problem. An additional solution could be to batch tokens, only allowing a maximum number of tokens present, the challenge here is to ensure that the maximum number does not lead to live or dead locks and that token selectors can still be satisfied. Moreover, using out of core mechanisms such as the ones used by MapDB can help to deal with high token count possibly sacrificing performance.

When executing a workflow distributed, work items need to be scheduled to workflow workers. In Section 5.2.2 p. 147 an adaptive scheduling policy was introduced that employs machine learning techniques to find an optimal scheduling scheme as well as adapt to changes to the distribution environment and infrastructure. First test and evaluations are promising, however the policy can be improved further, e.g., by using more features to represent the environment and utilizing adaptive state space implementations to cope with state space explosion in case of more state space features.

Lastly, in order to distribute workflow executions and to ensure reproducibility all workflow workers need to run the same versions of WorMS, Java and all the other libraries and classes, e.g., plug-ins from JAMES II. Right now this is done by hand. However, an automatic way of distributing the execution environment of WorMS including all necessary libraries, classes and so on is desirable. A possible solution could be to use cloned virtual machines on operating system level. Another solution might involve the development of a custom distributed class loader implementation that can synchronize classes and libraries from one central WorMS node to all the other nodes, enhanced by only transferring necessary classes, e.g., based on byte code checksums.

Data Store The *Data Store* already solves some problems, mainly memory efficiency, security and data management problems. For instance, instead of having each token link to its parent tokens this information is handled directly by the *Data Store*, reducing the memory footprint of a token dramatically, which in return increases token cloning performance as well as reduces the amount of data that needs to be transfered when a token is sent to, e.g., a distributed node and back to the *Data Store*.

This leads to another challenge that has to be mastered. Data size, the larger the data the more it affects performance of the workflow execution this is especially true when it comes to distributed executions, where large data needs to be transfered between nodes. A possible approach to this problem is to use monitoring statistics for work item executions, trying to estimate data size and using this information to efficiently distribute those work items to a node that is connected to the *Data Store* over a fast connection. This approach can be extended by analyzing the workflow before execution estimating which work item gets data from which work item combined with a distributed *Data Store* that can cache or prefetch data locally having large data asynchronously transferred to the work item's execution node before execution.

Converter When using a *Converter* component, a runnable WorMS workflow is expected. However, sometimes it is non-trivial to convert a task of another workflow into a WorMS task. It works well when dealing with web-services as they can easily be wrapped in a WorMS task, but poses a challenge when workflow description specific or workflow system specific internal routines are used or the task implementation is incompatible with Java out of the box. A solution could be to provide wrappers and native connectors to the other workflow system or task implementation but it has yet to be shown and investigated.

Monitoring *Monitoring* is responsible for providing provenance information and documentation. The question is how detailed should the documentation or provenance be and does it include all intermediate data or just input and output. Especially, in the case of simulation experiment workflows, a lot of data is produced over time. This is an interesting problem in terms of replay and reproducibility, providing enough information to be able to reproduce the results. Different approaches exist, for instance input, output and intermediate results can all be stored together with the used methods, workflows, software and hardware. Another approach could only store fractions of intermediate results at specific predefined points, e.g., a specific work item in the workflow or after a specific number of performed work items. Moreover, this could be combined with an incremental approach that can recalculate missing intermediate results either backwards or forward based on the recorded data, closing the gap between full recording and recording only a subset of data. This can especially be beneficial in case the amount of data is so high that storing would mean an enormous effort resource-wise as well as time-wise which makes it impractical. Even if storing would be practical recalculating data might be faster than storing and reading it back.

Workflow Repository A *Workflow Repository* is a necessary step to provide credible results as it gives people access to information on how those results were obtained. A problem poses the versioning of workflows, which not only includes the evolution of the structure of a workflow but also the evolution of each task implementation. It is essential that a workflow and the tasks and services it uses are versioned in order to ensure reproducibility (Woodman et al., 2011). However, an interesting challenge is to reapply existing provenance or documentation information to an evolved workflow. Implications and consequence need to be evaluated and investigated.

Another problem of a *Workflow Repository* is the possible decay of archived workflows. Overtime resources that were originally used to execute a workflow are changed, moved or are not available anymore. This usually applies to resources that are not controllable by WorMS or the Modeling & Simulation software, such as special hardware, operating system or simply external web services. In order to address those issues there are two ways. Firstly, there is preventing decay in the first place. Secondly, there is the fixing of decay once it occurred. For the first approach careful workflow planning and building, testing, reuse and maintenance could work (Hettne et al., 2012), while for the latter substituting missing services, adapting the workflow or leveraging provenance data for reconstructing behavior might work (Zhao et al., 2012).

5.3 Summary

The implementation has to consider two different scenarios when conducting a simulation study. On the one hand, there is the process of creating a simulation model (layer one). On the other hand, layer two deals with the execution of a simulation experiment, typically using the simulation model created in layer one.

Albeit both layers use workflows to describe the processes involved, they use two different approaches to cope with representing and controlling processes using workflows. In layer one an artifact-based approach is used which describes workflows declaratively. In contrast layer two uses a task driven imperative workflow approach. Thus, the implementation is divided into two different implementation approaches.

The presented implementation architecture for layer one is based on a rule-based engine. A rule-based engine is an obvious choice since it allows declarative descriptions of rules and condition, similar to the declarative nature of artifact-based workflows. It helps to specify what to do instead of how to do it and separate logic and data, representing logic as rules and data as facts or domain objects rather. Internally a rule engine typically employs the Rete algorithm which is efficient and scalable. Potential choices of rule engines comprise JESS, ILOG JRules, RuleML and Drools.

For the actual proposed architecture Drools is chosen as it was already used in another artifact-based workflow management system. Drools executes rules on facts generating or changing facts, which may trigger more rules until no more rule can be applied. Herein artifacts, milestones and stages are represented as facts and guards need to be converted to rules. Activities however are separated into activities that can be triggered directly using facts or by rules, called actions and activities that are activated by facts but need to be explicitly executed, e.g., by a human, called tasks. Albeit tasks and actions are part of the artifact-based workflow they are not mapped to entities of the Drools rules engine but kept separately. Executing actual tasks or actions is handled outside of Drools. In order to interact with facts tasks and actions can trigger events which become in return turned facts usable by Drools.

The most challenging part however poses the actual translation of artifacts, milestones and stages to facts and more importantly guards and sentries to rules.

Aside from being able to manage and execute an artifact-based workflow the proposed system needs to be integrated into JAMES II. Thus, two areas need adjustments. Firstly, plug-ins of JAMES II need to be made aware of restrictions that may occur to them based on facts currently present in the current workflow execution. This is achieved by hooking into the plug-in registry of JAMES II employing a proxy mechanism delegating the connection to available facts to the proxy implementation. By using a proxy existing plug-ins do not need to be directly altered or extended reducing integration effort dramatically. Secondly, the user interface of JAMES II needs to be adapted adding a mechanism to control the enable-state of buttons, menus, menu options and so on for specific tasks based on available facts. In contrast to plug-ins this integration is more complicated.

A completely different architecture and approach is used for the implementation of layer two. In principle existing generic workflow systems, originating from the business process or scientific workflow domain could be employed in order to implement the prensented experiment workflow. Alternatively, a system for supporting Modeling & Simulation workflows can be designed and implemented from scratch. The advantage of a new system is that it can be tailored specifically to the requirements of the domain of Modeling & Simulation and its processes at hand. In general, such a system should exhibit a software design that is flexible and extensible and supports a workflow representation with clearly defined semantics. Those semantics ensure that a workflow when executed by different systems is interpreted correctly. Such a representation is given by Workflow Nets.

In order to execute and manage workflow based on Workflow Nets in the domain of Modeling & Simulation the framework WorMS was developed. WorMS supports the integration of workflows in Modeling & Simulation software such as JAMES II. Thus, WorMS on the one hand supports and assists the creation and redefinition of workflows. On the other hand, it presents an architecture which is used to execute Modeling & Simulation workflows which guides and assists Modeling & Simulation scientist and produces reproducible and documented results.

Technically, WorMS is plug-in-based and comprises different exchangeable components, such as *Security and User/Role Management*, *Plug-in Provider*, *Workflow Repository*, *Workflow Engine*, *Intermediate Representation*, *Administration*, *Analysis*, *Converter*, *Monitoring* and the most essential parts the *Workflow Executor* and *Data Store*. WorMS comes with different implementations for those components. For instance it ships with a *Workflow Engine* that supports Workflow Nets based workflow descriptions, different *Data Stores* implementations (Memory-based, persistent, distributed, etc.), a *Workflow Executor* implementation that is able to execute a workflow parallel and distributed as well as different Monitoring (performance or documentation) implementations.

The implementation of the *Workflow Executor* realizes the parallel and distributed execution of a workflow by employing the actor-based concurrency paradigm. In particular the implementation leverages the Akka actor framework. Albeit Akka comes with schemes to distribute work across actors, such as round robin or a minimal queue-based policy, it does not always lead to the best distribution, leaving room for improvement. Herein, an adaptive distribution policy was presented, that uses machine learning mechanisms to learn an optimal distribution scheme over time for a specific workflow. In order to be user friendly, thus avoiding overly complicated or extensive configuration a model-free reinforcement learning, Q-Learning, is used. Being model-free no knowledge about the system is needed, hence configuration can be kept to a minimum. Q-Learning uses a reward mechanism to distinguish good from bad decisions, that is good or bad worker selection for a specific work item. If a good worker for a specific work item was chosen a high reward is granted, a low reward otherwise. Decisions are made based on features previously defined for the system, such the work item's id, worker queues, data size of input and output values and so on.

Evaluations have shown that an adaptive distribution policy based on Q-Learning is able to adapt distribution of work across actors as good as or better than standard distribution policies, such as round robin or minimal queue. However, performance of this method is highly dependent on the features used for learning.

When integrating WorMS into JAMES II some changes to JAMES II needed to be made. The idea is to replace the existing experimentation layer by an equally powerful, flexible and extensible experimentation layer based on WorMS and the herein presented experiment workflow, additionally providing automated documentation and provenance information. However, additionally to integrating WorMS special plug-ins for JAMES II needed to be created.

Firstly, a Monitoring and recording component that can monitor JAMES II specific information, such as available plug-ins and system setup as well as selected plug-ins and parameters during the execution of workflow work items, e.g., used simulation algorithm, random number generator with which seed and event queue. Secondly, JAMES II specific simulation and experimentation task, templates and frames were created. Thirdly, the presented experiment workflow itself had to be created and specified in WorMS using previously created JAMES II specific simulation tasks and frames.

During implementation different challenges arose and are only partly solved in WorMS. For instance, control-flow in WorMS is partly achieved using edge conditions and token selectors. As long as such edge conditions and token selectors only refer to one place there is no difficulty, however when a token selector spans more than one edge, therefore referring to multiple places to consume tokens from, ways to avoid combinatorial explosion are needed to ensure efficient execution. Another interesting task is the efficient wiring of input and output values of workflow tasks. Especially providing shimming functionality for values that do not have compatible data types is a challenging endeavor. When executing a workflow a trail of tokens is produced. Depending on the workflow the number of tokens produced over time can reach a point where managing them becomes challenging, memory, documentation and retrieval wise. Right now this is addressed by special *Data Stores* and by the workflow itself, e.g., producing tokens in batches rather than all at once. Directly related to the number of tokens present during workflow execution is the data associated to each token, that can even quicker become a challenge to handle, with storage and retrieval being only one problem and transferring that data across distributed nodes being the other. Partly, on the one hand this can be addressed by *Data Stores* that do not hold all data in memory. On the other by employing a work item distribution policy that takes data transfer into account, avoiding transfers of huge data to distributed nodes if not reasonable. Reason can for instance be determined by the herein presented adaptive distribution policy. The learning rate is highly dependent on the used features it is wise to further increase performance of the adaptive work item distribution policy by utilizing even better suitable features and an adaptive state space representation avoiding state space explosion, leading to increased learning rate. However, the challenge lies in selecting the right features and the appropriate state space representation.

6 Discussion & Future Work

If this is coffee, please bring me some tea; but if this is tea, please bring me some coffee.

Abraham Lincoln

In this chapter the presented tools, frameworks and methods are put into perspective with respect to how they help to overcome the crisis of credibility of simulation studies. The basic idea of the solutions presented is to structure the process of conducting a simulation study making it clearly defined. The frameworks presented help to automate the documentation and provenance data collection of that very process during the life-cycle of a simulation study as well as the assurance that the process if followed as specified. Documentation and provenance information can be used to reproduce, rerun, adapt and extend simulation studies later on.

Ultimately, a clear understanding of reproducibility is needed.

6.1 Reproducibility

"[. . .], reproducibility designates the ability to confirm the results of a previous experiment by means of another similar experiment. " (Dalle, 2012). Interestingly, in physics a standard exists that defines criteria for reproducibility and repeatability taking results of measurements into account (Taylor and Kuyatt, 1994).

In the field of Modeling & Simulation reproducibility refers to the ability to produce similar results by executing similar simulations. However, the results, or data for that matter, are not necessarily identical but have to exhibit the same properties according to a specific metric, e.g., a statistical property such as distribution or the same trajectory just sampled at different points.

Dalle (2012) defines four levels of reproducibility in terms of scenario and instrumentation. A scenario refers to a possible history of the system simulated and can be produced using dedicated means such as a domain specific language or simply use the same elements used to define the model. Instrumentation however, defines variables that are of interest and need to be observed, during the simulation task *Data Collection* and the computations that need to be applied to the data produced by these variables, during the simulation task *Analysis.* In the following the four levels of reproducibility are briefly described.

Level 1 Refers to the identical computation including completely deterministic behavior. It is also known as repeatable. For instance, in case of a discrete event simulation a level 1 reproduced or repeated simulation has to execute the same exact simulation, which means it has to have the same series of events which need to be processed in the very same order. This is especially difficult to achieve in parallel distributed environments, because processing has to be synchronized deterministically (Fujimoto, 2000). In addition, level 1 also requires identical instrumentation. It is also recommended to use the same source code, e.g., for simulation algorithm, for re-execution, but not necessary. Level 1 aims at debugging and error analysis of simulation experiments as well as simulation algorithm testing.

Level 2 This level is more relaxed than level 1. However, it still refers to an identical computation when re-executing a simulation, but does not require the deterministic nature level 1 exhibits. This means, for instance in cases where events occur at the same time, the order can be different. Level 2 reproducibility is often found in parallel and distributed environments because of the lack of synchronization or from the inability to completely control execution, e.g., in real-time simulation (McLean and Fujimoto, 2000).

An identical instrumentation is required in order to achieve level 2. This level can be used to verify existing simulation results or compare simulation algorithms to a reference implementation.

Level 3 Level 3 reproduces the scenario and does not necessarily require identical computation. Re-execution if typically based on detailed specification, e.g., by recreating the model and experiment using different tools, frameworks or formalism. This level occurs when the specification might not reveal all details for a level 1 or level 2 reproducibility, e.g., the sequence of random numbers cannot be reproduced without actual implementation and seed details. The data produced in a level 3 reproduced simulation might not produce the same data but still must inhibit the same properties according to a metric. This means that a re-execution at level 3 should reproduce the same phenomenon model, requiring that data is identical from a statistical point of view, with the assumption that instrumentation is identical. This level is what to aim for when conducting simulation experiments.

Level 4 This is the most relaxed and general form of reproducibility. It requires the re-execution to be a similar scenario and is typically found where it is impossible to reproduce the exact same experimental conditions, such as in real-time simulations.

Which level can be achieved depends on the specification (documentation) available and the elements used in the specification (model, algorithm source code, technical environment). Availability of elements and the specification is mainly influenced by two factors. On the one hand there is the human factor and on the other technical issues.

The human factor typically influences the specification. For instance, due to unawareness of hidden parameters and settings, e.g., which random number generator is used internally by a simulation algorithm, this is likely missing from the specification. Further, insufficient documentation and specification may result from space and time limitations, e.g., when publishing work. It can also stem from business limitations, for instance the model, or the source code of a simulation algorithm used cannot be made public due to copyright restrictions.

Additionally, the specification can be inaccurate or incomplete when manual tasks during simulations are involved and are not sufficiently tracked or hard to reproduce.

Technical issues on the other hand refer to software bugs, software availability, floating point number problems and hardware or operating system dependencies. Software availability issues usually result from business limitations, where software cannot be made available due to copyright or when a specific version of the software is not available anymore due to new versions of that software. This directly relates to the problem that software bugs pose to reproducibility. Due to software bugs, valid models can produce erroneous or different behavior, however after fixing these software bugs this might lead to different results and referenced versions of the software exhibiting the bug are likely not available anymore, which means level 1 or level 2 is not achievable anymore. Interestingly, the results from calculations using floating point numbers highly depend on the internal implementation of the floating point numbers. Simple calculations can lead to wrong and completely different results using different arithmetic implementations (floating point arithmetic and integer arithmetic) (Vicino et al., 2014). Long term reproducibility might require obsolete computer hardware or architectures not available anymore, this can also happen if a simulation has dependencies to a specific OS or OS version, e.g., due to the use of Operating System specific libraries which affects the level of reproducibility, if reproducible at all, in case such versions or hardware is not available anymore.

6.2 Layer One — Model Creation

6.2.1 Example Model Creation

In Section 2.4 p. 35 difficulties describing processes using a task-based workflow description are outlined, if the process involves a lot of freedom and flexibility, which is usually found when dealing with processes with a lot of human interactions requiring a large degree of freedom during execution. An alternative approach presented is to use declarative workflow descriptions. In particular a

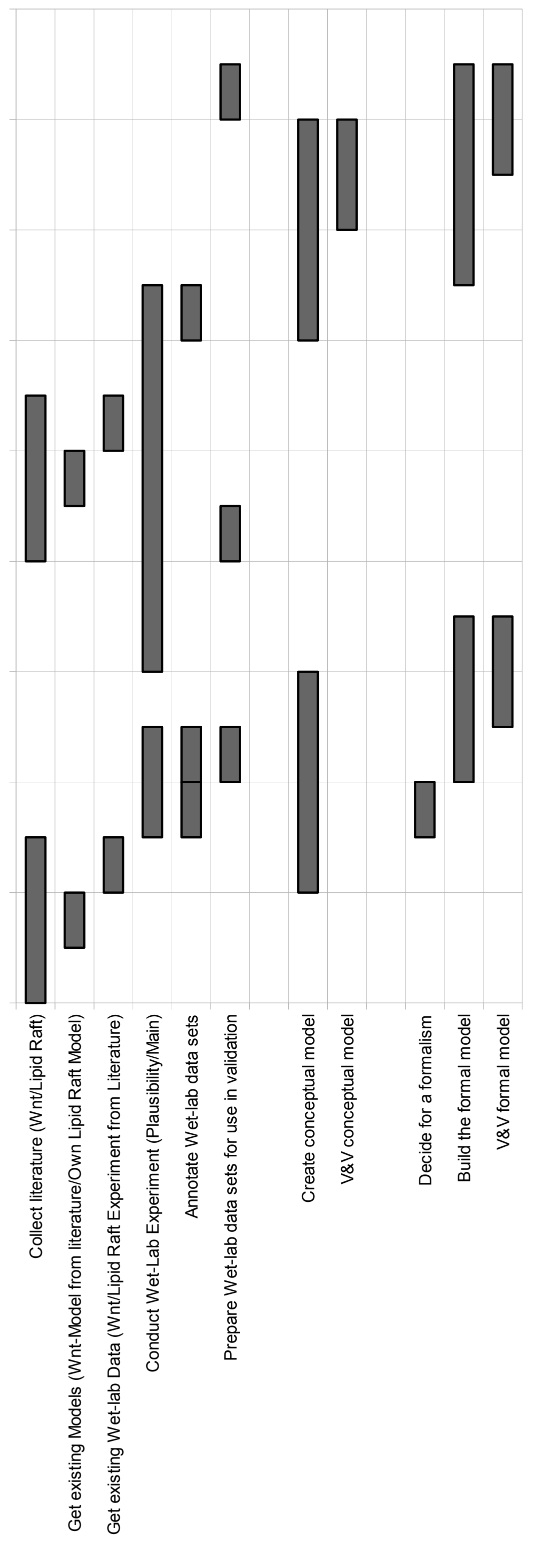


Figure 6.1: Gantt Chart showing the execution of different tasks of the case study over time.

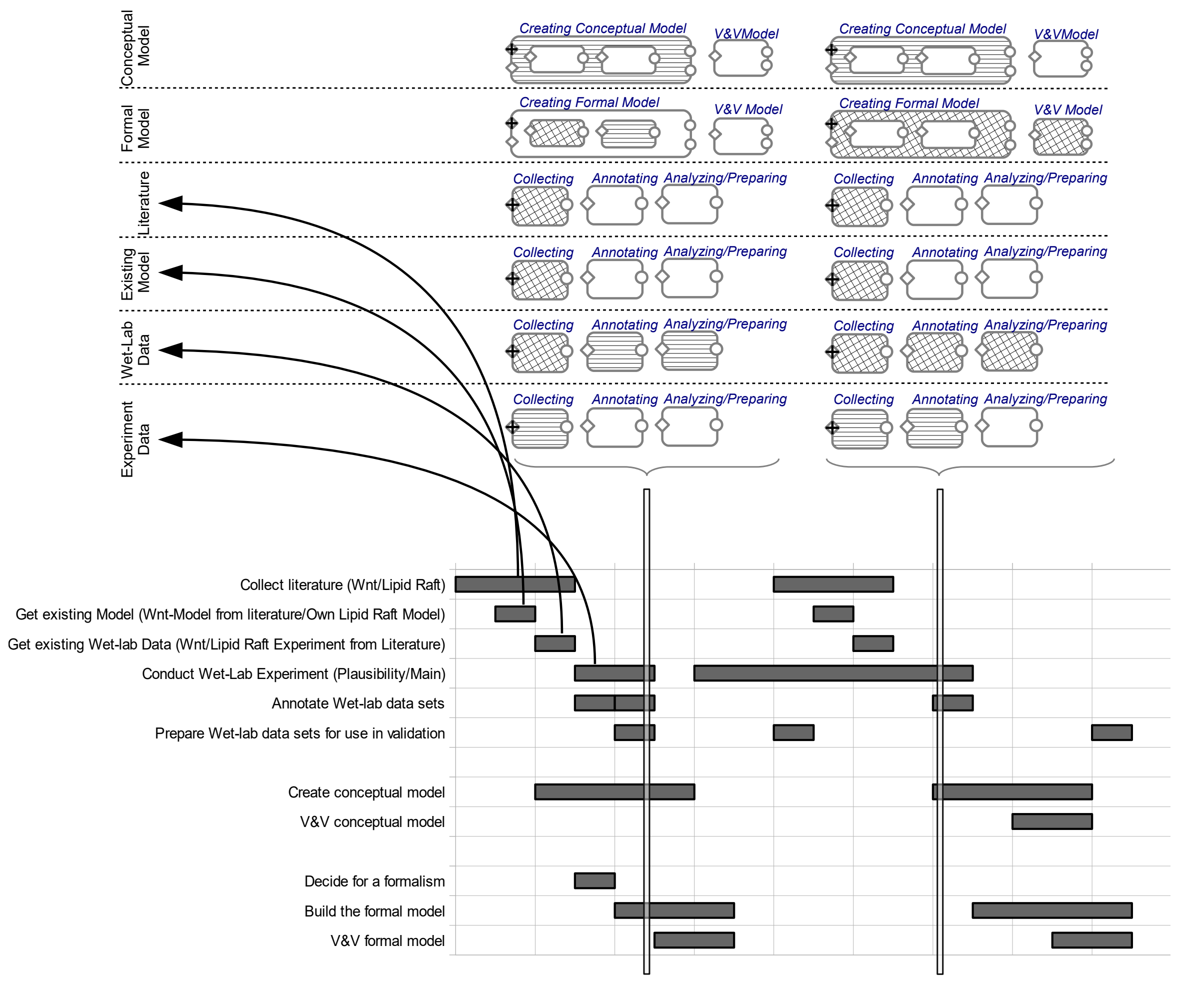

Conceptual Model
Formal Model
Literature
Existing Model
Wet-Lab Data
Experiment Data
Creating Conceptual Model
V&VModel
Creating Formal Model
V&V Model
Collecting
Annotating
Analyzing/Preparing
Collect literature (Wnt/Lipid Raft)
Get existing Model (Wnt-Model from literature/Own Lipid Raft Model)
Get existing Wet-lab Data (Wnt/Lipid Raft Experiment from Literature)
Conduct Wet-Lab Experiment (Plausibility/Main)
Annotate Wet-lab data sets
Prepare Wet-lab data sets for use in validation
Create conceptual model
V&V conceptual model
Decide for a formalism
Build the formal model
V&V formal model

Figure 6.2: Application of artifact-based model to case study; top: states of the artifacts at two selected points in time (Active, inactive and finished stages are filled with horizontal lines, white color and a cross hatched pattern respectively); bottom: tasks from case study over time

model creation workflow was presented for layer one facilitating an artifact-based description (see Section 4.2.1 p. 74).

In order to assess the suitability of the proposed model creation workflow for real world application, an example modeling scenario, originating in the domain of cellular biology, was used for evaluation (Rybacki et al., 2014).

It covers the creation of a simulation model investigating the influence of cell membrane-related processes (lipid rafts) on a specific, intra-cellular signaling pathway (Wnt-pathway). The model is based on existing models, experimental measurements and additional data found in literature. Figure 6.1 shows the activities that were carried out to create the simulation model over time. It should be noted that the model creation process had no workflow support.

The observed activities can be separated into three parts. Firstly, there are activities that dealt with the collection and gathering of data from which the model was derived. The data was also used for verification and validation of the model created as well as for simulation studies conducted later on. Secondly, there are activities dealing with the creation of the conceptual model and its verification and validation, applying the previously gathered data. Lastly, activities for the creation of the formal model based on the conceptual model and its verification and validation were performed.

Overall, this corresponds to the general modeling life-cycle, except that in this scenario the explicit creation of an executable model is missing. This is due to the fact that it can automatically be derived from the formal model.

The example modeling scenario aims at extending existing simulation models of the Wnt-pathway, instead of creating a completely new simulation model. Therefore, literature is researched and information about already existing models, in-vitro data and in-silico data were gathered. Then based on literature and the data available the conceptual model is created. For the conceptual model creation, an existing model was used as reference model. In case there are more than one models to start with available, typically the one that fits the objectives of the new simulation study best is selected.

The next step experimental data for model parameterization was collected and based on the goals and objectives of the simulation study a suitable formalism for the implementation of the formal model was chosen. In this example scenario the formalism chosen was different from the formalism of the selected reference model. Hence, the reference model had to be translated and validated, before the formal model could be built based on the reference model.

While preparing conceptual and formal model a wet-lab experiment was conducted evaluating the given conditions whether they fit the scientific question at hand.

After having successfully transferred and implemented the reference model (conceptual and formal model) to the new formalism the actual extension of the model in order to evaluate the scientific question could be executed. For this, further literature concerning lipid rafts had to be evaluated. Moreover, the newly gathered information was used to extend the conceptual model accordingly. At the same time the formal model was also extended by membrane-related processes (i.e., lipid raft dynamics).

Concurrently, parameterization and information taken from literature, given by in-silico and in-vitro data, had to be annotated and prepared in order to be used for validation as well as for the simulation study. Meanwhile, the previously started wet-lab experiments finshed and V&V was applied to both, conceptual and formal model using face validation, fitting and cross validation techniques.

The model creation process proved to be highly iterative executing the same phases multiple times. It is important to note that some activities took longer than others, e.g., conducting a wet-lab experiment compared to gathering information from literature for revising the model. However, long-term activities typically do not occupy the modeler the entire time giving time to perform other activities, e.g., process and compile data or work on the conceptual or formal model. Moreover, it seems that a typical modeling process comprises a variety of closely intertwined and concurrent activities.

Figure 6.2 p. 186 illustrates how the modeling tasks from the example scenario map to the artifacts. Over the course of the process six artifacts are created, i.e., a *Formal Model*, a *Conceptual Model* and four *Data* artifacts. The four data artifacts resemble artifacts for literature, for the model extracted from literature, for wet-lab data gathered from literature and for data generated by self-conducted wet-lab experiments respectively.

Moreover, the figure also depicts the state of the involved artifacts during the modeling process at two points in time.

The artifact-based workflow fits the described modeling scenario well and is able to reproduce the presented real world scenario. This is not surprising as it bases on well known life-cylce models for the creation of valid and verified simulation models.

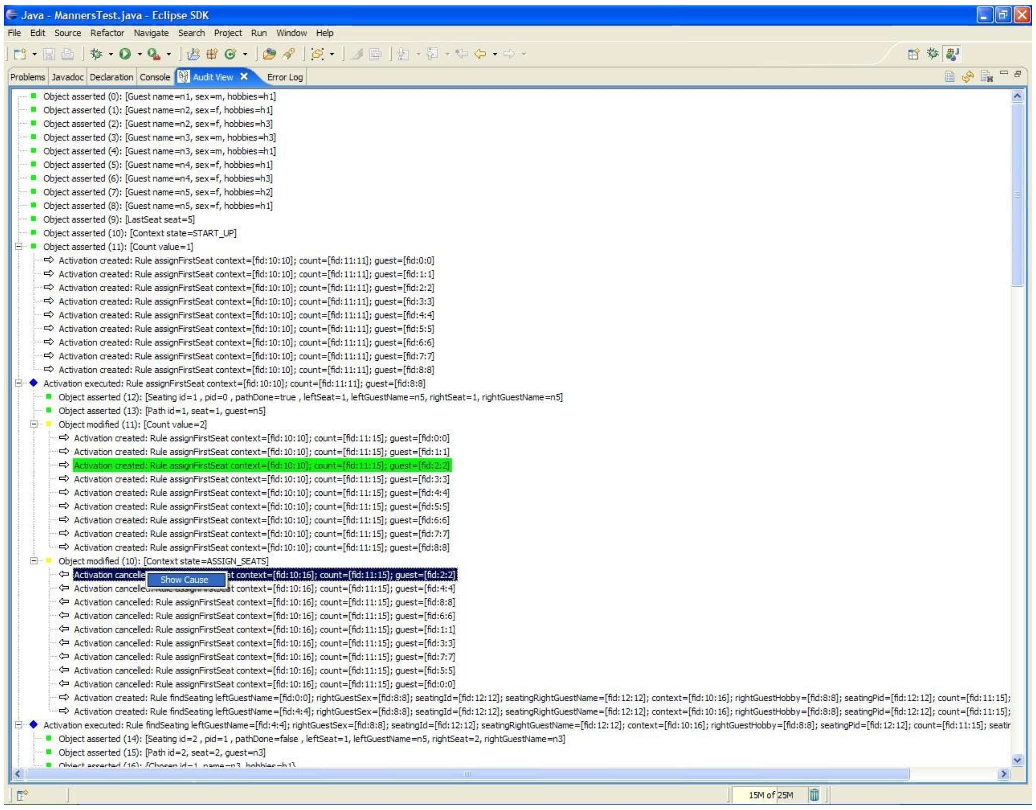
Although, a goal was to provide a process that covers a broad range of different simulation studies, ranging from simple parameter sweeps using an existing model to creating a valid and verified model from scratch which is then used in optimization experiments, it needs to be shown to what extend the presented artifact-based workflow can handle other modeling scenarios. More user and case studies from different fields involving modeling and simulation have to be conducted in order to evaluate its applicability.

6.2.2 Documentation, Provenance & Reproducibility

The aim of using artifact-based workflows is to have a flexible workflow with a clearly defined process to follow in order to provide a certain quality as well as credibility and transparency. A by-product is that it can also be used to guide a scientist through the process of model creation. However, the main reason is to produce a valid and verified model with a documentation and provenance data about the entire model creation process available alongside the model. This is extremely useful to ensure reproducibility aiding to the quality, credibility of the model and transparency, for the further use of the model in simulation experiments.

In order to provide documentation and provenance data the model creation process is monitored and audited each step of the way. This means, that each change to an artifact's data and state (milestones achieved, active stages, etc.) is recorded.

The implementation approach presented in Section 5.1 p. 97 using Drools as driving component provides the infrastructure to easily record those changes. It allows auditing rule executions and changes to facts which directly relates to


Java - MannersTest.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Audit View Error Log
Show Cause
15M of 25M

```
Object asserted (0): [Guest name=n1, sex=m, hobbies=h1]
Object asserted (1): [Guest name=n2, sex=f, hobbies=h1]
Object asserted (2): [Guest name=n2, sex=f, hobbies=h3]
Object asserted (3): [Guest name=n3, sex=m, hobbies=h3]
Object asserted (4): [Guest name=n3, sex=m, hobbies=h1]
Object asserted (5): [Guest name=n4, sex=f, hobbies=h1]
Object asserted (6): [Guest name=n4, sex=f, hobbies=h3]
Object asserted (7): [Guest name=n5, sex=f, hobbies=h2]
Object asserted (8): [Guest name=n5, sex=f, hobbies=h1]
Object asserted (9): [LastSeat seat=5]
Object asserted (10): [Context state=START_UP]
Object asserted (11): [Count value=1]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:0:0]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:1:1]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:2:2]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:3:3]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:4:4]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:5:5]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:6:6]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:7:7]
    Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:8:8]
Activation executed: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:11]; guest=[fid:8:8]
    Object asserted (12): [Seating id=1 , pid=0 , pathDone=true , leftSeat=1, leftGuestName=n5, rightSeat=1, rightGuestName=n5]
    Object asserted (13): [Path id=1, seat=1, guest=n5]
    Object modified (11): [Count value=2]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:0:0]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:1:1]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:2:2]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:3:3]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:4:4]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:5:5]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:6:6]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:7:7]
        Activation created: Rule assignFirstSeat context=[fid:10:10]; count=[fid:11:15]; guest=[fid:8:8]
    Object modified (10): [Context state=ASSIGN_SEATS]
        Activation cancelle                at context=[fid:10:16]; count=[fid:11:15]; guest=[fid:2:2]
        Activation cancelle                at context=[fid:10:16]; count=[fid:11:15]; guest=[fid:4:4]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:8:8]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:6:6]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:1:1]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:3:3]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:7:7]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:5:5]
        Activation cancelled: Rule assignFirstSeat context=[fid:10:16]; count=[fid:11:15]; guest=[fid:0:0]
        Activation created: Rule findSeating leftGuestName=[fid:0:0]; rightGuestSex=[fid:8:8]; seatingId=[fid:12:12]; seatingRightGuestName=[fid:12:12]; context=[fid:10:16]; rightGuestHobby=[fid:8:8]; seatingPid=[fid:12:12]; count=[fid:11:15];
        Activation created: Rule findSeating leftGuestName=[fid:4:4]; rightGuestSex=[fid:8:8]; seatingId=[fid:12:12]; seatingRightGuestName=[fid:12:12]; context=[fid:10:16]; rightGuestHobby=[fid:8:8]; seatingPid=[fid:12:12]; count=[fid:11:15];
Activation executed: Rule findSeating leftGuestName=[fid:4:4]; rightGuestSex=[fid:8:8]; seatingId=[fid:12:12]; seatingRightGuestName=[fid:12:12]; context=[fid:10:16]; rightGuestHobby=[fid:8:8]; seatingPid=[fid:12:12]; count=[fid:11:15]; seatir
    Object asserted (14): [Seating id=2 , pid=1 , pathDone=false , leftSeat=1, leftGuestName=n5, rightSeat=2, rightGuestName=n3]
    Object asserted (15): [Path id=2, seat=2, guest=n3]
    Object asserted (16): {Chosen id=1, name=n3, hobbies=h1}
```

Figure 6.3: Example Drools audit trail of rule executions

data and state changes of artifacts (see Figure 6.3 p. 190). Triggered events are transferred to facts internally and therefore are automatically covered by this audit trail as well as activities directly invoked by Drools rules. However, activities that simply rely on specific facts to hold for them to be executable, e.g., editor actions, need to be monitored using own mechanisms or those provided with JAMES II.

Consequently, from the documentation recorded provenance information could be derived and stored into an appropriate provenance format such (Missier and Goble, 2011; Costa et al., 2013; Missier et al., 2013).

Given the provenance information and documentation available the reproducibility levels 2, 3 and 4 are achievable. Level 1 is hard to achieve as a number of things can happen which is out of reach of the workflow system, mainly the human interaction as well as non-technical activities. For those activities documentation and provenance information purely rely on manual addition to the system.

6.3 Layer Two — Simulation Experiment

6.3.1 Example Simulation Experiment

In order to evaluate the developed simulation experiment template workflow shown in Figure 4.14 p. 93 (see Section 4.2.2) a concrete simulation experiment has been conducted applying the given workflow.

The simulation experiment computes a stochastic model using the ML-Rules formalism. The model describes biochemical reactions in a $MgCl_2$-Solution (Phillips, 2007). The goal is to maximize the number of species Mg in the equilibrium state.

Figure 6.4 p. 192 shows the actual experiment workflow with templates filled with frames that match the specification and objectives of this example experiment. Herein, the specification is given according to the optimization problem. After specifying the experiment, the *Configuration Setup* task deals with the creation of parameter configurations for the model. In order to find the maximum amount of Mg an optimization method is required. Various optimization methods exist in JAMES II, e.g., for optimizing according to single or multiple objectives. Here only one objective is to be optimized and for presentation purposes a standard Hill-Climbing method is used. The simulation algorithm that can also be part of the specification or configuration is not specified further which results in JAMES II selecting an appropriate simulation algorithm. Which simulation algorithm was used eventually will be documented nonetheless.

After the configuration has been executed, the amount of Mg needs to be estimated by calculating the steady state mean of Mg in the model. Typically, there are two ways of estimating the steady state mean in stochastic simulations. Firstly, one estimates the steady state mean of a single simulation replication during the *Single Run Analysis* task, and then calculates the mean of those estimates during the *Multiple Run Analysis* task. Secondly, the single run analysis is skipped in favor of calculating the steady state mean over all replications dur-

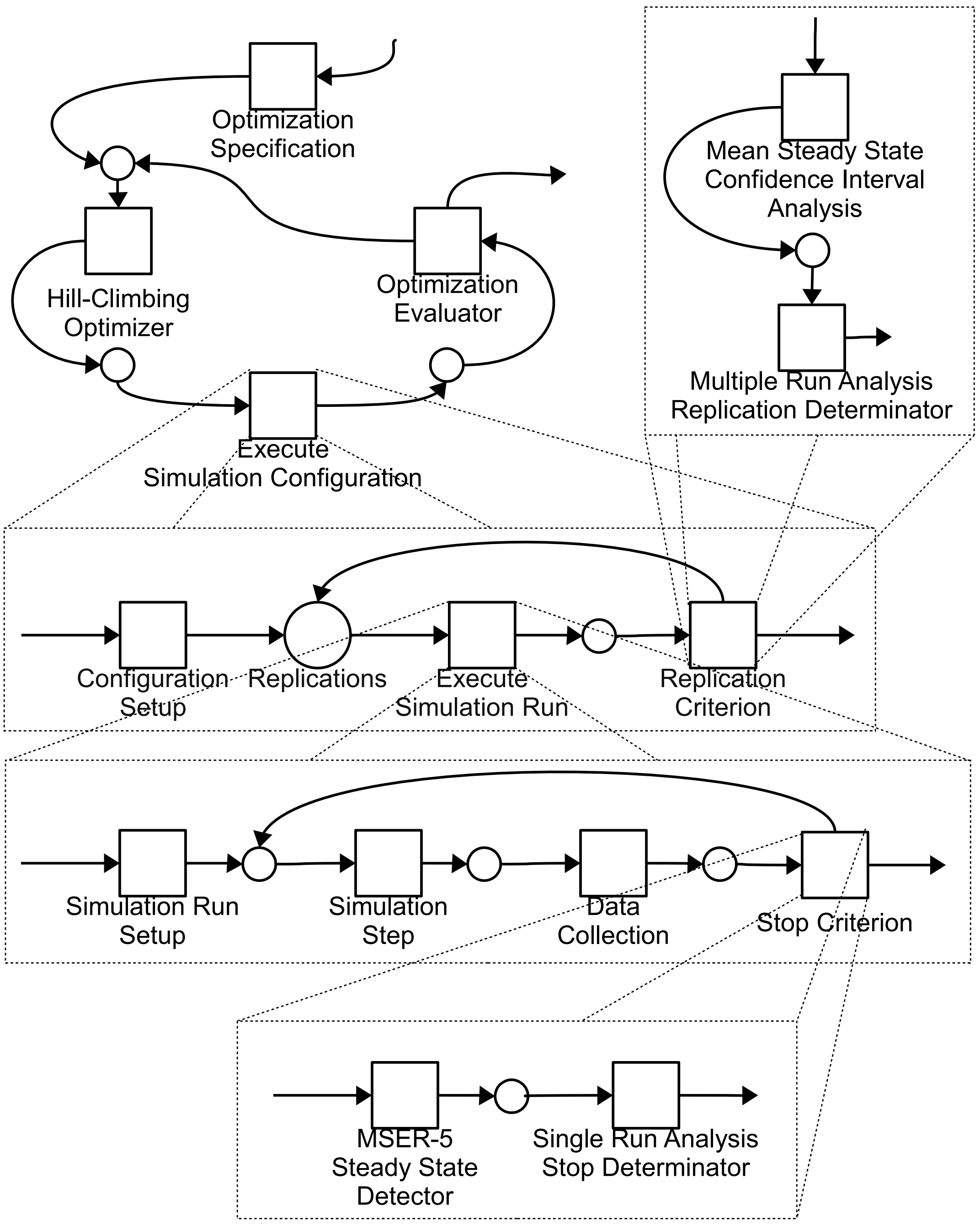

Optimization Specification
Mean Steady State Confidence Interval Analysis
Hill-Climbing Optimizer
Optimization Evaluator
Multiple Run Analysis Replication Determinator
Execute Simulation Configuration
Configuration Setup
Replications
Execute Simulation Run
Replication Criterion
Simulation Run Setup
Simulation Step
Data Collection
Stop Criterion
MSER-5 Steady State Detector
Single Run Analysis Stop Determinator

Figure 6.4: Specific workflow for analyzing the $MgCl_2$-model. All template activities have been assigned an actual frame.

ing the *Multiple Run Analysis* directly. In this example the first approach is applied, for which various techniques exist (Schruben, 1982; White Jr., 1997). From the applicable methods available in JAMES II the *MSER-5* algorithm is selected (White Jr., 1997). This is combined with the *Mean Steady State Confidence Interval method* which is applied to the means calculated by the *MSER-5* algorithm.

The value estimated will then be forwarded to the optimization evaluator which in combination with the optimizer generates more simulation configurations if need be.

Specifying all objectives, methods and algorithms beforehand makes sense if this information is available. However, another option is to not specify, e.g., the optimization method or the steady-state detection algorithm beforehand. It could be specified automatically according to, e.g., previous simulation results or question of interest. That gives the opportunity of an explorative optimization and a quicker jump start on simulation experiments because the entire specification does not need to be known and specified beforehand. Moreover, the specification can be completed automatically once the workflow was executed using its documentation and provenance data.

This in combination with the workflow support helps inexperienced users conduct efficient and effective simulation studies, which are reproducible and for which documentation and provenance information is automatically provided (Leye, 2014).

6.3.2 Documentation, Provenance & Reproducibility

If a simulation experiment is conducted in JAMES II the `BaseExperiment` and its experimentation layer is used. Given that, a quick run on the example simulation experiment produces for one replication of one simulation configuration the console output shown in Listing 6.1 p. 194. As can be seen, the output gives some information about the simulation experiment. For instance, the used simulation algorithm is disclosed as `MLRulesPopulationProcessor` and that it performed 137 310 simulation steps spending 47.92 seconds on them. Additionally, the *IDs* of the experiment and the simulation run are available. They can

Listing 6.1: Console Output produced by the `BaseExperiment` after executing the $MgCL_2$ MLRules model. Not much information can be extracted using this information.

```
Nov 05, 2015 2:16:52 PM
    org.jamesii.simulator.mlrules.population.reference.MLRulesPopulationProcessor cleanUp
    (Line: 1597)
INFO: Simulation steps taken: 137310
Overall time spent processing: 47.92127

Nov 05, 2015 2:16:52 PM org.jamesii.core.experiments.ExecutionMeasurements
    stopComputationTask (Line: 162)
INFO: 802703478-1446729144711-278392507709844-4.802703478-1446729144711-278400701775685-6
    Seconds needed for running the simulation: 48.037

Nov 05, 2015 2:16:52 PM org.jamesii.core.experiments.BaseExperiment runExperiment (Line: 783)
INFO: 802703478-1446729144711-278392507709844-4 Seconds needed for all runs: 59.5

Nov 05, 2015 2:16:52 PM org.jamesii.core.experiments.BaseExperiment execute (Line: 370)
INFO: About to stop the experiment 802703478-1446729144711-278392507709844-4

Nov 05, 2015 2:16:52 PM org.jamesii.core.experiments.BaseExperiment execute (Line: 374)
INFO: Stopped the experiment 802703478-1446729144711-278392507709844-4
```

be used to access data that was stored into a JAMES II data storage (not to mistaken with the herein presented *Data Store*).

Moreover, assuming the `BaseExperiment`'s setup is known information about the computed model, used stop criterion and replication policy can be inferred.

Given this documentation and provenance information a reproducibility level of 4 and maybe 3 can be achieved. When employing additional `IExperimentExecutionListener` and a custom `IExperimentSteerer` that monitor the experiment's execution level 3 is definitely feasible and depending on the complexity of the experiment level 2 could be achieved. But then again this just illustrates how hard it is in JAMES II to achieve documentation using conventional methods, yet alone doing all this for each experiment automatically.

Conducting the simulation experiment using WorMS and the workflow shown in Figure 6.4 p. 192 provides different information regarding available documentation and provenance information. Firstly, the experimentation process is clearly defined and a trace of which work item of the workflow was executed when and with which input and output data is available after execution of the workflow, assuming a *Data Store* that persists tokens and associated data. In case of the use of the `BasicMemoryDataStore` this data is only available as long as the *Data Store* instance is available, which requires an extra storing step of provenance data and documentation based on the information available in the memory datastore.

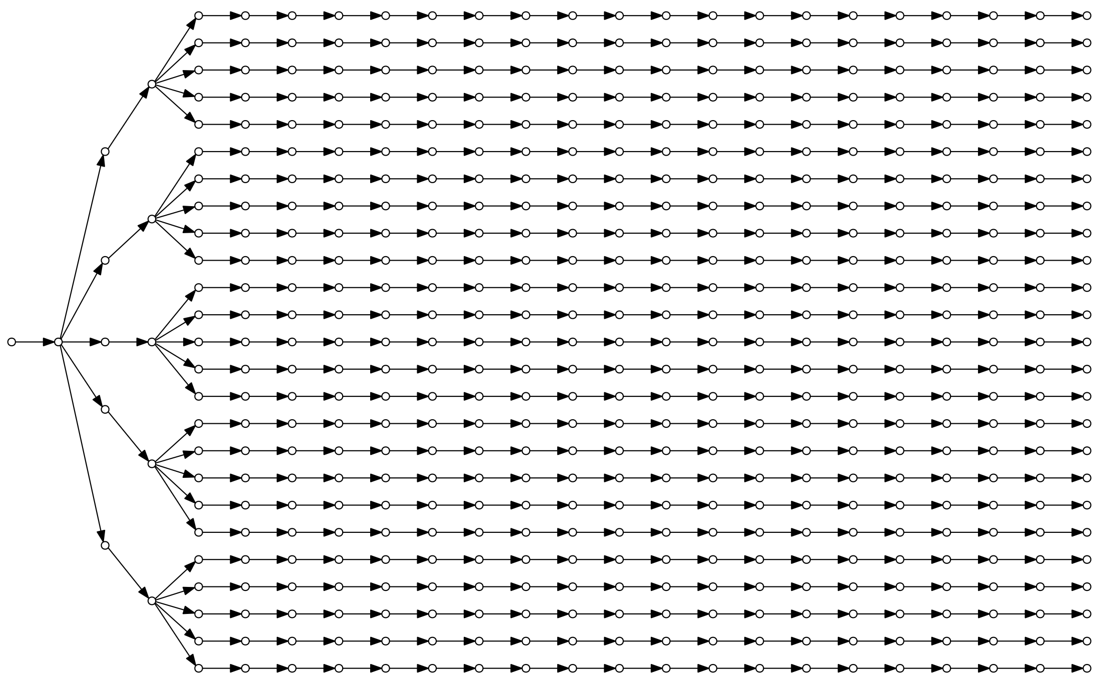


Figure 6.5: Example token graph after execution of the example experiment workflow with five configurations, where each configuration ran five replications and each replication was run for five simulation steps. As expected there are 25 tokens in the end markings available after workflow execution, each representing a finished replication (tokens at the bottom of the graph). Each of those tokens has an execution trace, and traces form groups at different level in the graph. Those levels are the level of configurations (second level), the level of replications (tokens at the 4th level) and level of the start token (top level). In other words, starting with one start token leads to the point where five sub-graphs are spanned, each representing one configuration and within each configuration sub-graph another five sub-graphs are spanned each representing one replication.

Given the token markings of the workflow after execution provides all the information needed to extract provenance data and documentation for executed work items and their input and output data. Each token has a history that comprises all the tokens that were consumed on the way to produce it, including all the data that is associated with each of those tokens. A token spans an acyclic graph of tokens resembling the path it took through the workflow to create it. For instance, in Figure 6.5 p. 195 a token graph is depicted that captures the execution of the example experiment workflow. The experiment conducted five configurations and five replications per configuration, leading to 25 simulation runs in total, where each run was executed five steps (for illustration purposes). Data can then be retrieved analogously to how the WorMS determines input values for a work item (see Section 5.2.1.4 p. 131) by traversing the graph up to the point of interest or data availability (see Figure 5.14 p. 135). Figure 6.6 p. 197 shows the same token graph but also showing for each token the name of the activity the token originated from as well as the data that was produced by the activity and attached to that token. In order to illustrate which data is attached and which activities produced which data Figure 6.7 p. 198 shows a zoomed in version depicting the start token and subsequent tokens. It can be seen that the initial token contains information about model location, stop criterion factory, number of configurations and replications. Additionally, it can be seen that e.g., the execution of the *Configuration Setup* activity leads to a token that has the actual configuration attached. Once a configuration is processed by the *Run Setup* activity, a simulation algorithm and actual stop policy is attached to the resulting token. This leads to the already mentioned token hierarchy and data history, allowing the documentation of each workflow step using the given token graph.

This allows restoring to any point in time of the workflow execution in order to e.g., rerun it from there or rollback during execution due to errors. However, special care needs to be taken if the workflow involves tasks that have side-effects to data outside of the workflow, such as external databases, rerunning a workflow with the very same input values may result in different output values due to the different external data (which is not under the control of the workflow system).

Additionally, as seen in Figure 5.17 p. 144 execution can be monitored to provide even more data. That data can range from execution timings, to de-

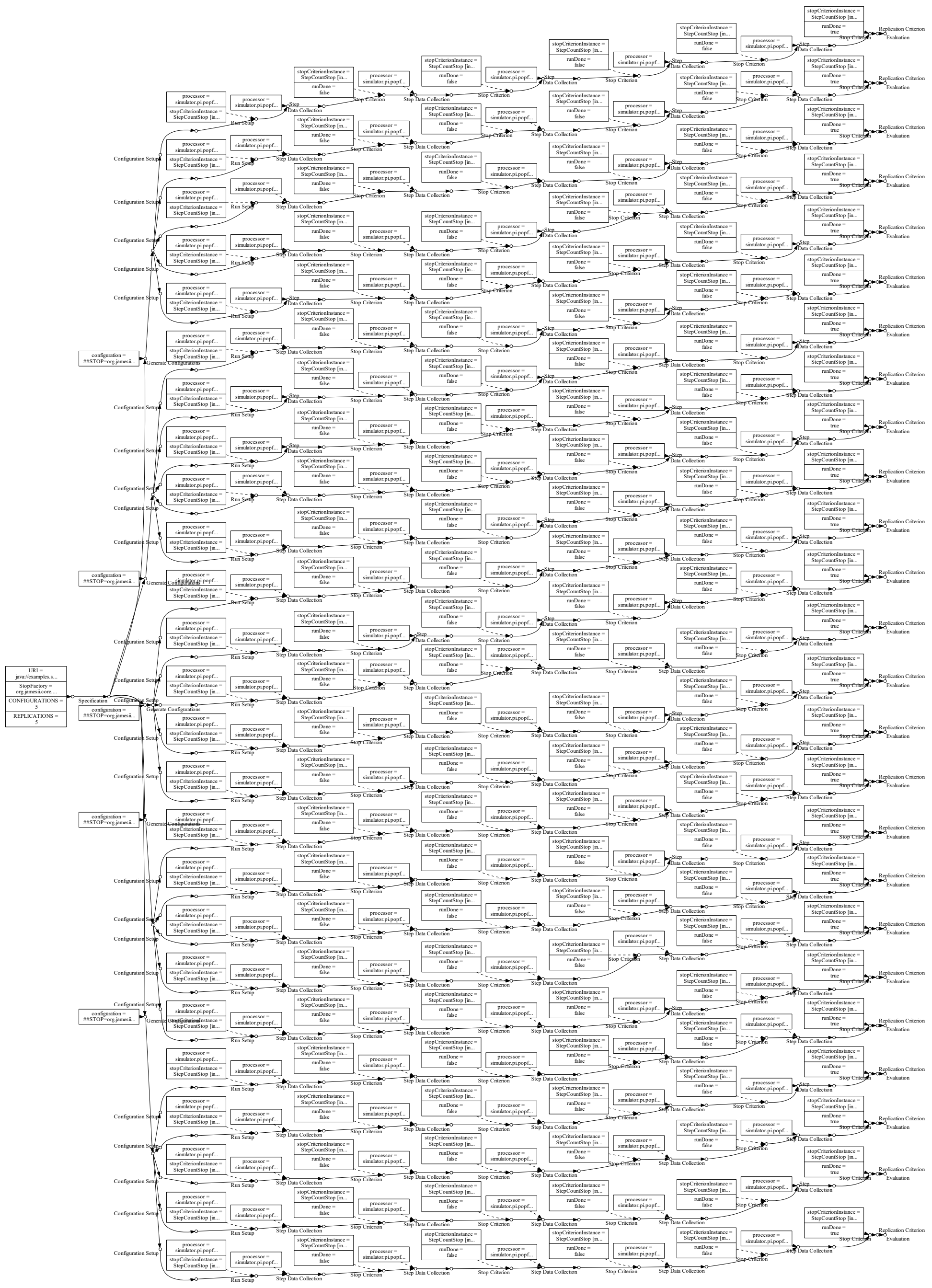

URI = java://examples.s...
StopFactory = org.jamesii.core....
CONFIGURATIONS = 5
REPLICATIONS = 5
Specification
configuration = ##STOP=org.jamesii...
Generate Configurations
Configuration Setup
processor = simulator.pi.popf...
stopCriterionInstance = StepCountStop [in...
Run Setup
Step
Data Collection
runDone = false
runDone = true
Stop Criterion
Step Data Collection
Replication Criterion Evaluation

Figure 6.6: Example token graph for the experiment workflow execution as seen in Figure 6.5 p. 195 with additional annotations showing the token producing activity as well as the produced output data that is attached to that token.

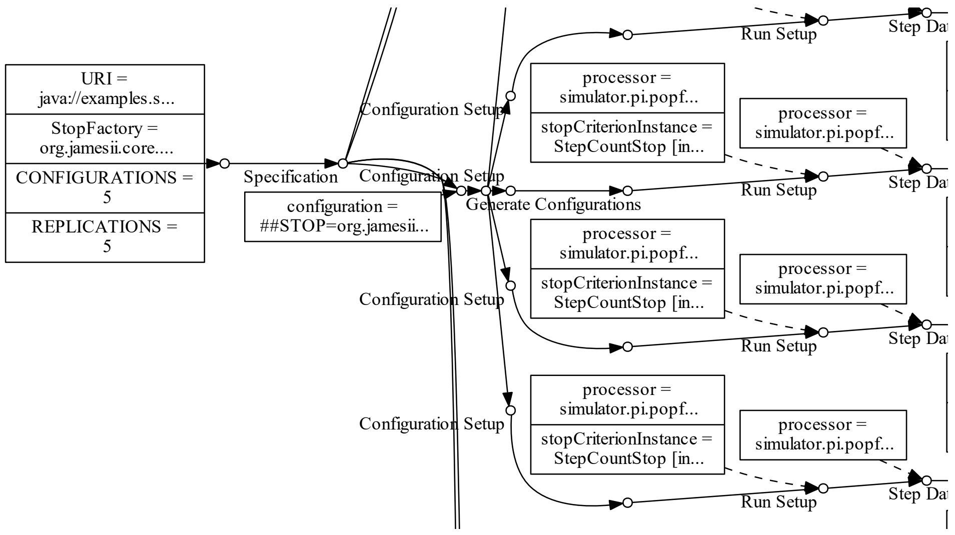

URI = java://examples.s...
StopFactory = org.jamesii.core....
CONFIGURATIONS = 5
REPLICATIONS = 5
Specification
configuration = ##STOP=org.jamesii...
Configuration Setup
Generate Configurations
processor = simulator.pi.popf...
stopCriterionInstance = StepCountStop [in...
Run Setup
Step Da

Figure 6.7: A close up version of a selected part of the token graph shown in Figure 6.6 p. 197. It shows the data attached to the initial token, i.e., the model URI, the stop factory to use, the number of configurations and replications. It also reveals the data attached to subsequent tokens, such as the actual generated configuration, stop criteria and used simulation algorithms.

termining which work items executed concurrently and which workflow worker executed which work item, hence information about where and how work items and data were distributed.

Regarding reproducibility levels, level 2, 3 and 4 are directly achievable using just the provenance and documentation derivable from the tokens and associated data. However, level 1 requires additional information, i.e., work item execution order and ideally the workflow worker used for each work item execution.

It is easy to see, that using WorMS for executing simulation experiments simplifies the creation and derivation of documentation and provenance information about a simulation experiment tremendously by making it transparent and automatic. This information is always available for each and every simulation experiment executed without further hoops to jump through. Nevertheless, if only parts or no information is needed or wanted it can simply be discarded after workflow execution by clearing the *Data Store* and not requesting it. By explicitly needing to clear the *Data Store*, assuming it to be persistent, it is always possible to come back and retrieve documentation and provenance even if not intended previously or directly after workflow execution. Moreover, by always generating this information a scientist cannot accidentally forget to enable documentation generation. However, currently the work item execution order is not automatically tracked and would need to add an explicit monitor in order to achieve level 1 reproducibility.

By using WorMS the workflow can be easily executed in parallel and distributed, according to the setup of WorMS without explicit activation or interaction needed from the experiment conductor. In contrast to using the `BaseExperiment` of JAMES II this does not involve additional efforts regarding documentation.

6.4 Future Work

This work lends itself as starting point of further research and future work. Research and future work can diverge into different directions and range from conducting user and case studies across various scientific domains, over efficient implementation driven work to big data and machine learning domain.

A possible next step would be to implement and integrate the proposed artifact-based workflow management system in JAMES II. Which then opens up the opportunity to conduct extensive user and case studies, testing both the proposed artifact-based workflow and the workflow management system.

The focus of this work lies on providing documentation and provenance information, what is needed is a report generation facility which generates readable (for machine as well as for humans) documentations which can be easily attached to published data and results. In order to provide machine readable documentation and provenance data it makes sense to provide them in an open standardized format such as the Open Provenance Model (OPM) and others (Missier and Goble, 2011; Costa et al., 2013; Missier et al., 2013).

Persisting provenance and documentation data is important for rerunning and reproducing experiments based on that information. However, how reruns can be efficiently handled and implemented is to be researched. Questions arise on how to handle rerun and existing information from previous runs. What if results diverge, how to deal with changing environments and how to manage the amount of data that can build up when running experiments with a lot of data, replications and configurations involved are questions that need to be considered and open up a broad area of future work to pursue.

As shown the distribution and scheduling of activities smartly and efficiently is important to the overall performance of the workflow execution. The presented adaptive scheduling approach using machine learning is open for improvement regarding delivered performance. For instance, the use of deep learning approaches could be used to automatically select feature sets that are then used by the reinforcement learning method, removing the need to select a suitable set of features manually.

In concert with the previous ideas, an easier mechanism of setting up workflow worker actors on other nodes could be investigated. An important and challenging requirement is that all workflow worker actors work with the same versions of software, e.g., specific versions of Java classes and libraries, per workflow, the problem is elevated when software versions change from workflow to workflow. This will need a dynamic isolation mechanism and exchange mechanism that can switch software per workflow on demand, as workflow worker actors can be shared across multiple concurrent workflow executions.

Once provenance data can be efficiently and standardized persisted the connection to repositories, such as the myExperiment platform can be established. However, workflow decay and workflow versioning pose a problem when it comes to storing this information over time. Decay refers to the loss of the ability of re-executing or reproducing a workflow execution over time, typically due to the volatility of resources involved and required in the previous execution (Zhao et al., 2012; Hettne et al., 2012). Counteracting the decay of workflows is a large field of research and is interesting direction of research for future work.

7 Summary

All generalizations are false,
including this one.

Mark Twain

Credible and high quality scientific results are usually backed by a well-defined and documented process and provide documentation and provenance information about their creation, i.e., providing enough information for making them reproducible. Documentation and provenance information is not always provided which led to a *Crisis of Credibility* of scientific results. If documentation needs to be done manually, it is likely to be incomplete or not as extensive at it could be. In this work a workflow approach was taken to tackle the problem of automatable documentation and provenance information generation for conducting simulation studies with the Modeling & Simulation framework JAMES II.

Before using workflows, the general process of a simulation study needed to be analyzed, deriving properties of the workflows later developed. However, the analysis revealed that it makes sense that a simulation study is divided into two layers, because both layers put different demands on the workflows and their systems. Layer one involves the creation of the simulation model that is used throughout the simulation study. It is a highly human driven process requiring a lot of flexibility and freedom during execution. Layer two covers the technical part of the simulation study, i.e., the execution of a simulation experiment. In contrast to layer one, layer two is mainly an automatable rigid process.

For each layer a separate workflow approach was used to cope with the demands of that layer. For layer one an artifact-based declarative workflow was introduced, while for layer two a task-based imperative workflow using a workflow net derivative was employed.

The artifact-based workflow approach used in this work allowed for the definition of a generic workflow representing the creation of a valid and verified simulation model. The workflow model bases on established Life-Cycle models found in literature in order to comply to best practices of model creation in the domain of Modeling & Simulation. For the management and execution of such a workflow an architecture was presented, that uses a rule-based system internally for the evaluation and execution of artifacts and their tasks and actions. It allows for recording rule application and fact changes over time, providing provenance and documentation automatically for each execution.

The simulation experiment execution workflow developed in this work provides a generic description of a wide range of simulation experiments, e.g., parameter scan, optimization and validation experiments. In order to achieve this generic description workflow nets were extended by the concept of templates and frames allowing placeholder tasks, which can be filled on demand during execution with actual tasks. To evaluate this workflow a prototypical framework for workflow execution and management, called WorMS, specifically targeted at Modeling & Simulation and JAMES II was developed. It features a component-based architecture, allowing easy exchange and extension of components and features. Execution can automatically be parallelized and distributed if desired. Scheduling of work items during parallel or distributed execution is handled by a herein developed adaptive scheduling policy that uses machine learning techniques to improve its distribution scheme in order to maximize performance. Workflow executions in WorMS can be monitored using the *Monitoring* component, allowing recording and collecting arbitrary data, such as execution statistics, input/output data of work items and used JAMES II plug-ins including their parameters.

Since both workflow architectures provide means to monitor and collect information about their workflow execution automatically, this work paves the way towards reproducible simulation studies with JAMES II. This eventually allows the answering of questions such as *What task was executed when, with what data* *and what was the output?* and *What was used, such as software, hardware, system configuration or algorithm to execute a specific task?*, leading to credible and high quality scientific results.

Bibliography

Accorsi, R. and Wonnemann, C. (2010). Auditing workflow executions against dataflow policies. In *BUSINESS INFORMATION SYSTEMS*, volume 47, pages 207–217. Springer Berlin Heidelberg.

Agha, G. (1986). Actors: a model of concurrent computation in distributed systems, series in artificial intelligence. *MIT Press*, 11(12):12.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., and Mock, S. (2004). Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 423–424. IEEE.

Anand, M. K., Bowers, S., Mcphillips, T., and Ludäscher, B. (2009). Exploring scientific workflow provenance using hybrid queries over nested data and lineage graphs. In *Scientific and Statistical Database Management*, pages 237–254. Springer.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al. (2003). Business process execution language for web services. http://xml.coverpages.org/BPELv11-20030505-20030331-Diffs.pdf. Accessed: May 3, 2016.

Balci, O. (1998). Verification, validation, and accreditation. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, pages 41–4, Los Alamitos, CA, USA. IEEE Computer Society Press.

Balci, O. (2003). Verification, validation, and certification of modeling and simulation applications: Verification, validation, and certification of modeling and simulation applications. In *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation*, WSC '03, pages 150–158. Winter Simulation Conference.

Balci, O. (2004). Quality assessment, verification, and validation of modeling and simulation applications. In *Simulation Conference, 2004. Proceedings of the 2004 Winter*, volume 1, page 129. IEEE.

Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., Grochow, K., and Lazowska, E. (2008a). Trident: Scientific workflow workbench for oceanography. *Services, IEEE Congress on*, 0:465–466.

Barga, R., Jackson, J., Araujo, N., Guo, D., Gautam, N., and Simmhan, Y. (2008b). The trident scientific workflow workbench. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 317–318, Washington, DC, USA. IEEE Computer Society.

Barga, R. S. and Digiampietri, L. A. (2008). Automatic capture and efficient storage of e-science experiment provenance. *Concurrency and Computation: Practice and Experience*, 20(5):419–429.

Bell, T. E. and Thayer, T. A. (1976). Software requirements: Are they really a problem? In *Proceedings of the 2nd international conference on Software engineering*, pages 61–68. IEEE Computer Society Press.

Binkele, P. and Schmauder, S. (2003). An atomistic monte carlo simulation of precipitation in a binary system. *Zeitschrift für Metallkunde*, 94(8):858–863.

Blatt, M. and Bastian, P. (2007). *The Iterative Solver Template Library*, pages 666–675. Springer Berlin Heidelberg, Berlin, Heidelberg.

Boley, H., Paschke, A., and Shafiq, O. (2010). Ruleml 1.0: the overarching specification of web rules. *Lecture Notes in Computer Science*, 6403(4):162–178.

Bowers, S., Ludascher, B., Ngu, A., and Critchlow, T. (2006). Enabling scientific-workflow reuse through structured composition of dataflow and control-flow. In *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, pages 70–70. IEEE.

Boyer, M. J. and Mili, H. (2011). Ibm websphere ilog jrules. In *Agile Business Rule Development*, pages 215–242. Springer.

Brade, D. (2004). *A Generalized Process for the Verification and Validation of Models and Simulation Results*. PhD thesis, Universität der Bundeswehr München.

Center of Excellence (2015). Business process management - glossary. https://www.ftb.ca.gov/aboutFTB/Projects/ITSP/BPM_Glossary.pdf. Accessed: May 3, 2016.

Chaudhuri, S. (1998). An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43. ACM.

Chwif, L., Banks, J., de Moura Filho, J. P., and Santini, B. (2013). A framework for specifying a discrete-event simulation conceptual model. *Journal of Simulation*, 7(1):50–60.

Cohn, D. and Hull, R. (2009). Business artifacts: A data-centric approach to modeling business operations and processes. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 32(3):3–9.

Concepcion, A. I. and Zeigler, B. (1988). Devs formalism: A framework for hierarchical model development. *IEEE Transactions on Software Engineering*, 14(2):228.

Costa, F., Silva, V., de Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J., and Mattoso, M. (2013). Capturing and querying workflow runtime provenance with prov: A practical approach. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 282–289, New York, NY, USA. ACM.

Curcin, V. and Ghanem, M. (2008). Scientific workflow systems-can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, pages 1–9. IEEE.

Dadam, P. and Reichert, M. (2009). The adept project: a decade of research and development for robust and flexible process support. *Computer Science-Research and Development*, 23(2):81–97.

Dadam, P., Reichert, M., Rinderle-Ma, S., Lanz, A., Pryss, R., Predeschly, M., Kolb, J., Ly, L. T., Jurisch, M., Kreher, U., et al. (2010). From adept to aristaflow bpm suite: a research vision has become reality. In *Business process management workshops*, pages 529–531. Springer.

Dalle, O. (2012). On reproducibility and traceability of simulations. In *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pages 1–12. IEEE.

Davidson, S. B. and Freire, J. (2008). *Provenance and scientific workflows.* ACM Press, New York, New York, USA.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220.

Decker, G., Dijkman, R., Dumas, M., and García-Bañuelos, L. (2008). Transforming bpmn diagrams into yawl nets. In *Business Process Management*, pages 386–389. Springer.

Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540.

Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., et al. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237.

Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., and Zbyslaw, A. (1996). Freeflow: mediating between representation and action in workflow systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 190–198. ACM.

Downing, T. B. (1998). *Java RMI: remote method invocation.* IDG Books Worldwide, Inc.

Dyer, L., Henry, F., Lehmann, I., Lipof, G., Osmani, F., Parrott, D., Peeters, W., Zahn, J., et al. (2012). *Scaling BPM Adoption: From Project to Program with IBM Business Process Manager.* IBM Redbooks.

Eckermann, O. and Weidlich, M. (2011). Flexible artifact-driven automation of product design processes. In *Enterprise, Business-Process and Information Systems Modeling*, pages 103–117. Springer.

Ellis, C. A. and Nutt, G. J. (1993). Modeling and enactment of workflow systems. In *International Conference on Application and Theory of Petri Nets*, pages 1–16. Springer.

Ewald, R. (2012). *Automatic Algorithm Selection for Complex Simulation Problems*. Springer Science & Business Media.

Ewald, R., Himmelspach, J., Jeschke, M., Leye, S., and Uhrmacher, A. M. (2010a). Flexible experimentation in the modeling and simulation framework james ii–implications for computational systems biology. *Briefings in bioinformatics*, 11(3):290–300.

Ewald, R., Himmelspach, J., and Uhrmacher, A. M. (2008). An algorithm selection approach for simulation systems. In *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*, pages 91–98, Rome, Italy. Ieee.

Ewald, R., Schulz, R., and Uhrmacher, A. M. (2010b). Selecting simulation algorithm portfolios by genetic algorithms. In *24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation, PADS 2010, Atlanta, Georgia, USA, May 17-19, 2010*, pages 48–56. IEEE.

Ewald, R. and Uhrmacher, A. M. (2014). Sessl: a domain-specific language for simulation experiments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 24(2):11.

Feinberg, A. (2011). Project voldemort: Reliable distributed storage. In *Proceedings of the 10th IEEE International Conference on Data Engineering*. IEEE.

Fomel, S. and Claerbout, J. F. (2009). Reproducible research. *Computing in Science & Engineering*, 11(1):5–7.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, 19(1):17–37.

Freire, J., Silva, C. T., Callahan, S. P., Santos, E., Scheidegger, C. E., and Vo, H. T. (2006). Managing rapidly-evolving scientific workflows. In *Provenance and Annotation of Data*, pages 10–18. Springer.

Friedman-Hill, E. (2003). *JESS in Action*, volume 46. Manning Greenwich, CT.

Fritz, C., Hull, R., and Su, J. (2009). Automatic construction of simple artifact-based business processes. In *Proceedings of the 12th International Conference on Database Theory*, pages 225–238. ACM.

Fujimoto, R. M. (2000). *Parallel and distributed simulation systems*, volume 300. Wiley New York.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional.

Garvin, D. A. (1984). What does "product quality" really mean? *Sloane Management Review*, 26(1):25–43.

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G. C., Gannon, D., Goble, C., Livny, M., Moreau, L., and Myers, J. (2007). Examining the challenges of scientific workflows. *Computer*, 40(12):24–32.

Gilles, K. (1974). The semantics of a simple language for parallel programming. *In Information Processing*, 74:471–475.

Goble, C. and De Roure, D. (2009). The impact of workflow tools on data-centric research. http://eprints.ecs.soton.ac.uk/17336/2/workflows-submitted.pdf.

Goble, C. A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., Borkum, M., Bechhofer, S., Roos, M., Li, P., and De Roure, D. (2010). myexperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic acids research*, 38(Web Server issue):W677–82.

Goodman, A., Pepe, A., Blocker, A. W., Borgman, C. L., Cranmer, K., Crosas, M., Di Stefano, R., Gil, Y., Groth, P., Hedstrom, M., et al. (2014). Ten simple rules for the care and feeding of scientific data. *PLoS Comput Biol*, 10(4):e1003542.

Görlach, K., Sonntag, M., Karastoyanova, D., Leymann, F., and Reiter, M. (2011). Conventional workflow technology for scientific simulation. In Yang, X., Wang, L., and Jie, W., editors, *Guide to e-Science*, Computer Communications and Networks, pages 323–352–352. Springer London, London.

Gupta, M. (2012). *Akka essentials.* Packt Publishing Ltd.

Heavey, C. and Ryan, J. (2006). Process modelling support for the conceptual modelling phase of a simulation project. In Perrone, L. F., Lawson, B., Liu, J., and Wieland, F. P., editors, *Winter Simulation Conference*, pages 801–808. ACM.

Helms, T., Ewald, R., Rybacki, S., and Uhrmacher, A. M. (2013). A generic adaptive simulation algorithm for component-based simulation systems. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM PADS '13, pages 11–22, New York, NY, USA. ACM.

Helms, T., Ewald, R., Rybacki, S., and Uhrmacher, A. M. (2015). Automatic runtime adaptation for component-based simulation algorithms. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 26(1):7.

Hettne, K. M., Wolstencroft, K., Belhajjame, K., Goble, C. A., Mina, E., Dharuri, H., De Roure, D., Verdes-Montenegro, L., Garrido, J., and Roos, M. (2012). Best practices for workflow design: How to prevent workflow decay. In *SWAT4LS.*

Himmelspach, J. (2009). Toward a collection of principles, techniques, and elements of modeling and simulation software. In *2009 First International Conference on Advances in System Simulation*, pages 56–61. IEEE.

Himmelspach, J. (2012). Tutorial on building m&s software based on reuse. In *Simulation Conference (WSC), Proceedings of the 2012 Winter*, pages 1–15. IEEE.

Himmelspach, J., Ewald, R., and Uhrmacher, A. M. (2008). A flexible and scalable experimentation layer. In *Proceedings of the 40th Conference on Winter Simulation*, WSC '08, pages 827–835. Winter Simulation Conference.

Himmelspach, J. and Uhrmacher, A. M. (2009). What contributes to the quality of simulation results? In Lee, L. H., Kuhl, M. E., Fowler, J. W., and Robinson, S., editors, *Proceedings of the 2009 INFORMS Simulation Society Research Workshop*, pages 125–129, University of Warwick, Coventry, U.K. INFORMS Simulation Society.

Hinz, S., Schmidt, K., and Stahl, C. (2005). Transforming bpel to petri nets. In van der Aalst, W. M. P., Benatallah, B., Casati, F., and Curbera, F., editors, *Proceedings of the Third International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*, pages 220–235. Springer-Verlag.

Hollingsworth, D. et al. (2004). The workflow reference model: 10 years on. In *Fujitsu Services, UK; Technical Committee Chair of WfMC*. Citeseer.

Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., and Kummer, U. (2006). Copasi - a complex pathway simulator. *Oxford Journal - Bioinformatics*, 22(24):3067–3074.

Horng, J.-T., Kao, C.-Y., and Liu, B.-J. (1994). A genetic algorithm for database query optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 350–355. IEEE.

Hoyle, D. (2009). *ISO 9000 Quality Systems Handbook: Using the standards as a framework for business improvement*. Routledge.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., and Oinn, T. (2006). Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):729–732.

Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath, III, F. T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P. N., and Vaculin, R. (2011). Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *Proceedings of the 5th ACM International Conference on Distributed Event-based System*, DEBS '11, pages 51–62, New York, NY, USA. ACM.

InfiniBand Trade Association (2000). *InfiniBand Architecture Specification: Release 1.0.* InfiniBand Trade Association.

International Organization for Standardization (ISO) (2008). Iso 9001:2008 - quality management systems – requirements. http://www.iso.org/iso/catalogue_detail?csnumber=46486. Accessed: May 3, 2016.

Jablonski, S. (2010). Do we really know how to support processes? considerations and reconstruction. In Engels, G., Lewerentz, C., Schäfer, W., Schürr, A., and Westfechtel, B., editors, *Graph Transformations and Model-Driven Engineering*, volume 5765 of *Lecture Notes in Computer Science*, pages 393–410–410. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jahnes, S. and Schüttenhelm, T. (2008). *ISO 9001: Die Revision 2008.* WEKA Media.

Jamil, H. and El-Hajj-Diab, B. (2008). Bioflow: A web-based declarative workflow language for life sciences. In *Services-Part I, 2008. IEEE Congress on*, pages 453–460. IEEE.

Jensen, K. (1986). Coloured petri nets. In Brauer, W., Reisig, W., and Rozenberg, G., editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer.

Jensen, K., Kristensen, L. M., and Wells, L. (2007). Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254.

John, M., Lhoussaine, C., Niehren, J., and Uhrmacher, A. M. (2008). The attributed pi calculus. In *Computational Methods in Systems Biology*, pages 83–102. Springer.

Johns, M. (2013). *Getting Started with Hazelcast.* Packt Publishing Ltd.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

Ko, R. K., Lee, S. S., and Lee, E. W. (2009). Business process management (bpm) standards: a survey. *Business Process Management Journal*, 15(5):744–791.

Kossow, C., Rybacki, S., Millat, T., Rateitschak, K., Jaster, R., Uhrmacher, A. M., and Wolkenhauer, O. (2015). An explicit numerical scheme to efficiently simulate molecular diffusion in environments with dynamically changing barriers. *Mathematical and Computer Modelling of Dynamical Systems*, 21(6):535–559.

Kotek, J. (2015). Mapdb. http://www.mapdb.org/. Accessed: May 3, 2016.

Koutsonikola, V. (2004). Ldap: Framework, practices, and trends. *IEEE Internet Computing*, 8(5):66–72.

Kurkowski, S. (2006). *Credible Mobile Ad Hoc Network Simulation-Based Studies.* Phd thesis, Colorado School of Mines.

Kurkowski, S., Camp, T., and Colagrosso, M. (2005). Manet simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61.

Laddaga, R. and Veitch, J. (1997). Dynamic object technology. *Communications of the ACM*, 40(5):36–38.

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

Lattner, A. D. (2013). Towards automation of simulation studies - artificial intelligence methodologies for the control and analysis of simulation experiments. *KI*, 27(3):287–290.

Law, A. M. and Kelton, D. M. (2007). *Simulation modeling and analysis*. McGraw-Hill International, 4 edition.

Le Novere, N., Bornstein, B., Broicher, A., Courtot, M., Donizelli, M., Dharuri, H., Li, L., Sauro, H., Schilstra, M., Shapiro, B., et al. (2006). Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic acids research*, 34(suppl 1):D689–D691.

Lee, E. A. and Messerschmitt, D. G. (1987). Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245.

Lee, E. A. and Neuendorffer, S. (2007). Tutorial: Building ptolemy ii models graphically. Technical Report UCB/EECS-2007-129, EECS Department, University of California, Berkeley.

Lee, E. A. and Parks, T. M. (1995). Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801.

Lehmann, A. (2008). Expanding the v-modell® xt for verification and validation of modelling and simulation applications. In *2008 Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing*, pages 404–410. Asia Simulation Conference, Ieee.

Leye, S. (2014). *Toward Guiding Simulation Experiments*. PhD thesis, University of Rostock, Germany.

Leye, S., Mazemondet, O., and Uhrmacher, A. M. (2010). Parallel analysis with famval to speed up simulation-based model checking. In *Computer Modeling and Simulation (EMS), 2010 Fourth UKSim European Symposium on*, pages 344–350. IEEE.

Leye, S. and Uhrmacher, A. M. (2010). A flexible and extensible architecture for experimental model validation. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 65:1–65:10, ICST. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Liang, S., Fodor, P., Wan, H., and Kifer, M. (2009). Openrulebench: an analysis of the performance of rule engines. In *Proceedings of the 18th international conference on World wide web*, pages 601–610. ACM.

Lin, C., Lu, S., Fei, X., Pai, D., and Hua, J. (2009). A task abstraction and mapping approach to the shimming problem in scientific workflows. In *2009 IEEE International Conference on Services Computing*, pages 284–291. IEEE.

Littauer, R., Ram, K., Ludäscher, B., Michener, W., and Koskela, R. (2012). Trends in use of scientific workflows: insights from a public repository and recommendations for best practice. *International Journal of Digital Curation*, 7(2):92–100.

Liu, J., Perrone, L. F., Nicol, D. M., Liljenstam, M., Elliott, C., and Pearson, D. (2001). Simulation modeling of large-scale ad-hoc sensor networks. In *European Simulation Interoperability Workshop*, volume 200.

Lohmann, N. (2008). A feature-complete petri net semantics for ws-bpel 2.0. In Dumas, M. and Heckel, R., editors, *Web Services and Formal Methods*, volume 4937 of *Lecture Notes in Computer Science*, pages 77–91–91. Springer Berlin Heidelberg, Berlin, Heidelberg.

Luboschik, M., Rybacki, S., Ewald, R., Schwarze, B., Schumann, H., and Uhrmacher, A. (2012). Interactive visual exploration of simulator accuracy: a case study for stochastic simulation algorithms. In *Proceedings of the Winter Simulation Conference*, page 419. Winter Simulation Conference, IEEE.

Luboschik, M., Rybacki, S., Haack, F., and Schulz, H.-J. (2014). Supporting the integrated visual analysis of input parameters and simulation trajectories. *Computers & Graphics*, 39:37 – 47.

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. (2006). Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065.

Ludäscher, B., Weske, M., McPhillips, T., and Bowers, S. (2009). Scientific workflows: business as usual? In *Business Process Management*, pages 31–47. Springer.

Mangan, P. and Sadiq, S. (2002). On building workflow models for flexible processes. In *Proceedings of the 13th Australasian Database Conference - Volume 5*, ADC '02, pages 103–109, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Mao, C. (2010). Control flow complexity metrics for petri net-based web service composition. *Journal of Software*, 5(11):1292–1299.

MathWorks, I. (2005). *MATLAB: the language of technical computing. Desktop tools and development environment, version 7*, volume 9. MathWorks.

Maus, C., Rybacki, S., and Uhrmacher, A. (2011). Rule-based multi-level modeling of cell biological systems. *BMC systems biology*, 5(1):166.

McLean, T. and Fujimoto, R. (2000). Repeatability in real-time distributed simulation executions. In *Proceedings of the fourteenth workshop on Parallel and distributed simulation*, pages 23–32. IEEE Computer Society.

Merali, Z. (2010). Error: why scientific programming does not compute. *Nature*, 467(7317):775–777.

Merz, M., Moldt, D., Müller, and Lamersdorf, W. (1995). Workflow modelling and execution with coloured petri nets in cosm. In *Workshop on Applications of Petri Nets to Protocols, Proceedings 16th International Conference on Application and Theory of Petri Nets, 1995.* Universität Hamburg.

Migliorini, S., Gambini, M., Rosa, M. L., and ter Hofstede, A. H. (2011). Pattern-based evaluation of scientific workflow management systems. http://eprints.qut.edu.au/39935/.

Miles, S., Groth, P., Deelman, E., Vahi, K., Mehta, G., and Moreau, L. (2008). Provenance: The bridge between experiments and data. *Computing in Science & Engineering*, 10(3):38–46.

Miller, S. P., Neuman, B. C., Schiller, J. I., and Saltzer, J. H. (1987). Kerberos authentication and authorization system. In *In Project Athena Technical Plan*. Citeseer.

Missier, P., Dey, S., Belhajjame, K., Cuevas-Vicenttin, V., and Ludäscher, B. (2013). D-prov: Extending the prov provenance model with workflow structure. In *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13)*, Lombard, IL. USENIX Association.

Missier, P., Embury, S., and Stapenhurst, R. (2008). Exploiting provenance to make sense of automated decisions in scientific workflows. In Freire, J., Koop, D., and Moreau, L., editors, *Provenance and Annotation of Data and Processes*, volume 5272 of *Lecture Notes in Computer Science*, pages 174–185. Springer Berlin Heidelberg, Berlin, Heidelberg.

Missier, P. and Goble, C. (2011). Workflows to open provenance graphs, round-trip. *Future Generation Computer Systems*, 27(6):812 – 819.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Nance, R. (1986). The conical methodology: A framework for simulation model development. Technical Report SRC-87-002, Virginia Institute of Technology.

Ngamakeur, K., Yongchareon, S., and Liu, C. (2012). A framework for realizing artifact-centric business processes in service-oriented architecture. In *Database Systems for Advanced Applications*, pages 63–78. Springer.

Ngu, A., Bowers, S., Haasch, N., McPhillips, T., and Critchlow, T. (2008). Flexible scientific workflow modeling using frames, templates, and dynamic embedding. In Ludäscher, B. and Mamoulis, N., editors, *Scientific and Sta-*

tistical Database Management, volume 5069 of *Lecture Notes in Computer Science*, pages 566–572. Springer Berlin / Heidelberg.

Nilsen, R. and Karplus, W. (1974). Continuous-system simulation languages: A state-of-the-art survey. *Mathematics and Computers in Simulation*, 16(1):17–25.

NS-3 Consortium (2013). Network simulator 3. https://www.nsnam.org/. Accessed: May 3, 2016.

Oberweis, A., Schätzle, R., Stucky, W., Weitz, W., and Zimmermann, G. (1997). *Income/Wf — A Petri Net Based Approach to Workflow Management*, pages 557–580. Physica-Verlag HD, Heidelberg.

Oinn, T., Greenwood, M., Addis, M., Alpdemir, N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M., Senger, M., Stevens, R., Wipat, A., and Wroe, C. (2006). Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100.

Orchard, R. (2001). Fuzzy reasoning in jess: the fuzzy j toolkit and fuzzy jess. http://nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=8913294. Accessed: May 3, 2016.

Oreizy, P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., and Wolf, A. L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, 14(3):54–62.

OSGi Alliance (2003). *Osgi service platform, release 3.* IOS Press, Inc.

Park, M.-J. and Kim, K.-H. (2010). A workflow event logging mechanism and its implications on quality of workflows. *Journal of Information Science and Engineering*, 26(5):1817–1830.

Pawlikowski, K., Jeong, H.-D., and Lee, J.-S. (2002). On credibility of simulation studies of telecommunication networks. *IEEE Communications Magazine*, 40(1):132–139.

peng XIU, J., ting XU, Y., DENG, F., and LIU, C. (2010). A petri net-based approach for data race detection in bpel. *The Journal of China Universities of Posts and Telecommunications*, 17:10 – 15.

Perrone, L. F., Kenna, C. J., and Ward, B. C. (2008). Enhancing the credibility of wireless network simulations with experiment automation. In *WIMOB '08: Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, pages 631–637, Washington, DC, USA. IEEE Computer Society.

Perrone, L. F., Main, C. S., and Ward, B. C. (2012). Safe: simulation automation framework for experiments. In *Proceedings of the Winter Simulation Conference*, page 249. Winter Simulation Conference, IEEE.

Pesic, M., Schonenberg, H., and van der Aalst, W. M. P. (2007). Declare: Full support for loosely-structured processes. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, pages 287–287. IEEE.

Petri, C. (1966). Kommunikation mit automaten. bonn: Institut für instrumentelle mathematik, schriften des iim, no. 3 (1962); also, english translation: Communication with automata, griffiss air force base. Technical report, New York. Tech. Rep. RADC-TR. 1 (suppl. 1).

Phillips, A. (2007). Chemical examples in spim. http://research.microsoft.com/en-us/um/people/aphillip/SPiM Accessed: May 3, 2016.

Proctor, M. (2012). Drools: a rule engine for complex event processing. In *Applications of Graph Transformations with Industrial Relevance*, pages 2–2. Springer.

Rabe, M., Spieckermann, S., and Wenzel, S. (2009). Verification and validation activities within a new procedure model for v&v in production and logistics simulation. In *Winter Simulation Conference*, pages 2509–2519. Winter Simulation Conference, IEEE.

Rademakers, T. (2012). *Activiti in Action: Executable business processes in BPMN 2.0*. Manning Publications Co.

Reichert, M. and Dadam, P. (1998). Adept flex—supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129.

Reijers, H. A., Rigter, J., and van der Aalst, W. M. P. (2003). The case handling case. *International Journal of Cooperative Information Systems*, 12(03):365–391.

Ribault, J. and Wainer, G. (2012). Simulation processes in the cloud for emergency planning. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 886–891. IEEE Computer Society.

Robinson, S. (2011). Choosing the right model: conceptual modeling for simulation. In Jain, S., Jr., R. R. C., Himmelspach, J., White, K. P., and Fu, M. C., editors, *Winter Simulation Conference*, pages 1428–1440. ACM.

Romano, P. (2008). Automation of in-silico data analysis processes through workflow management systems. *Briefings in Bioinformatics*, 9(1):57–68.

Roure, D. D., Goble, C., Bhagat, J., Cruickshank, D., Goderis, A., Michaelides, D., and Newman, D. (2008). myexperiment: Defining the social virtual research environment. In *4th IEEE International Conference on e-Science*, pages 182–189. IEEE Press.

Royce, W. W. (1970). Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26, pages 328–338. Los Angeles.

Russell, N. and ter Hofstede, A. H. M. (2009). Surmounting bpm challenges: the yawl story. *Computer Science - Research and Development*, 23(2):67–79–79.

Russell, N., van der Aalst, W. M. P., and ter Hofstede, A. H. M. (2009). Designing a workflow system using coloured petri nets. In Jensen, K., Billington, J., and Koutny, M., editors, *Transactions on Petri Nets and Other Models of*

Concurrency III, volume 5800 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rybacki, S., Haack, F., Wolf, K., and Uhrmacher, A. M. (2014). Developing simulation models - from conceptual to executable model and back - an artifact-based workflow approach. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '14, pages 21–30, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Rybacki, S., Himmelspach, J., Haack, F., and Uhrmacher, A. (2011). Worms-a framework to support workflows in m&s. In *Simulation Conference (WSC), Proceedings of the 2011 Winter*, pages 716–727. IEEE.

Rybacki, S., Himmelspach, J., Seib, E., and Uhrmacher, A. (2010). Using workflows in m&s software. In *Winter Simulation Conference (WSC), Proceedings of the 2010*, pages 535–545. IEEE.

Rybacki, S., Himmelspach, J., and Uhrmacher, A. M. (2012a). Using workflows to control the experiment execution in modeling and simulation software. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 93–102, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Rybacki, S., Leye, S., Himmelspach, J., and Uhrmacher, A. (2012b). Template and frame based experiment workflows in modeling and simulation software with worms. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*, pages 25–32. IEEE.

Sargent, R. G. (2008). Verification and validation of simulation models. In *Proceedings of the 40th Conference on Winter Simulation*, WSC '08, pages 157–169. Winter Simulation Conference.

Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of Simulation*, 7(1):12–24.

Sawyer, J. T. and Brann, D. M. (2008). How to build better models: applying agile techniques to simulation. In *WSC '08 Proceedings of the 40th Conference on Winter Simulation*, pages 655–662. IEEE.

Schick, S., Meyer, H., and Heuer, A. (2011). Flexible publication workflows using dynamic dispatch. In *Digital Libraries: For Cultural Heritage, Knowledge Dissemination, and Future Creation*, pages 257–266. Springer.

Schmidt, K. (2000). Lola a low level analyser. In Nielsen, M. and Simpson, D., editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474–474. Springer Berlin Heidelberg, Berlin, Heidelberg.

Schruben, L. W. (1982). Detecting initialization bias in simulation output. *Operations Research*, 30(3):151–153.

Schützel, J., Meyer, H., and Uhrmacher, A. M. (2014). A stream-based architecture for the management and on-line analysis of unbounded amounts of simulation data. In *Proceedings of the 2nd ACM SIGSIM/PADS conference on Principles of advanced discrete simulation*, pages 83–94. ACM.

Shannon, R. (1975). *Systems Simulation the Art and Science.* Prentice-Hall, Englewood Cliffs, N.J.

Shannon, R. E. (1998). Introduction to the art and science of simulation. In *Proceedings of the 30th Conference on Winter Simulation*, WSC '98, pages 7–14, Los Alamitos, CA, USA. IEEE Computer Society Press.

Shapiro, R. and Marin, M. (2008). Workflow management coalition workflow standardprocess definition interface-xml process definition language. *The Workflow Management Coalition*, 99.

Sims, C. and Johnson, H. L. (2011). *The elements of Scrum.* Dymax.

Sommerville, I. (2007). *Software Engineering.* Addison-Wesley, 8 edition.

Sonntag, M., Görlach, K., Karastoyanova, D., Currle-Linde, N., et al. (2010a). Towards simulation workflows with bpel: deriving missing features from gricol. In *Proceedings of the 21st IASTED International Conference*, volume 15, page 17. ACTA Press.

Sonntag, M., Hotta, S., Karastoyanova, D., Molnar, D., and Schmauder, S. (2011). Using services and service compositions to enable the distributed execution of legacy simulation applications. In Abramowicz, W., Llorente, I., Surridge, M., Zisman, A., and Vayssière, J., editors, *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, pages 242–253. Springer Berlin / Heidelberg.

Sonntag, M., Karastoyanova, D., and Deelman, E. (2010b). Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows. In *Proceedings of the 6th IEEE International Conference on e-Science*, pages 206–213. IEEE.

Sonntag, M., Karastoyanova, D., and Leymann, F. (2010c). The missing features of workflow systems for scientific computations. In *Software Engineering (Workshops)*, pages 209–216. Citeseer.

Talia, D. (2013). Workflow systems for science: Concepts and tools. *ISRN Software Engineering*, 2013.

Taylor, B. and Kuyatt, C. (1994). Nist technical note 1297 1994 edition guidelines for evaluating the uncertainty of nist measurement results. *National Institute of Standards and Technology, Gaithersburg, MD.*

Taylor, I., Shields, M., Wang, I., and Harrison, A. (2007). The triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer.

Toussaint, A. (2003). Java rule engine api: Jsr-94. *Java Community Process*, pages 8–33.

Van Der Aalst, W. and Van Hee, K. M. (2004). *Workflow management: models, methods, and systems.* MIT press.

van der Aalst, W. M. P. (1996a). Structural characterizations of sound workflow nets. *Computing Science Reports*, 96(23):18–22.

van der Aalst, W. M. P. (1996b). Three good reasons for using a petri-net-based workflow management system. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201. Citeseer, Springer.

van der Aalst, W. M. P. (1998). The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66.

van der Aalst, W. M. P., van Hee, K. M., and Houben (1994). Modelling workflow management systems with high-level petri nets. In *In Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms (1994)*, pages 31–50.

van der Aalst, W. M. P., Weske, M., and Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162.

Van Gorp, P. and Dijkman, R. (2011). Bpmn 2.0 execution semantics formalized as graph rewrite rules: extended version. Technical report, Beta Working Paper, series 353, Eindhoven University of Technology.

Verbeek, E. and van der Aalst, W. M. P. (2000). *Woflan 2.0 A Petri-Net-Based Workflow Diagnosis Tool*, pages 475–484. Springer Berlin Heidelberg, Berlin, Heidelberg.

Vicino, D., Dalle, O., and Wainer, G. (2014). A data type for discretized time representation in devs. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 11–20. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Walker, D. W. (1994). The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20(4):657–673.

Wang, S. and Wainer, G. (2015). A simulation as a service methodology with application for crowd modeling, simulation and visualization. *Simulation*, 91(1):71–95.

Wassermann, B., Emmerich, W., Butchart, B., Cameron, N., Chen, L., and Patel, J. (2007). Sedna: A bpel-based environment for visual scientific workflow modeling. In *Workflows for e-Science*, pages 428–449. Springer.

Wassink, I., Rauwerda, H., Van Der Vet, P., Breit, T., and Nijholt, A. (2008). *e-BioFlow: Different perspectives on scientific workflows.* Springer.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, University of Cambridge England.

Weißbach, M. and Zimmermann, W. (2010). Termination analysis of business process workflows. In *Proceedings of the 5th International Workshop on Enhanced Web Service Technologies*, WEWST '10, pages 18–25, New York, NY, USA. ACM.

Weidemann, A., Richter, S., Stein, M., Sahle, S., Gauges, R., Gabdoulline, R., Surovtsova, I., Semmelrock, N., Besson, B., Rojas, I., Wade, R., and Kummer, U. (2008). Sycamore–a systems biology computational analysis and modeling research environment. *Bioinformatics (Oxford, England)*, 24(12):1463–4.

Weld, D. S. (1994). An introduction to least commitment planning. *AI magazine*, 15(4):27.

Weske, M. (2007). *Business Process Management – Concepts, Languages, Architectures.* Springer Verlag Berlin Heidelberg.

WfMC (1999). Workflow management coalition terminology & glossary (document no. wfmc-tc-1011). *Workflow Management Coalition Specification.*

White, S. and Miers, D. (2008). *BPMN modeling and reference guide.* Future Strategies Inc.

White Jr., K. P. (1997). An effective truncation heuristic for bias reduction in simulation output. *Simulation*, 69(6):323–334.

Wong, S. C., Miles, S., Fang, W., Groth, P., and Moreau, L. (2005). Validation of e-science experiments using a provenance-based approach. In *4th UK e-Science All Hands Meeting (AHM).* University of Southampton.

Woodman, S., Hiden, H., Watson, P., and Missier, P. (2011). Achieving reproducibility by combining provenance with service and workflow versioning. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, pages 127–136. ACM.

Wynn, M., Verbeek, H., van der Aalst, W., ter Hofstede, A., and Edmond, D. (2009). Business process verification – finally a reality! *Business Process Management Journal*, 15(1):74–92.

Yildiz, U., Guabtni, A., and Ngu, A. H. (2009). *Business versus Scientific Workflows: A Comparative Study.* IEEE.

Zha, H., van der Aalst, W., Wang, J., Wen, L., and Sun, J. (2011). Verifying workflow processes: a transformation-based approach. *Software and Systems Modeling*, 10(2):253–264.

Zha, H., Yang, Y., Wang, J., and Wen, L. (2008). *Transforming XPDL to Petri Nets*, pages 197–207. Springer Berlin Heidelberg, Berlin, Heidelberg.

Zhao, J., Gomez-Perez, J. M., Belhajjame, K., Klyne, G., Garcia-Cuesta, E., Garrido, A., Hettne, K., Roos, M., De Roure, D., and Goble, C. (2012). Why workflows break—understanding and combating decay in taverna workflows. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–9. IEEE.

Zinn, D. (2010). *Design and optimization of scientific workflows.* PhD thesis, University of California Davis.

Zinn, D., Bowers, S., McPhillips, T., and Ludäscher, B. (2009). Scientific workflow design with data assembly lines. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, page 14. ACM.

zur Muehlen, M. and Indulska, M. (2010). Modeling languages for business processes and business rules: A representational analysis. *Information Systems*, 35(4):379 – 390.

Theses

1. A lot of published results in the domain of Modeling & Simulation are not reproducible, leading to a *Crisis of Credibility.*

2. Credible scientific results are obtained using a well-defined and documented process and are reproducible.

3. Providing documentation and provenance data for a simulation study is a complex endeavor.

4. Workflows pose an instrument for providing and collecting documentation and provenance information automatically.

5. The process of performing a simulation study consists of two layers that make different demands on the workflow description. The first layer, the simulation model creation, is a highly interactive and human driven process. The second layer, the execution of a simulation experiment, is rigid and typically automatable.

6. A highly interactive process is easier described using a declarative workflow description, while a mainly automatable rigid process is better described using imperative workflow descriptions.

7. Using an artifact-based workflow description, in particular the Guard, Stages and Milestones approach, allows describing the model creation process using one generic workflow.

8. Using workflow nets with the template and frames extension, allows describing the process of executing a simulation experiment using one generic workflow. This workflow is integrable through and executable with WorMS.

9. Reinforcement learning, Q-Learning in particular, can be used for automatically adapting the scheduling and distribution of workflow work items in a parallel or distributed environment, optimizing the overall performance of the workflow execution.