Universität Rostock Traditio et Innovatio

Fakultät für Maschinenbau und Schiffstechnik
Lehrstuhl Schiffbau

Knowledge-Based Design Patterns for Detailed Ship Structural Design

Doctoral Thesis

for the academic degree of

Doktor-Ingenieur

submitted to the Board of the Faculty of
Mechanical Engineering and Marine Technology
of the University of Rostock

by

Dipl.-Ing. Michael Zimmermann
born on 30.04.1976 in Freiburg (Breisgau)

Rostock, Mai 2010

urn:nbn:de:gbv:28-diss2011-0091-0

Reviewer

Prof. Dr.-Ing. Robert Bronsart

Universität Rostock
Fakultät für Maschinenbau und Schiffstechnik
Lehrstuhl Schiffbau

Prof. Dr.-Ing. Wolfgang Fricke

Technische Universität Hamburg-Harburg
Institut für Festigkeit und Konstruktion von Schiffen

Prof. Dr.-Ing. Dr. h.c. Pentscho Pentschew

Universität Rostock
Fakultät für Maschinenbau und Schiffstechnik
Lehrstuhl für Schiffstechnische Konstruktionen

The thesis defence took place on April 6^{th} 2011 at the University of Rostock.

Contents

List of Figures

List of Tables

1. Introduction

The development process in the maritime industries is characterized by a complex interaction of many partners working in parallel. From the initial concept to the final design, diverse and often conflicting requirements need to be taken into account. For this purpose, the constant exchange and consideration of a significant amount of information is needed. Also, shipyards are faced with high requirements regarding design and build quality, so that approaches towards quality assurance in design and production need to be taken. In addition, as stringent cost considerations are given, design solutions need to be highly optimized towards the manufacturing capabilities available. For this purpose and to facilitate the exploration of series effects, so-called best-practice standards can be applied and are in use.

In the domain of detailed ship structural design, *Design Standards* are hence used or worked out for almost every design project, whereby a set of optimal solutions for a selection of precisely defined design problems is determined. Ranging from simple agreements like common naming conventions to elaborate definitions, e.g., of design methodologies and calculation procedures for the design of engine foundations, with such standards compulsory regulations for the design or for certain design aspects can be defined.

With standardization, multiple effects within the design process can be achieved, as explained by means of the following design standard:

> Within the ice belt of NB 123, brackets are required on one end of every longitudinal frame. In case of an identical grade of material (A36) and an identical plate thickness of frame, bracket and shell profile, it is as follows:
> – Use cutouts of type C10 with a notch radius R20 for all profiles
> – For HP 100x8 profiles use a bracket BC5 130x8 A36 with Offset 0, Notch VU 50x30 and Bevel Code 200 on all sides.
> – For HP 140x9 profiles ...
> – ...

Taken from the internal communication within the design department of a major German shipyard, this short text defines a specific design procedure for all engineers working on a specific design task (stiffening system) in a specific location (ice belt) of a specific vessel (NB 123) for a specific stiffener configuration (HP 100x8) and under additional restrictions (material grade etc.). Hereby, for this design problem and within the context defined, a common understanding among the engineering personnel involved is achieved resulting in common design solutions. The standard hence serves as a medium of communication. Furthermore, the standard represents an approved best-practice solution, i.e. the application of this standard ensures that relevant class rules are fulfilled intrinsically. Therefore, standards are a means to guarantee that expert knowledge is considered, which is typically not known to the engineer working in the field of detailed steel design.

During design, therefore, design knowledge is an important asset. Optimal design solutions with respect to function, quality and cost can only be reached if for each decision all required

information is available and taken into account. In case of incomplete knowledge or if the knowledge available is not fully regarded, no optimal solution can be reached. As a result, suboptimal solutions or even design errors may occur. Additional costs for corrections might be required.

While standards are a means to apply knowledge within the design process, the expressibility of standards is often limited with respect to allowed complexity or general validity. For example, the design of a simple deckhouse superstructure as shown in figure 1.1 is primarily based on standards; yet the overall layout of the stiffener system used for the top-plate shown upside-down on the right is primarily governed by aspects regarding vibration that are often not directly considered in standards formulations. Therefore, more advanced design methods, the so-called *Knowledge-based Engineering* (KBE), can provide case-specific means to test existing design solutions with respect to certain criteria or to guide the engineer during the actual design process.

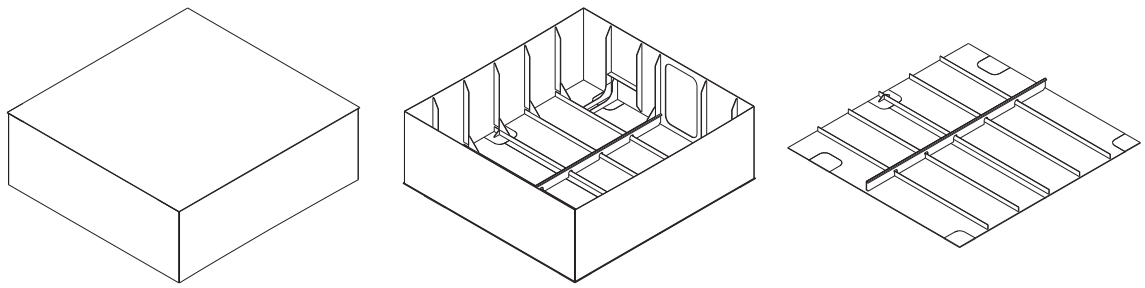


Figure 1.1.: Steel Design of a Simple Superstructure

For major CAD/CAM software systems used in naval architecture today, support for such a standards-compliant, knowledge-based, optimal design is limited. Primarily, the engineering personnel is responsible to guarantee standards compliance with respect to shipyard standards and design context. Information from external data sources like compartmentation plans or FEA analyses needs to be interpreted and integrated manually. For example, as part of the superstructure design, see figure 1.1, the overall layout and spacing of the stiffener system may be determined by an expert within the shipyard or by an external vibration consultant. Hence for each design action, manual interaction is required to select feasible solutions and to determine the location of installation. The actual design activity, i. e. the codified series of design steps and interactions with the CAD system required for the actual design to be defined, is fully performed by the user.

Therefore, in this thesis, approaches towards the knowledge-based application of standards for detailed ship structural design are presented. The development of an autonomous or semi-autonomous design system for the standards-compliant knowledge-based design is sought. With a focus on the domain of naval architecture, the key research objectives can be formulated as follows:

- What types of knowledge are required for the definition of a design procedure? How much information is needed for an unambiguous automatic design? What is the situation with respect to the availability of this information?
- What are the capabilities that are required for an electronic representation of this knowledge? Which are the technical approaches available and their respective capabil-

ities?

- What kind of approaches can be applied for the definition of formalized, reusable design procedures for a standards-compliant design?
- Is it possible that such design procedures can be applied automatically and can hence the design process be automated? To what extent is this feasible? Where are the limitations?
- What are the effects of such methods on build quality and design time?

Viewing design from a perspective of so-called applied problem solving, the concept of knowledge as relevant for this thesis is presented in chapter 2. The role of knowledge within detailed design is explored and means for the electronic representation of all types of knowledge relevant in this design stage are developed. Methods for an electronic representation of design knowledge are introduced. Also, methods for problem solving are explained, i.e. systematic approaches for the application of said knowledge towards automatic design are explored. In chapter 3, these fundamental concepts are extended and detailed for the domain of engineering. Key activities performed as part of engineering design, the so-called design tasks, are presented and the concept of knowledge-based engineering is introduced.

In chapter 4, the design process as given in naval architecture is summarized and examples of knowledge-based engineering methods as applied in this domain are presented. In addition, a more elaborate analysis of the role of standardization is performed. Design elements as used for detailed ship structural design are analyzed and classified.

For the evaluation of knowledge-based approaches towards detailed design, a set of test cases is introduced in chapter 5. With the help of these test cases, the requirements placed on software solutions for knowledge-based engineering are presented in chapter 6. The system architecture developed as part of this thesis is presented. For simple design problems like the example of a standards formulation given above, the adequate electronic formulations are presented in chapter 7. A detailed explanation of the KBE algorithms developed is given. Using the test cases as reference, hence, methods for context-sensitive design are developed. With the help of so-called *Design Patterns*, solutions for the automatic design of ship structural elements are finally presented in chapter 8. Approaches towards a fully automatic definition of more elaborate, optimal and standards-compliant solutions are presented.

2. Knowledge and Problem Solving

Knowledge is a key foundation for any activity or understanding in life. While an exact and unambiguous definition of knowledge is a matter of on-going debate among philosophers, in this chapter the term knowledge as relevant in the domain of engineering and naval architecture is defined. The processes of knowledge creation and knowledge transfer are discussed. The consequences for IT-based systems are considered. Concepts for knowledge management and for the electronic representation of engineering knowledge are introduced.

2.1. Knowledge and Knowledge Management

According to Plato's definition of knowledge as *Justified True Belief* knowledge can be defined as the thorough understanding of a subject or a problem domain based on a person's skills, expertise or values. It encompasses the set of all known facts and pieces of information available in a particular field today [1] and depends on the understanding of a problem or situation where the understanding is influenced by the individual's insight into the context.

Constructivism extends this approach and therefore assumes that knowledge is construed by individuals based on their perception. Therefore, it is directly bound to an individual. Perceived facts or experiences are used to construct mental models of reality. Existing mental models can be assimilated or accommodated by new perceptions.

In contrast, the theory of *positivism* is based on the assumption of an objective, universally true reality that is perceived and understood to a varying degree by an individual. The process of perception does not change the reality. The competence of an individual yet influences the efficiency and results of a reasoning process, i.e. different conclusions may be reached depending on the preconditions. Positivism can therefore be seen as one of the foundations of modern natural sciences [2]. In engineering with its strong ties into natural sciences, hence, a positivistic approach is applied.

For the work presented in the following, a positivistic understanding of knowledge is used. As such an understanding is geared towards the representation of facts or other measurable and comparable entities in the domain of natural sciences and engineering, the components of knowledge can be defined as shown by [3]:

Definition 1. *As part of knowledge, a set of informatorial elements, the so-called* ***Facts*** *or* ***Data*** *are given where each fact represents an atomic entity of the domain of intercourse. A fact therefore represents some piece of information that is self-supportive and not related to other entities. Based on facts,* ***Information*** *is defined as a collection of interrelated facts where the relation between these facts is known and given explicitly. This network of facts defines the so-called* ***Knowledge Base*** *or* ***Information Base****.*

Based on existing knowledge or experience, *Reasoning* is used to assert and evaluate new

facts. New knowledge is created where the context in which reasoning is performed influences the outcome of the reasoning process. With this approach, the essence of information can be extracted from the information base. Here, depending on the personal view taken, differing conclusions may be drawn from an identical information base. Therefore, albeit being assumed to be universally true, knowledge is bound to individuals.

2.1.1. Types of Knowledge

From a theoretical point of view, different types of knowledge can be identified, see [4]. Knowledge can be classified by the field of applicability, explicitness and structure.

Knowledge is either *globally valid* or *domain-specific*. The first type denotes knowledge or pieces of information that are valid independent of the context they are used in. As an example, mathematic axioms can be named. Domain-specific knowledge is valid only for a certain problem domain or area of discourse. For example the class rules from classification societies are valid in the domain of naval architecture only. Often, global knowledge and general common agreements serve as foundation for different domain-specific sets of knowledge. For a concise definition of domain-specific knowledge, the problem domain has to be identified. Limitations have to be defined.

Explicit Knowledge is knowledge that can be described using facts and relations. If knowledge is based on natural sciences or can be expressed unambiguously and independent of the qualification of any communication partners, the transfer of such knowledge is possible with only little danger of misinterpretation. In contrast, *Implicit Knowledge* is knowledge that cannot be stated explicitly. Often, implicit knowledge forms one important foundation a team or society is based on. This type of knowledge encompasses pre-existing social norms, shared agreements or experience and is based on a positivistic view.

Structured Knowledge is given in a predefined format, i.e. the semantics of the knowledge is clearly defined. Examples in ship design are e.g. parts libraries or certain parts of class rules. In general, this type of knowledge is expressed as a collection of facts with only a limited number of relations between them where the layout of these relations is known. For *Unstructured Knowledge* no *Meta-Information* about the semantics used is given, i.e. the relationship between the facts that make up a certain part of knowledge is not known. No information about how the individual facts should be assembled into a coherent network of information is available. The evaluation of this type of knowledge is therefore highly dependent on information about the context as well as on experience. In engineering, an informal, generally accepted notation, comments or experience enable the engineer to abstract the required knowledge. For any automatic evaluation, the determination of the context and constraints is difficult.

2.1.2. Application of Knowledge

For any activity performed knowledge serves as a key foundation. Often, the application of knowledge is therefore action-driven where four distinct states can be identified as shown in figure 2.1. Using reasoning processes, transitions between these states are feasible.

Based on an observation of the domain of intercourse or the problem at hand, the *world*, new informational artifacts can be subsumed by *induction*. According to Sowa [5], these

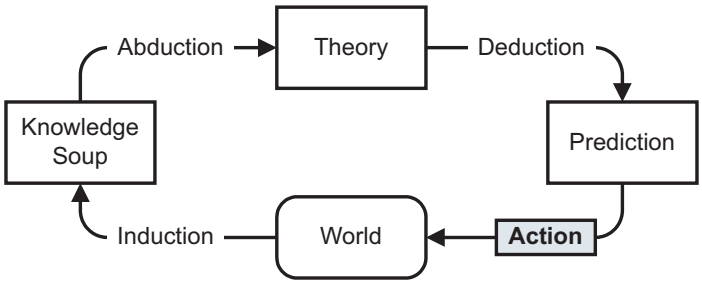


Figure 2.1.: The Cycle of Pragmatism [5]

artifacts are created as unordered and unconnected facts resulting in a collection of pieces of information, the so-called *Knowledge Soup*. By distilling new facts from the domain of intercourse and adding them to the knowledge soup, the information base to operate on is enriched. At this stage, no deeper understanding of the concepts represented by these facts is given, though. In the next step, knowledge *theories* are derived from the collection of facts by *abduction*. Here, existing knowledge is adapted to accommodate the gathered facts. For this purpose, previously observed knowledge elements are reassembled and, hence, new ideas or concepts can be introduced. Hypotheses are created [6] and existing knowledge is reused, revised or combined. Depending on the semantic distance of the elements involved, either modified or novel theories are reached. If an existing or newly created theory is applied to the problem at hand, a so-called *prediction* is reached. New knowledge or advisories for specific actions to take are reached using a process called *deduction*. Finally, the execution of such an advisory leads to an *action* that influences the current situation. The world or domain of intercourse is changed whereby new input might be generated to process by the following *Cycle of Pragmatism*.

In engineering as well as in many other operative areas, the methods of induction, abduction and deduction are considered as means to initiate an action [4]. For this purpose, the whole context of the problem is taken into account. A prediction therefore does not only include the information what to do (know-what), but also implies knowledge about the processes required to actually perform the action (know-how). A motivation for the action needs to be given (know-why).

Knowledge with its strong ties to an individual's general understanding and awareness of the context is difficult to transfer. According to Riempp [4], the transfer of knowledge includes multiple steps as shown in figure 2.2. For the transfer of knowledge between two acting entities, the mental model of the sender is explicated and reflected. Existing knowledge is broken down into the underlying facts and their relationships, i.e. the information base. During reflection, the sender takes the general situation, the problem at hand as well as the knowledge about the competence of the receiver into account. Only those pieces of information are expressed that can be understood by the receiver and that are considered as relevant, i.e. a filtering process takes place and the problem at hand is disconnected from the complete, yet complex mental model of the sender. Hereby, the number of associations is reduced. By explicitation, the reduced and tailored mental model is expressed explicitly in such a way that it can be communicated verbally.

Not taking possible incorrect transmission from sender to receiver into account, the receiver uses the information base provided as input. Facts and associations are decoded, i.e trans-

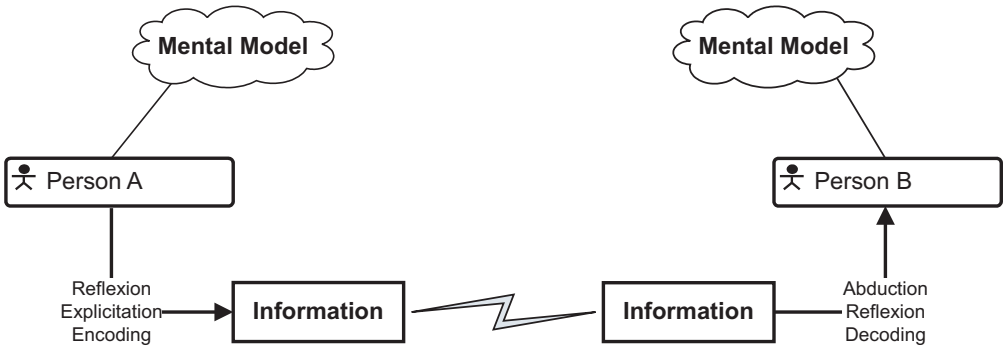


Figure 2.2.: The Process of Knowledge Transfer [4]

ferred into the language of the receiver, and reflected upon. From the collection of facts, the knowledge soup, theories are derived by abduction. New insights are gained and the existing mental model can be adapted.

The process of abduction is possible if the receiver is capable to understand and accept the information provided. An ability to communicate must be given between sender and receiver. Social factors like the willingness to communicate and to transfer knowledge as well as cultural or collaborative aspects are important factors that determine the outcome of a transfer. Incomplete decoding of the information base can lead to an incomplete transfer of knowledge. Based on existing knowledge on the receiver's side, the information base can be interpreted differently resulting in an incorrect knowledge transfer.

For any machine-machine or man-machine interaction the same general principles hold true [7] as computer-based algorithms for, e.g., encoding and decoding, reflecting and abducting information objects are designed by human beings. Hence, general understandings of the developers are key factors for the successful realization of these components or interfaces. These are issues that need to be addressed during design and development.

In contrast to interactive communication between humans, any interaction involving machines is based on a time-invariant definition of the relevant context, i.e. it includes static assumptions about the operational culture as well as the capabilities of understanding. Man-machine or machine-machine interaction therefore can be considered to be more inflexible. In case of changes in the context or the underlying assumptions often human interaction is required [8]. As a result, a constant monitoring of the evolving situation and, if necessary, maintenance operations are required.

2.1.3. Knowledge Management

Knowledge is an important asset in a corporate environment. Methods for the management of knowledge have become important means for documentation and archiving. Approaches for the transfer and use of knowledge within a corporation or society have been derived. Following Grant [9] the notation of knowledge management can be defined as follows:

Definition 2. *Knowledge management is a systematic approach towards the definition of activities for the creation, aggregation, transfer and use of knowledge. It is focused on the introduction of processes for the application of knowledge throughout a company.*

Therefore, processes for knowledge management often play an integral part within corporations where these processes are developed in order to support business objectives and to improve the optimal use of the available human resources. In the domain of engineering and naval architecture, IT systems play an ever growing role for the efficient and context sensitive application of knowledge.

Depending on the perspective, three different approaches to knowledge management can be identified. *Strategy-oriented* knowledge management is a goal-driven approach with a strong focus on cost vs. effect aspects. It is mainly used to support decisions with long-term effects and focuses on the supportive role of knowledge for decision making. *Process-oriented* knowledge management can be used to identify or define knowledge-related business processes. These processes can be seen as instructions for the creation, documentation and distribution of knowledge. Processes for an efficient application of knowledge are installed. Finally, a *system-oriented* approach focuses on a technical perspective and concentrates on the role of information systems. Solutions and corresponding usage scenarios are developed to support or augment knowledge management processes or strategies.

In the field of engineering, most activities are product-driven, i.e. the development and manufacturing of or life-cycle support for a product are the main business objectives. Here, two different areas for the application of knowledge management strategies can be identified.

On the one hand, novel ideas or concepts are generated by the process of abduction. In ship design, the development of a completely new vessel according to some rough ideas of a prospective owner can be named as example. For this purpose and based on experience, existing knowledge is applied to determine a reasonable design proposal. Here, an adequate infrastructure for the efficient retrieval of existing solutions is supportive. A process-oriented view on knowledge management can be taken. Yet, as innovation is primarily based on the competence of the people involved, the role of IT is of minor importance for these processes. Implicit knowledge plays an important role.

On the other hand, the detailed definition and further planning of such concepts, i.e. the detailed design, is concerned with the further development of the product idea in accordance with established standards. For the mentioned example from the domain of ship design, the detailed definition of the steel structure, i.e. the determination of plate thicknesses, profile arrangements or bracket connections, are such design activities. Here, a significantly higher amount of structured knowledge is present. Information systems are not only used as knowledge providers but take an active part in the development process. In engineering, task-specific IT-systems are applied to support or automate well-known activities within the design process. In the following, approaches for the improved usage of this system-oriented view on knowledge and knowledge management are developed.

2.2. Electronic Knowledge Representation

In engineering, in the field of detailed design, a focus is placed on explicit, structured information in a known domain of intercourse. Hence, a positivistic understanding of knowledge can be assumed where actual data and not meta-data is of primary importance. For this information, a leveraging strategy as proposed by Van Krogh [10] is applied, i.e. the reuse of existing knowledge is sought where knowledge-based information systems are applied as

supportive tools. For this purpose, an electronic representation of the design knowledge is required.

2.2.1. What is a Knowledge Representation?

A knowledge representation or knowledge base can be defined as follows [11]:

Definition 3. *An electronic knowledge representation is a dynamic, IT-based and machine-interpretable representation of an information base for a known and agreed-upon domain of intercourse that reflects relevant knowledge about the problem at hand. The information base consists of a collection of symbolic structures representing the system's beliefs that are used for reasoning during operation.*

Therefore, three main characteristics can be identified:

- Symbolic or formal structures are used to represent information.
- The representation is grounded on unstated beliefs.
- Reasoning is performed.

The application of knowledge in an IT system requires explicit knowledge. During development and use, information therefore needs to be explicitated. Here, the knowledge representation acts as surrogate, i.e. as substitute for the real world or problem at hand. Formal symbolic structures are used to define the surrogate. For the definition of these structures, decisions regarding identity and fidelity are required. That means it needs to be determined what the knowledge representation should express; the accuracy of the representation needs to be set. As reality cannot be expressed completely, the knowledge representation is always incomplete. Any reasoning process performed may hence lead to results that differ from reality [12].

To define the fidelity of a surrogate representation requires decisions about the perspective and scope of the world or problem at hand, i.e. a set of so-called ontological commitments is made. Depending on the commitments made, different views or perspectives can be taken. Here, often unstated beliefs can have an influence on the perspective taken, see the discussion about explicitation and knowledge transfer in section 2.1.2. Also, as the selection of ontological commitments is driven by the capabilities of the technology used, often the technical perspective restricts the perspective taken or limits the modeling approaches available.

For any IT-based solution applied for knowledge representation and reasoning, the technology chosen is based on a theory of intelligent reasoning where a knowledge representation is founded on a certain theory of inference and knowledge representation. In the field of natural sciences and engineering, a logical view is dominant. As intelligent inference is assumed to follow the process of logical inference as introduced by Aristotle, permissible or sanctioned actions in a reasoning process are hence defined. In contrast, psychological models, as applied for stock markets, can be mentioned as examples for alternative inference theories.

Using computational algorithms, a knowledge representation is a means to perform an inference process, i.e. it is a medium of logical expression that is designed by humans. As more expressive and therefore more complex representations become computationally inefficient, a compromise has to be found where ease of use to express and communicate knowledge as well as expressibility has to be taken into account.

2.2.2. Foundations of Knowledge Representations

As a knowledge representation needs elements to operate on, these entities reasoned over need to be expressed. For this purpose, a set of propositions is used for the definition of a single entity where a single proposition consists of one or multiple statements. Each statement is self-contained. A knowledge representation therefore requires some sort of language that is used to formulate the knowledge. Using a logic-oriented perspective, a knowledge representation can be described using the language of first order logic or one of its descendants. Two different levels can be distinguished, namely:

- the knowledge level,
- the symbol level.

On the symbol level, computational aspects are considered. Procedures regarding the handling of symbols in the knowledge base are addressed. E.g., the parsing and execution of statements needs to be defined. Transformations of symbols are given.

On the knowledge level, definitions regarding *Syntax*, *Semantics* and *Pragmatics* are required. *Syntax* defines the feasible logical structure of statements and their valid combinations into propositions. Requirements for a well-formed knowledge base are defined. The core components on the syntactical level are variables that contain data about facts as well as functional symbols used to define relationships between variables. Predicate symbols like quantifiers and boolean operators are present [13].

A set of facts, relationships and terminology therefore can be used to define a complete knowledge representation. Following a frame-based or object-oriented approach, the terminology, i.e. a set of concepts, are used to state general characteristics of a certain entity. In naval architecture, e.g., the concept *Frame* may be defined to represent a physical object within the ship structure that has a certain contour, location in X direction etc. In case of a simplified frame representation, a frame may consist out of a single plate of a given thickness and material and may reach from the outer shell up to a maximum value in Z direction. A fact, i.e., an instance may then reference the concept *Frame* and supply actual data for the parameters of an representation of the *Frame* concept.

As an example, two different instances of the *Frame* concept are given below. On the left, a syntactically correct representation for an instance of Frame 34 at an x location of $x = 20.4m$ is shown. The example on the right is invalid in so far as a string of letters is assigned for the x location instead of the number expected.

```
1 Frame34 a Frame
2   id: FR34
3   xlocation: 20.4
4   zmax:8.5
5   material: A36
6   ...
```

```
1 Frame41 a Frame
2   id: FR41
3   xlocation: abcd
4   zmax: 6.8
5   material: A32
6   ...
```

Semantics are responsible to express the meaning of a syntactically correct proposition. This defines how a certain statement is tied into the world or problem at hand, i.e. the relationship of surrogate and reality is proposed and the meaning of propositions as a function of the interpretation of predicate and function symbols is defined. The limitations of a knowledge representation with respect to the domain of intercourse are hence staked out.

In the following two frame definitions, in both cases the semantics of the design problem are violated, i.e. the limitations imposed by general requirements proposed by the design project are not regarded. For the instance of *Frame188*, a material of grade D should be used according to the instance definitions. If it is not planned to use such a material for the ship structure, a design error with respect to the list of allowed materials is given. On the right, a frame is defined located at an x position $x = 4.05m$ before the aft perpendicular. With a requirement that all frames should be located on the frame grid, and a given fixed frame distance of $0.6m$, the frame can not be defined in such a way. In addition, the numbering schema applied for the naming of Frame5.5 is not compatible with the framing metrics applied. A frame Frame5.5 should be located at $x = 3.3m$.

```
1  Frame188 a Frame
2    id: FR188
3    xlocation: 112.8
4    zmax: 8.5
5    material: D
6    ...
```

```
1  Frame5.5 a Frame
2    id: FR5.5
3    xlocation: 4.05
4    zmax: 12.8
5    material: A
6    ...
```

A syntactically correct and semantically meaningful expression needs to be interpreted for a given context. *Pragmatics* provide this kind of information in so far as allowed logical consequences, the entailments, are stated. Here, for each knowledge base, a specific model of reasoning is applied. By deductive inference, i.e. by calculating the entailments of a given knowledge base, new propositions of the knowledge model can be explicated and added to the knowledge representation.

Again, for the frame definition used as example, two cases of incorrect definitions with respect to the design pragmatics are shown in the following. On the left a syntactically and semantically valid definition of a frame instance Frame296 is shown. Yet, if the vessel designed is a small container vessel designed for feeder services, the overall length of the vessel is significantly shorter than the x position of $x = 177.6m$ as given for Frame296. The frame definition therefore is invalid. On the right, a frame definition behind the aft perpendicular located at $x = -1.8m$ is given. If the vessel designed is classified according to a high ice class and if the location at FR -3 corresponds with the stern, material of grade A might be insufficient depending on the plate thickness and stiffening system used. Here, pragmatics, i.e. an understanding of the complete design problem would indicate the need for a design change or for further actions.

```
1  Frame296 a Frame
2    id: FR296
3    xlocation: 177.6
4    zmax: 4.5
5    material: A36
6    ...
```

```
1  Frame-3 a Frame
2    id: FR-3
3    xlocation: -1.8
4    zmax: 7.0
5    material: A
6    ...
```

In the last example, the action to take is not totally given; it depends on parameters unknown in this assumed scenario. Therefore, depending on the problem at hand, the reasoning process may have to take uncertainty into account. I.e. measures have to be taken to handle propositions that are either true, false or in-between. Adequate means to express the trust level for a given proposition are required. As reasoning or deductive inference does not guarantee logically sound and complete results, the inference process may return invalid solutions, be intractable, i.e. will never complete depending on the facts and relations of a knowledge base [14].

2.2.3. Classification and Examples of Knowledge Representations

Most knowledge representations in use today are based on a subset of first order logic, the so-called *Description Logics* [15]. This approach limits the expressibility of first order logic so that deductive reasoning always leads to a tractable and decidable solution. Such a knowledge representation consists of facts, relations and rules where rules are not supported by all knowledge representation systems.

Based on the field of applicability and modeling approaches taken, a classification of existing knowledge representation languages can be determined. In general, knowledge representations can be classified as:

- inference-oriented systems,
- rule-based systems.

The first group is focused on the usage of inference algorithms to determine and explicate knowledge already present in the model. The computation of entailments is performed, i.e. logical consequences are derived from existing data. For example, if a plate representation within a product model has two different thicknesses assigned, yet only a single thickness is allowed for such a definition, the identity of both thickness values can be calculated as entailment. Rules are of minor importance. In contrast, rule-based systems or so-called production-rule systems are systems that work on an existing fact base, the working memory. Here rules act as codified recipes for actions that operate on facts and associations. For example, in case of a known configuration of a pipe installation, insulation can be automatically added around this pipe by the execution of this rule. Initially, inference can be used while populating the working memory.

From a historic point of view, two different research fields can be named as key factors for the development of knowledge representations. These are:

- logic and artificial intelligence,
- semantic web technologies.

Logic Initial work on the electronic representation of knowledge was driven by researchers working in the fields of pure and applied logic. Systems like *KL-ONE* [16], *Loom* [17] or *Classic* [18] applied a so-called frame-based, object-oriented approach where a *Frame* is used to describe properties relevant for a certain concept. Terminological knowledge, i.e. concepts, are modeled as so-called T-Boxes that consist of a concept definition, a frame, and subsequent axioms. With the latter, restrictions on the model can be implemented. For assertional knowledge, so-called A-Boxes are used that link to the concepts fulfilled. This way, facts are defined and associations between different A-Box Entities are modeled. A network of data, i.e. a graph or network of items, is generated.

These modeling systems for knowledge representations emphasize a high level of expressibility. A sound logical basis is given while decidability is not always guaranteed. Often, inference processes are supported to validate a given knowledge base or to extend a given knowledge base with implicit knowledge. Based on newly asserted facts, the evaluation of the information base at run-time is not the main focus; interaction is of low importance.

Semantic Web With the rising importance of the internet, interoperability of distributed knowledge representations has become an important aspect. The Semantic Web envisions a set of independent knowledge representations, so-called *Ontologies* that are accessible over the internet where inference processes can freely combine and use any existing knowledge base. For this purpose and based on *XML* [19] and *XML Schema* [20], a series of standards and standard recommendations with increasing expressiveness has been defined by the W3C [21]. With these standards, subsets of description logic are supported [22].

The *Resource Description Framework* (RDF) [23] and *RDF Schema* (RDFS) [24] are focused on the representation of concepts, facts and associations [25]. No support for rules or complex expressions about cardinality or reference types is given. E.g. a plate in a marine structure can reference the material used for this plate. Yet, it is not possible to define the concept in such a way that the existence of a material definition for each instance is required. The *Web Ontology Language* (OWL) [26] extends RDF. Here, e.g., cardinality and type restrictions are introduced [27] [28].

Rules can be defined as conceptual definitions for actions performed on facts. Currently, there are several non-standardized languages, e.g. the *Rule Markup Language* (RuleML) [29] and the *Semantic Web Rule Language* (SWRL) [30], see [31] [32]. Software support and user adoption is limited, though [33].

The semantic web is build on the vision to connect distributed data. Unknown data sources play an important role. The existence of facts and concepts is not known. One of the fundamental principle of the semantic web technology therefore is the *Open World Assumption* (OWA) [34] [35]. Under an OWA, any proposition is assumed to be true as long as the opposite is not known. Further information about the practical effects of this assumption is given in section 7.3.4.2.

Engineering In engineering, there are currently no generalized extensive systems for knowledge representation available. Where knowledge representations are applied, often specialized representations using conventional databases or programming solutions are used. In most cases, such representations are used with the intent to represent and manage data; inference is rarely used. Semantics, if considered at all, are only represented weakly. In addition, computer applications are in use that often use special, closed data-formats for data storage of information represented within these systems. Again, a formalized approach to data representation and reuse is seldom given.

2.2.4. Knowledge Representations in Naval Architecture and Engineering

In naval architecture, only a limited number of open standards or formats for some kind of knowledge representation have been proposed. In most cases, these are focused on data representation, i.e. the standards are aimed towards facilitating the tasks of data modeling and data exchange. In addition, custom data formats are used for commercial applications that hence provide means to persist information like input data or results.

In contrast to real knowledge representations, all these representations can be classified as data representations where semantics and pragmatics are not key concerns and are often incomplete, inconsistent or missing. Commonly inference is not used and the design rationale or design intent is not expressed explicitly. In the following, a selection of some

representations is presented.

SFI Group System The *SFI Group System* is a classification system for the marine and offshore industry [36]. Based on a hierarchical structure, information about technical components or equipment can be classified according to functional aspects. In figure 2.3, the top-level classes of the system are shown. Using a document-centric view, existing documentation is assigned to categories within a fixed taxonomy. Depending on the categories determined as relevant for an individual project or task, views on the information base can be configured. The representation of concepts, facts and their properties is not supported. A document, drawing or other file can only be assigned to a single predefined category so that a precise classification is therefore not always feasible. In the industry the SFI Group system is used for classification and search of file-based information [37].

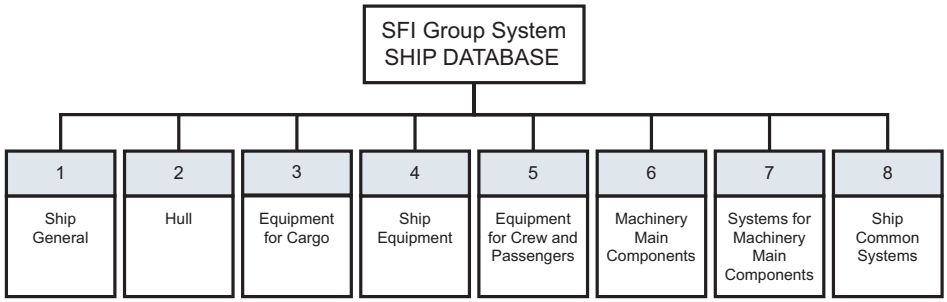


Figure 2.3.: Main Groups of the SFI Group System

STEP Initiated by ISO, the *Standard for the Exchange of Product Model Data* (STEP) is a general purpose standard for the transfer, storage, and transformation of all data for a product throughout the life-cycle of engineering products without regard to discipline or area of application [38] [39]. Using a layered, modular approach, a heterogeneous data model is defined. Due to the size of the problem domains approached, on the top level the STEP standard is divided into a set of standards. By an *Application Protocol* (AP), the relevant information model for a specific domain of application is defined [40]. For the field of naval architecture, the following application protocols are of importance:

- AP 215 - Ship Arrangement
- AP 216 - Ship Moulded Forms
- AP 218 - Ship Structures
- AP 226 - Ship Mechanical Systems

An *Application Activity Model* describes activities that are within the scope of the AP [41]. The vocabulary relevant for each application domain is defined in an *Application Resource Model* (ARM). APs and ARMs are build on top of a set of *Integrated Resources* that are reused within every AP. For an improved interoperability, so-called *Conformance Classes* are used to define different levels of implementation support for an AP [42].

During the development of STEP, four usage scenarios were considered. A passive file containing product data can be used to transfer or archive information. An active file can be

understood as database of STEP data. It is used interactively by the application. Using a database as data provider, additional capabilities like advanced search can be provided. Different views can be derived from the data. Relational, hierarchical or object-oriented perspectives can be derived. Finally, a knowledge-based implementation augments the information model with constraints and rules that are defined as part of the APs [43].

From a perspective of knowledge representation, ISO 10303 can be classified as data representation. Inference is not part of the standard; user-configurable or customized rules are not supported while the constraints or rules included in the standard are incomplete. While STEP offers a well-defined syntax, on the semantic and pragmatic level ambiguities are present, see [44]. Interoperability of different application protocols is limited, as terms assume different meanings depending on the APs [45] [46] [47] [48].

Ship Common Model and Integrated Shipbuilding Environment Based on STEP, an integrated approach was chosen for the development of the *Ship Common Model*. This model aggregates all application protocols relevant for the domain of naval architecture. Ambiguities are tried to resolve, yet the complexity of the data model is still given [49]. By contrast, the *Integrated Shipbuilding Environment* (ISE) that is used in the US Navy applies a lightweight approach [50] [51]. Where feasible, simple and clear solutions only for required data are chosen. Compared to STEP, a simplified data model concentrating on core concepts is derived. Using *XML Schema*, the required data structures are defined. Semantics and pragmatics are not considered [52]; support for user-configurable rules or inference is not supported. Both approaches therefore can be classified as data representations only.

PLIB The *Parts Library* Standard (PLIB), ISO 13584 [53], has been developed for the modeling and exchange of digital libraries of technical components, i.e. the exchange of technical data should be supported. Using this standard, knowledge and other criteria relevant for the selection of components can be represented. Where required, additional behavioral aspects of a part like, e.g., the life time of a needle bearing can be defined. Various component representations including fixed and parametric geometric representations can be defined. For modeling, i.e. for the syntax, the modeling language *EXPRESS* as defined in the STEP application protocol 20's series is used. Parts can be reused in multiple applications [54] so that hence modeling efficiency can be increased. Following a distributed approach, parts libraries are provided by the manufacturer of components [55].

An ontology model as introduced in section 2.2.2 is used to capture the meaning of structured data. The conceptualization applied is defined explicitly where a modular approach allows extensions to the model. Custom concepts and classification schemata can be introduced [56]. A PLIB data repository consists of a general ontology defining the syntax, semantics given by a schema and the actual component data. In contrast to the modeling languages used for the semantic web, see section 2.2.3, strong support for dimensional properties is built in [57] [58].

As the standard is based on precise syntactical and semantical definitions and is designed with search and reasoning processes as targets, the PLIB standard is an inference-oriented knowledge representation from a knowledge perspective. Implementations for inference support are not available, though, so that rules are not supported. As ISO 13584 uses EXPRESS [59] as modeling language, interoperability is mostly limited to STEP product data models.

Initial work towards the use of semantic web technologies has started, see [60]. Advanced context-based requirements modeling is not feasible as neither the required context models nor appropriate data exchange interfaces are available.

2.3. Methods for Problem Solving

A knowledge-based system is based on the objective to determine solutions for a given problem. The type of the problem, the data representation used and the solution strategy chosen are important aspects where different methods to solve the problem can be used depending on the solution strategy applied. In this section the notation of problem solving methods (PSMs) as relevant for engineering is introduced. A categorization of problem types is given and the role of knowledge representations is discussed.

2.3.1. Problem Solving - A Definition

As shown in section 2.2, electronic knowledge representations as used for automated problem solving are restricted to a limited number of entities in the world, the so-called universe of discourse. In this universe a *Problem* is characterized by at least two distinguishable states, the initial state and the final state, the goal. The initial state consists of a set of specifications, the input knowledge. The final state represents a solution for the problem where the specifications are satisfied [61].

Definition 4. *A Problem Solving Method (PSM) defines a generic, problem-independent description of a transformation algorithm between different states where the transformation is governed by a set of requirements that are formulated using domain specific knowledge. Being goal-driven, a sequence of actions towards the intermediate or final goal is defined by a PSM with the objective to satisfy the requirements stated. As part of a PSM, control structures are given to guide the solution process.*

Design in engineering, i.e. the execution of problem solving methods, can therefore be defined as a transformation of a set of input specifications that result in a feasible solution. According to Motta [62], as input a set of constraints C, requirements R and preferences P can be identified where

- constraints define conditions which must not be violated by the design,
- requirements define properties that have to be satisfied by the solution,
- preferences describe knowledge about the quality of design models.

In contrast to constraints that limit the space of admissible solutions, requirements have a positive connotation. Desired properties of the final design are described. Starting with an initial design context D_I, as a result of the transformation process, a design proposal model D_K is generated. This design model consists of a set of parameters p_i and their corresponding values v_i. Different states of the design model can be identified as shown in figure 2.4, namely:

- *Complete*: Each parameter p_i has a value in D_k.
- *Consistent*: No parameter violates any constraint in C.

- *Suitable*: All requirements in R are satisfied.
- *Valid*: The design model is suitable and consistent.

A design proposal model is a *solution* if it is complete and valid.

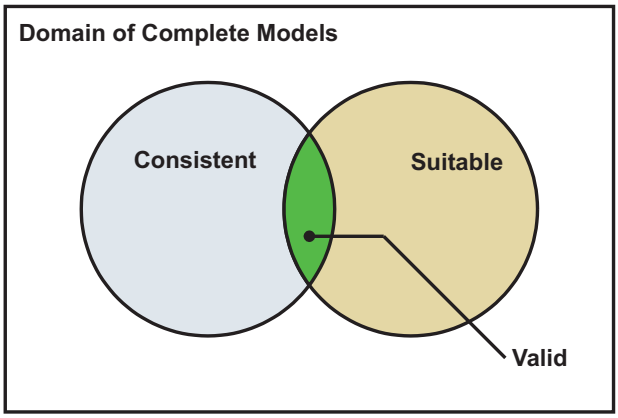


Figure 2.4.: States of the Design Model

Therefore, as objective of a problem solving method the selection of an optimal algorithm for the creation of a final valid design proposal of optimal quality and based on the initial design context can be named. Here, the quality of a design proposal can be assessed using a global *cost function* c_f and hence an *optimal solution* D_{K_i} can be determined:

$$c_f(D_{K_j}) < c_f(D_{K_i}) \quad \forall \quad j, j \neq i \tag{2.1}$$

2.3.2. Generic Approaches to Problem Solving

For the selection of adequate problem solving methods, the data representation given and modeling approach chosen for problem solving are of importance. According to e.g. Beierle [63] three distinct classes of modeling approaches can be distinguished. These are:

- Heuristic-based approaches,
- Case-based modeling,
- Constraint- or rule-based systems.

Heuristic-Based Modeling Neural networks as used in artificial intelligence (AI) apply a heuristic, connectionist approach to computation. A network of processing elements is construed. Here, each interconnecting element is responsible for a single transformation or processing task. With this approach, complex behaviors can be emulated by the network where the behavior of the network is influenced by the structure of the connections between elements. For each connection the strength, i.e. the contribution to a result, can be adjusted, see figure 2.5. Therefore, depending on the strength distribution, different solution paths can be emphasized.

A neural network is designed as an adaptive system. Starting with a set of user-defined processing elements, test cases with known solutions are fed into the system during the

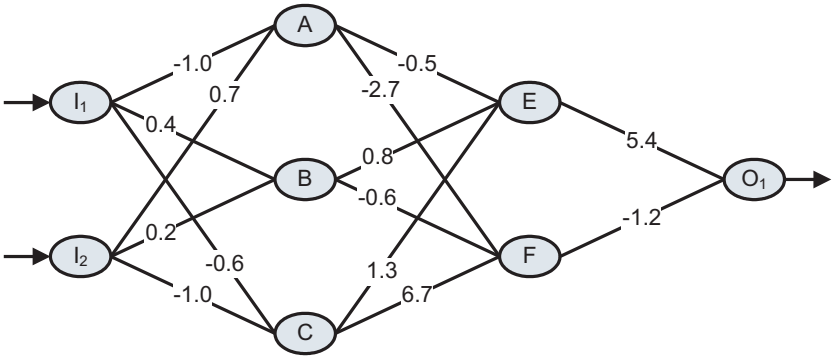


Figure 2.5.: Model of a Heuristic Neural Network

learning phase. The validity or applicability of the generated results are reported back to the system so that the structure of the connections as well as their strengths can be adjusted accordingly. Here, the paths used are assigned a higher weight for valid solutions and vice versa [64].

A neural network can be applied for problems where the processing elements are known or can be identified but where an explicit definition of all relevant relations and their respective strength are not known in detail. Once the main processing elements are identified, the solution process can be used as a black-box algorithm. Only input and output information are considered. Self-learning, i.e. the automatic adaption to changing requirements during the learning phase, is a key advantage of such an approach.

Case-Based Reasoning If the results of previous design processes can be made available as part of an IT-based design system, the experience captured in these existing results can be made available for subsequent designs. With case-based modeling and reasoning the transfer of the design knowledge to the current design is made possible. I.e. a knowledge base with validated, existing solutions can be used for a given problem domain. Therefore, for any problem fed into the system, the knowledge base is searched for suitable solutions. An objective function is used to determine the semantic distance, i.e. some measure for the applicability to the given problem. Candidate solutions are chosen and presented. Upon completion of a new design, this design solution is added to the knowledge base hence enriching the set of known solutions [65].

As a result, for identical problems a solution is derived only once by an engineer. Reuse reduces the design time needed and the process of knowledge transfer is augmented. For similar, yet not identical problems, candidate solutions can express the experience of previous design processes. The design of a final solution can hence be supported. Also, if more advanced approaches like abstraction, qualitative or analogical reasoning are used, reusability of parts of the knowledge model can be eased and an incomplete solution can be derived [66].

For case-base reasoning careful considerations regarding the definition of the knowledge model used are required, as this model is relevant for the variations that can be represented and evaluated by this approach. The knowledge model must therefore capture all relevant aspects of the design. Also, the domain of solutions is restricted by the expressibility of the knowledge representation. The same holds true for the definition of functions for the

calculation of the semantic distance [67].

In the domain of engineering, case-based reasoning is mostly used for the retrieval of knowledge about example solutions. The final adaption to the problem present is the responsibility of the engineer.

Constraint- and Rule-Based Systems In contrast to heuristic or case-based solutions, a declarative deterministic approach is chosen for constraint- and rule-based systems that allows for an explicit verification of the solution strategy adopted and the solution steps taken. Here, a distinction between *Constraint Systems* and *Rule-based Systems* can be made.

The former uses constraints, requirements and parameters to define relevant properties of the outcome, see the previous section. I.e., the key characteristics of the desired solution are described. The solution process is then governed by a *constraint solver*, a core component of a constraint-based system. This solver uses the input knowledge given, constructs design candidates by specific search algorithms and compares the result obtained with the given objective. Constraint programming can therefore be defined as search for a solution that fulfills all requirements. For the search, different solution strategies like e.g. combinatorial operations or decomposition strategies can be applied [68] [69].

For rule-based systems, an action-driven approach is taken. Rules are formulated that express requirements and consequent actions taken [70]. Depending on the state of the knowledge representation, rules are executed and the results generated are fed back into the representation. Hence, rules act on and modify the facts used. Two different system types can be identified, namely *goal-driven* and *input-driven* systems. In input-driven systems, rules are executed starting from the input data. A so-called forward-chaining approach is applied. The execution of rules propagates from the input data until either a result is obtained or all matching rules have been executed. Backward-chaining systems use a goal-driven approach. Based on a formalization of the desired result, the known facts are used to construe intermediate data that can lead to the result. The process of propagation stops once the input data required for the assumed result specification has been derived and this input data is compatible with the information base [71].

In engineering, often a definite reproducibility and traceability is required, i.e. *Confluence* and *Operational Equivalence* must be satisfied. Heuristic or case-based systems are not applicable as their underlying concepts of self-learning and heuristic approaches cannot guarantee such an absolute reproducibility and traceability of the reasoning performed. Therefore, only constraint- and rule-based systems are suitable for design problems in the domain of engineering as presented in this thesis [72].

2.3.3. Problem Solving by Configuration Design

Problem solving in the domain of engineering can be defined as goal-oriented search. Here, in most cases, starting with a preliminary design a process of successive design refinements is used where a top-down approach is applied. Working on a more and more detailed solution, in each iteration loop, constraints and requirements are identified and tested; design decisions are made. For this purpose, the shape of components and their respective fit is evaluated. Functional aspects are considered so that the working principle can be regarded [73]. If a primarily combinatorial design problem is given, *Configuration Design* methods can be

applied as search strategy. As solution strategies, heuristic and case-based PSMs as well as rule- or constraint-based algorithms can be used, see section 2.3.2 [74].

Definition 5. *Using a set of predefined components, Configuration Design as problem solving method can be applied to determine a feasible, complete and valid arrangement or assembly of distinct components into a coherent, specification-compliant product.*

In configuration design, a so-called component can be a physical entity, an activity, a process etc. Problem solving hence concentrates on the variation of these components and their arrangement. An optimality criterion or cost function can be applied to rank plausible design candidates [75]. According to Wielinga [76], a classification with respect to configuration design as shown in table 2.1 can be used. The type of components, requirements and constraints used and the assembly generated are of importance.

Components	**Assembly**	**Requirements and Constraints**
Fixed	Fixed	Local & Direct
Parametrized	Skeleton	Incremental
Parametrized Set	Free	Functional

Table 2.1.: Classificatory Aspects of Configuration Design Problems

Focusing on the type of components used, a combinatorial problem needs to be solved, if the set of components is fixed, i.e. determined entirely in advance where all properties relevant for the design task are known and defined. Here, only the arrangement or assembly of the components needs to be determined as part of the solution process such that requirements and constraints are satisfied. As an example, a list or set of known brackets with given dimensions as shown in figure 2.6 left can be named. In contrast, a parametrized set is given if the parameters and shape of each component is known, yet the parameter values for one or more parameters can be chosen by the solution. In the field of naval architecture, e.g. stiffeners can be described as a parametrized set in so far as for each stiffener the cross section is defined by international standards, yet length, thickness or end cut definitions etc. depend on the given design problem, see figure 2.6 center. Finally, a parametrized set of types is an abstraction of the previous class. For each type of component multi-dimensional parameters are present. If a complete bulkhead e.g. is given as a type, not only the dimensions of the stiffeners applied can be chosen. Also, the number and position of the stiffeners is configurable as shown in figure 2.6 right. Therefore, for a parametrized set of types the final generic shape is not given in detail, yet the essence of this type is captured by the set definition.

A second source of variants is defined by the relations that define the assembly, i.e. the arrangement of the components. If the arrangement is known completely, the configuration design task is reduced to the selection and assignment of parameter values for the components of the arrangement. A less restrictive case is given if the skeleton of the arrangement is known. The specific arrangement has to be determined for the solution. The most flexible and most complex case is a free assembly. The arrangement of the components can be chosen by the design task.

Requirements or constraints can be directly applicable, i.e. they restrict or define the value of an individual parameter of a component directly. As an example, the length of a design

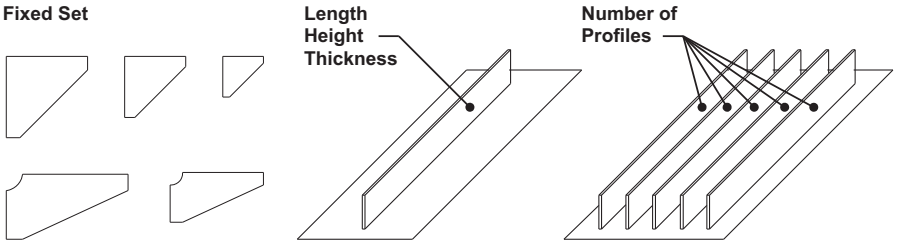


Figure 2.6.: Components in Configuration Design: Fixed Set, Parametrized, Parametrized Set

element can be named. Incremental constraints are constraints that act on parameters which are derived from a combined set of components. For example, the weight of a structure that consists of multiple elements is an example for an incremental constraint where the total weight is calculated as sum over all the weight of the components involved. The evaluation of such constraints is only possible if a sufficiently detailed model is already given. Even more complex, a functional model, i.e. the relation of arrangement and functional characteristics is evaluated for a functional requirement. For a complete vessel, e.g., the relationship between hull shape, engine power and vessel speed can be named as such a requirement.

Based on this classification, parametric design as used in mechanical engineering can be defined as a problem solving method operating on a fixed set of components where an assembly of a known structure is generated. Constraints and requirements are applied locally or incrementally. In contrast, a fixed set of components is arranged for generic layout activities where the structure of the assembly is not known. Constraints and requirements of any type may exist.

In detailed ship structural design, i.e. for the scope of the work presented in this thesis, the design process is often characterized by the application of a fixed set of components where a fixed or skeleton arrangement is given. As requirements, in most cases direct and local requirements and constraints are given. Incremental constraints are used for certain cases like weight requirements. Functional constraints are normally not considered.

Finally, in the process of configuration design, i.e. for the application of problem solving methods, knowledge plays an important role. As shown in figure 2.7, two main categories of knowledge, namely *Problem-specific Knowledge* and *Persistent Knowledge*, can be distinguished.

The former consists of *Input Knowledge* relevant for the configuration of the chosen problem solving algorithm. Requirements, constraints and technological information like manufacturing options are used as input knowledge. In addition, *Case-specific Information* about the design context is of importance. Therefore, information about the design problem at hand like the dimensions of a bulkhead to design, the assembly structure (layout) required or the dimensions of individual elements of the structure are given [8].

In contrast, *Persistent Knowledge* remains valid over multiple problem solving sessions and is used to configure the problem solving method chosen to the general type of design problem. As part of persistent knowledge, *Domain-specific Knowledge* is made up out of data about the domain of intercourse the algorithm is executed in and hence acts as interface between design algorithm and problem domain. For the design of a bulkhead structure, e.g., the

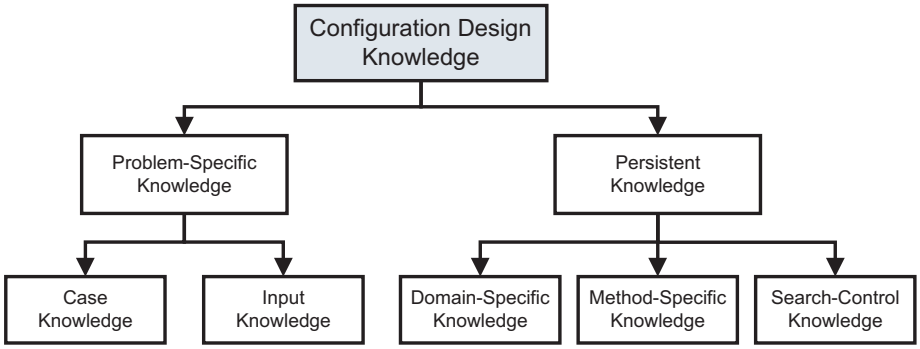


Figure 2.7.: General Classification of Knowledge in Configuration Design [8]

general characteristics for bulkhead design like a functional relationship between the required moment of inertia of the cross section of a stiffener that is located on a bulkhead and the corresponding unsupported span of such a stiffener can be named. In contrast, *Method-specific Knowledge* is used to support and configure the execution of a single problem solving algorithm. Finally, *Search-control Knowledge* is used to support e.g. an iterative solution process or to control the parallel or consecutive execution of multiple PSMs [62].

3. Design Theory, Design Activities and Knowledge-Based Engineering

From a life-cycle perspective, the design process in engineering is one of many interrelated activities and processes. In this chapter, a short summary of the relevant processes is given. Design as heterogeneous, collaborative and parallel design activity is defined. A general design theory as well as the concept of design tasks are identified. The roles of knowledge and knowledge representations in the design process are shown; the term knowledge-based engineering is introduced.

3.1. Design from a Life-Cycle Perspective

The Product-Life Cycle In engineering and naval architecture, a product-oriented perspective is dominant, i.e. all relevant processes are geared towards the development, production or maintenance of a product. Following Sendler [77], when the life-cycle of a product is analyzed, five different types of processes can be identified as shown in the outer ellipsis in figure 3.1.

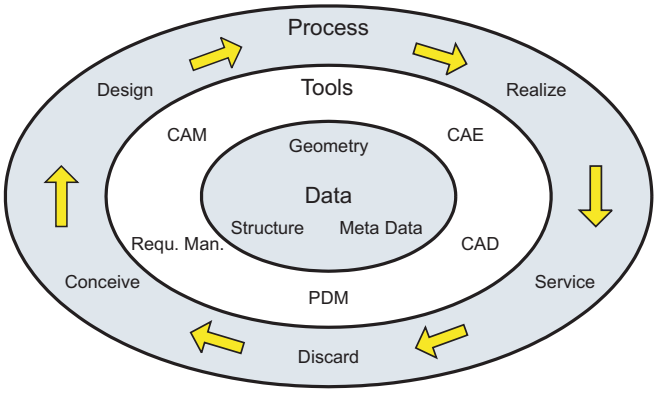


Figure 3.1.: The Product Life Cycle [77]

Starting with a vision, an innovative idea or based on an inquiry by a potential customer, a specification defines requirements and constraints for the product. A design is *conceived*. For this purpose, customer, company, market and regulatory viewpoints are analyzed and taken into account. Major technical parameters are defined. In ship design, e.g., the main particulars and certain details deemed design critical are defined or are developed in close cooperation with prospective partners. A concept design is created so that the feasibility of the approach chosen can be evaluated. E.g., the speed of the vessel with a given engine capacity is tested or total mass and payload capacities are validated.

In the *Design Phase*, a detailed design is developed that addresses all technical aspects of the product. While the concept design provides a sketch of the solution envisioned, here detailed solutions are developed. In naval architecture, e.g., a detailed model of the steel structure is generated for a given project. Solutions regarding machinery, equipment or accommodation are developed. Where necessary, numerical calculations or experiments are performed to assess, validate and optimize candidate solutions. As different aspects interact and influence each other within a complex product, often, an iterative approach is required to find an optimal solution that addresses all requirements and constraints of the specification. For example, in ship design the steel structure often needs to be changed to accommodate requirements specified by machinery or ventilation design. Here, activities of multiple parties may need to be coordinated where changing requirements may require modifications of design proposals.

The *Realization Phase* is focused on manufacturing preparation and the actual manufacturing process. Based on the capabilities available, a planning stage defines methods for manufacturing of the product components. For a given ship design, a modular assembly strategy is developed where multiple building blocks of a vessel are built in parallel. For each part of such a block, manufacturing plans, machining information etc. are created. Where applicable, outsourcing or procurement of certain components is carried out and material as required for manufacturing is ordered. The final product, e.g. a ship, is assembled, launched and tested.

The *Service Phase* is concerned with the operation and ongoing support for the product. In-service information is managed. The performance of the product, i.e. a ship, in real life is monitored and customer support is provided. Repair and maintenance are addressed, if required, and are used to optimize subsequent design variants. Finally, upon the end-of-life of the product, in the *Discard Phase* issues regarding recycling or disposal need to be solved. Dismantling may be required.

These last two phases, the service phase and the discard phase, are currently of minor importance in ship design and operations. With the start of operation of a vessel, a transfer of ownership takes place so that the final product leaves the responsibility of the shipyard. Also due to the low numbers of vessels built of a given design, a direct design optimization for subsequent shipbuilding projects is limited. Finally, dismantling and disposal are normally not considered as objective during the design phase.

In ship design, in each of the first three phases, software tools are used to support the processes. These tools are either phase-specific and highly specialized software packages or can be applied to multiple stages. As design knowledge is created, changed or managed in all stages, such tools operate on product data, i.e. information about shape and topology of a vessel needs to be managed. Meta-information like test results, cost considerations etc. need to be considered. Also, a transparent access to the data available from all life-cycle phases is an important aspect to consider so that data migration and transformation need to be addressed.

Design Processes during the Design Phase Within the product life-cycle, a detailed and realizable definition of the concept conceived is created in the design phase. In ship design, as interrelations are given, the design phase operates in close cooperation with activities performed as part of the conception phase and the realization phase. The scope of the

design process performed during ship design as relevant for this thesis can thus be defined as follows:

Definition 6. *The Design Process is concerned with the detailed, specification-compliant definition of shape, structural and functional information about parts and features of a product (ship) with the objective of optimal performance with respect to identified or defined characteristics like, e.g., profitability, efficiency or functionality.*

As shown in figure 3.2, different categories of tasks are involved as part of the design process in naval architecture [78]. Within and between these tasks a constant exchange of information is given. For the categories shown, two different groups of design tasks can be identified. On the one hand (left), design is made up out of design tasks that focus on the actual design process per se. On the other hand (right), within the design process tasks are required for the coordination of all the activities performed as well as for the integration of such tasks with external activities like the management of human resources or financial aspects.

From an organizational perspective, the product information generated needs to be managed. Here data exchange between, e.g., different IT-systems or involved parties is required. Processes required for manufacturing or development are defined. Where required, scheduling activities are performed to coordinate all relevant design-oriented activities within the design process and to guarantee that these activities cooperate with any other activities performed as part of product development and manufacturing.

As part of the design-oriented design tasks, simulation and experiments are used to evaluate the solution developed with respect to performance. A manufacturability analysis is carried out to guarantee that the solutions developed can be built and assembled. In ship design, accessibility plays an important role as it needs to be guaranteed that a location is accessible at the time, where e. g. the installation of a pipe is scheduled. As part of the actual design task focusing on design, this information is used as input and design solutions are developed such that these solutions adhere to the requirements hence specified.

Focusing on the actual design as performed as part of the design process, i.e. on design tasks as used in the following and as shown in blue in figure 3.2, the notation of a design task can be defined a follows:

Definition 7. *Within the design phase of naval engineering, a Design Task performs the actual shape, structural or topological definition or modification of the parameters of a product. Effects of other design processes are taken into account and are adhered to.*

As shown in figure 3.2 a strong interrelationship between and interaction of the actual Design Task and the other tasks can be identified. Therefore, a common information base is supportive for an IT-based data management as presented in this thesis where extensive information exchange plays an important role. Without on-going coordination and communication, a solution for the often conflicting requirements defined as part of the specification cannot be determined.

In mechanical engineering in general as well as in ship design in particular, design can be described by using concepts from the field of configuration design. Hence, following Huang [79] and substantiating the notion of the Design Phase as introduced by PLM methods, three different scopes can be identified for design as shown in figure 3.3.

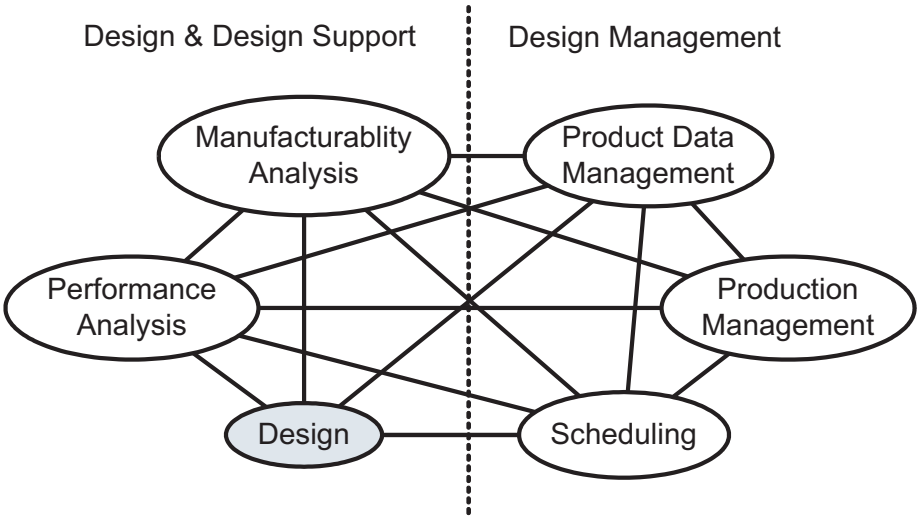


Figure 3.2.: Classification of Design Tasks

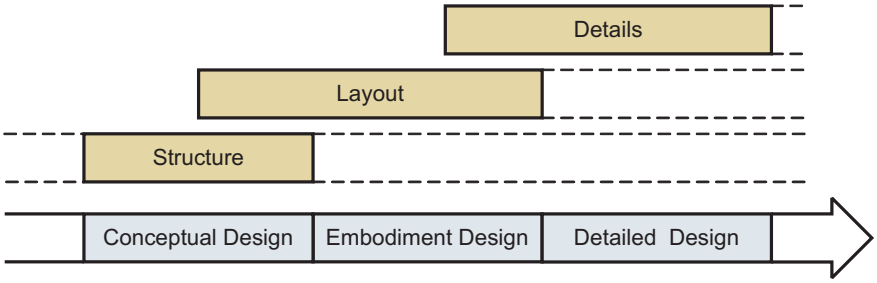


Figure 3.3.: Design as Configuration Design [79]

Here, conceptual design focuses on the definition of the primary structure of a product. It is thus comparable with the conception phase as introduced as part of the product life-cycle. Generic structures are identified and functional principles are developed. Within the design phase as defined for the product life-cycle, two different types of activities can be distinguished if seen from the perspective of configuration design. Firstly, the specific arrangement and schema, i.e. the layout, are defined as part of embodiment design. Here, issues regarding component connectivity and interaction or, e.g., aspects relating strength are addressed. Secondly, detailed design can be reduced to a parametric design problem that needs to be solved. Based on the known product, topology and product structure, parameters for the known product components are derived and the final solution is developed [80].

In ship design, the phase of conceptual design is focused on the definition of the general design of the vessel. Here, main particulars as well as e.g. the mainframe drawings are used to communicate the design intent and rationale. In the following design phase of the product life cycle, as part of embodiment design, all major details of the design problem are considered and adequate solutions are developed. Finally, detailed design is focused on the implementation of known solutions as part of the ship structure. For example, bracket connections are defined where required and using a known table of standard solutions.

Focusing on these three phases and applied to design in naval architecture the design freedom changes over time as shown in figure 3.4. From conceptual design over embodiment design

to detailed design, design freedom decreases, i.e. for a mostly finalized design candidate, consecutive design activities have a very limited amount of freedom. In contrast, during the concept design, the freedom of design is very high. Similarly, the effect on the total cost of a vessel is very high during early design. Here, changes to the design solutions chosen can have a large effect on labor or material costs. In subsequent design phases, decisions with a major impact have already been made, so that embodiment and detailed design therefore do not have a large effect on the overall cost. In contrast, the resources required to fulfill the tasks given increase over the design evolution. Compared to the conceptual and embodiment design, for detailed and production design more resources, i.e more engineering manpower, are required.

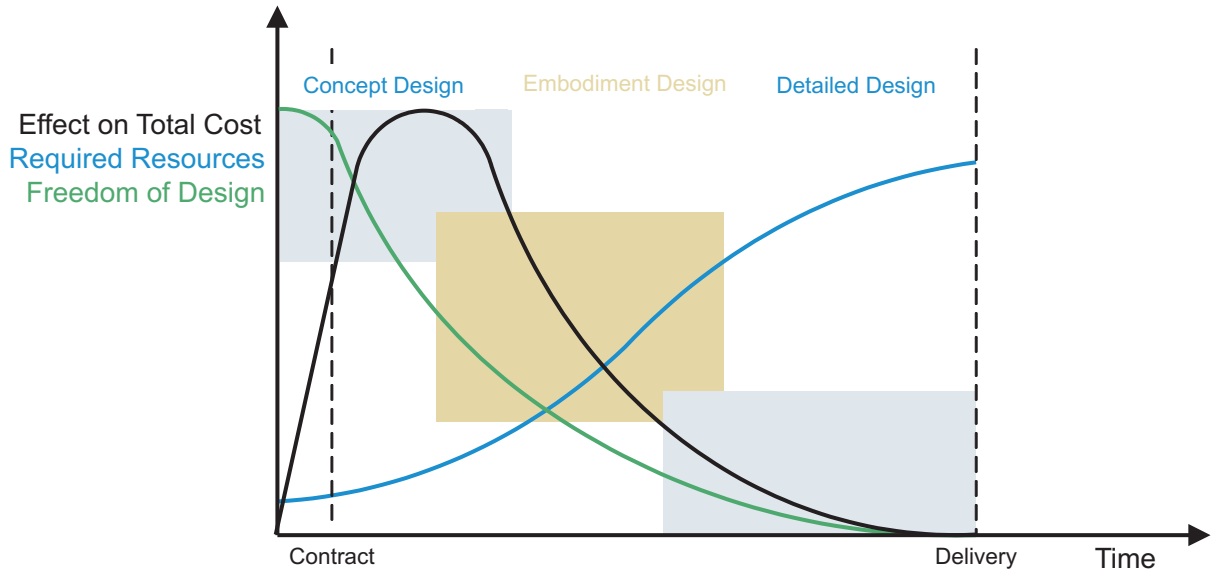


Figure 3.4.: Resources, Costs and Freedom of Design as Function of the Design Phase

For detailed ship structural design, the design freedom hence is limited. Therefore, the direct influence on the total production costs are low while, due to the application of improved design methods, labor costs during the design phase can be reduced. In addition, if an increased usage of optimal solutions with respect to cost and performance can be reached by this approach, the positive effect of such design decisions made during concept design can be intensified.

3.2. Generic Design Activities

Focusing on a design objective, the design task can consist of multiple atomic so-called *Design Activities*. Here, these activities are executed sequentially or in parallel such that an optimal design is achieved. Design activities therefore are of key importance for any IT-based system that concentrates on design support on the level of detailed design. Following [81], the generic notion of a design activity can be defined as follows:

Definition 8. *A Design Activity is a transformation of the state of information by means of a design or problem solving method based on knowledge of the designer or any other acting*

entity and reflecting the context of the evolving design.

In other words - with a design activity, a single design objective should be reached such that all relevant requirements and constraints given are adhered. During the solution process, i.e .within the execution of an individual design activity, multiple steps as shown in figure 3.5 are performed.

Taking existing concepts into account, in a first step a so-called candidate solution is generated. While not being a guaranteed valid solution, this solution presents a first approach towards the design problem given. Here, constraints and requirements given as part of the design context or proposed by other design activities need to be taken into account. In a following step, this candidate solution is assessed and the design intent is validated. Also, functional performance as well as the fulfillment of constraints or requirements are evaluated. In case of a positive assessment, a so-called solution concept is defined. If multiple solution concepts are given for a single design problem, a suitable and valid design is selected from this set in a following operation. Here, management constraints like cost considerations or ratings representing the performance of the individual solutions are taken into account. Finally, the selected solution needs to be implemented within the design model and the actual design is finalized.

In all steps, the design activity may need to be restarted in case of design problems. In this case, the design activity is started anew. For more complex design activities, intermediate results may be used used to adapt objective and input knowledge of the design activity. Hence, alternative solutions can be sought.

Design results obtained may prompt other design activities. The first design activity hereby serves as provider of input knowledge for consequent activities, i.e. constraints or requirements are created for further design steps. In real world situations, often, a design goal may cause several design activities to be executed in sequence or in parallel. Here, the design task terminates when the global objective is reached or no more actions can be performed with the knowledge available.

This kind of process is given for any kind of design activity. More experienced engineers are often capable to perform the actual design activity mentally, alone and - in case of rather simple problems - straightforward. For more complex problems, iterative approaches or the use of tools like analysis software may be required. For an IT-based implementation of such activities, the general process definition holds true. Therefore, this process model as developed by Sim [82] is used as initial concept for the following development of knowledge-based design patterns.

From the knowledge level, design activities can be described by goals, actions and knowledge as shown in figure 3.6. The input and output of the design activity can be expressed as knowledge. Here, the scope and focus of input as well as output knowledge I_K and O_K are influenced by the perception of the design context and the general understanding of the problem domain. As design in engineering can be described as goal-driven activity, a design goal G_D is needed that defines requirements regarding the characteristics of the output knowledge O_K. In other words, the goal G_D defines the objective and motivation, the design activity is executed for. The actual design activity A_D acts as transformation and solution process. Input knowledge like the product data model is analyzed and design steps are taken. The result is evaluated with respect to the given goal and - if sufficient - is returned as output knowledge.

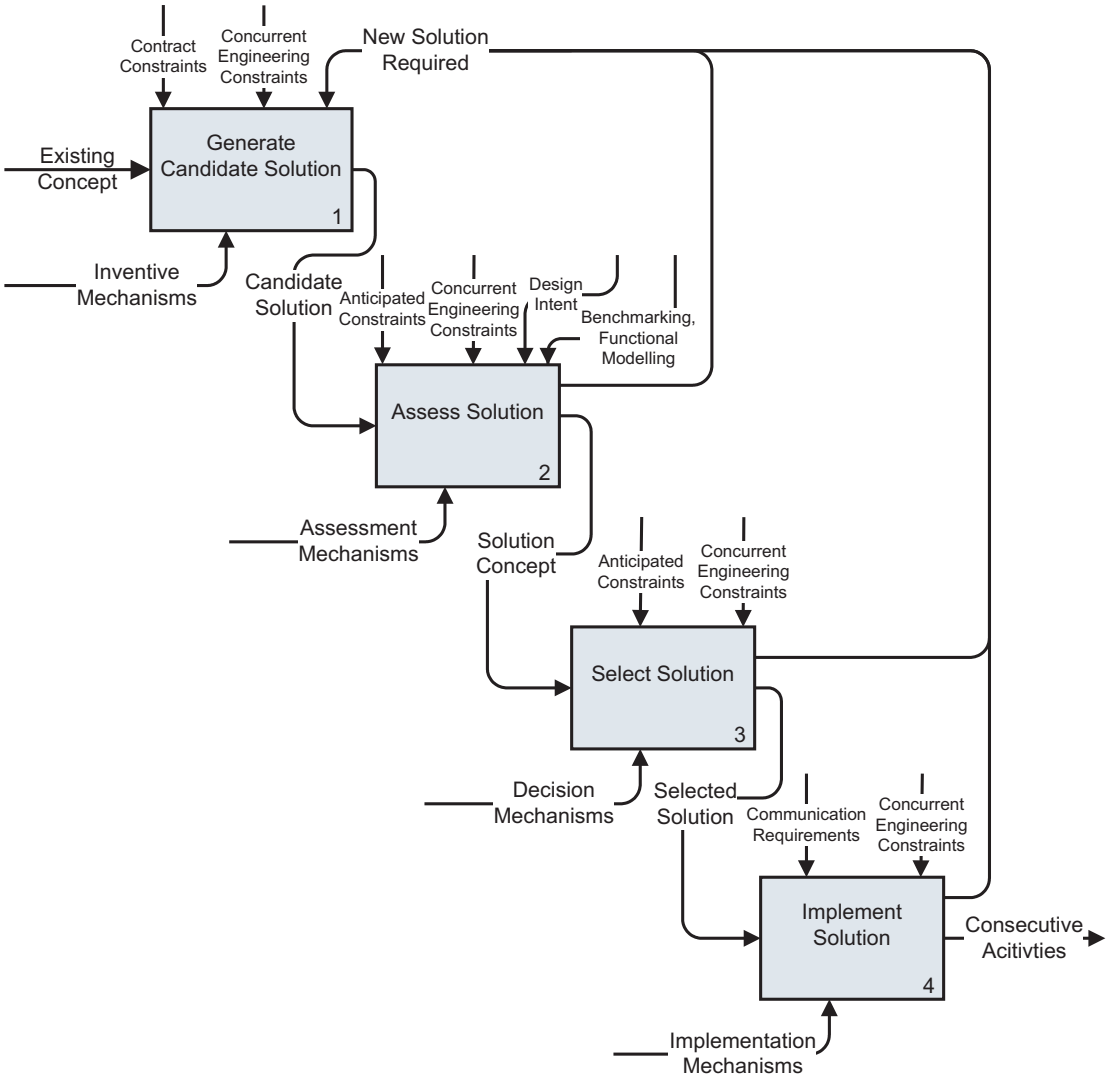


Figure 3.5.: The Solution Process within a Design Activity in the Engineering Design Context [82]

Also, as a problem formulation is often goal-driven, the goal influences the selection of an applicable solution algorithm or design activity A_D. The formulation of a design activity is hence not possible without taking the solution into account [82].

Design activities can be classified according to the concreteness of the design objective and the level of detail present within the design model as shown in figure 3.7. In this figure, a classification of different classes of design activities with respect to the number of details given as part of the design problem and the number of defined and thus known values is shown. On the vertical axis, the level of detail of the design problem increases from top to bottom. On the abscissa, the level of concreteness, i.e. the completeness of the design objective, is shown.

In engineering, the detailed design phase operates on detailed models where a well-known design objective is given. Therefore, for a design problem during this design phase, the design is improved and the level of detail is increased through activities like defining, detailing or standardizing.

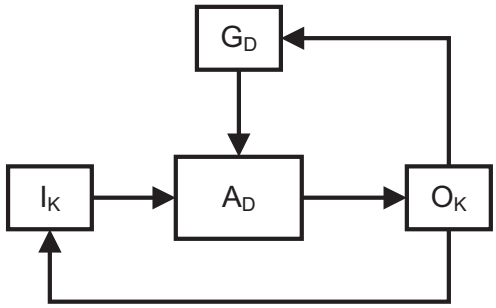


Figure 3.6.: Formalism for a Design Activity [82]

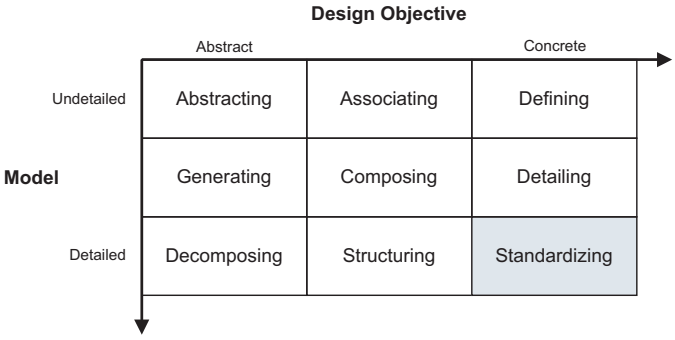


Figure 3.7.: Taxonomy of Design Activities

As shown in blue in figure 3.7, the phase of detailed design operating on a structurally complete model with a high number of known values focuses on the design activity of the so-called standardization [83], i.e. on the application of standardized solutions for clearly defined problems. In the following, IT-based algorithms for this design activity are investigated.

3.3. Examples for Knowledge-based Engineering

In chapter 2, a strict definition of knowledge is given. In the field of *Knowledge-Based Engineering* (KBE) the term knowledge is used less strict. Here, a knowledge-based activity is any activity that at least acts as a provider of information relevant for the design activity and where context-awareness is used to validate design solutions chosen. Using KBE methods, the process of problem solving is supported actively [84] [85].

Definition 9. *Knowledge-based engineering encompasses design activities where knowledge-sensitive, context-aware modeling is performed in a semi-autonomous or fully automatic way. Knowledge representations are a key asset for the formulation of the computational algorithms applied.*

Today, a general and common understanding about the nature of knowledge-based engineering does not exist. In contrast, for the domain of naval engineering various interpretations about what KBE stands for and what kind of approaches are required can be found in research and application use. As typical cases of KBE solutions from the domain of engineering are not known to the author nor are such cases published in the major conference

proceedings in recent years, therefore, based on three examples from the domain of mechanical engineering and engineering in general, some key characteristics of KBE are introduced in the following sections. Different approaches towards the formulation of KBE problems and the respective solution strategies applied are described. For the marine domain, an outline about the situation today with respect to KBE is given in section 4.2.

3.3.1. CIS/2 as Knowledge Provider and for System Integration

In building construction and civil engineering, design activities in the domain of steel structures are focused on the design of steel-framed buildings and similar structures. Here, based on STEP the domain-specific standard *CIMSteel Integration Standards* (CIS/2) [86] has been developed to ease the computer-assisted construction of such structures. For an improved interaction of different applications, a generic data model is defined that supports the information exchange within a heterogeneous software landscape. With this generic data model, all information relevant for a design project from the domain of structural steel construction can be represented. This includes material definitions, lists of stock parts, the shape and topology of a steel structure as well as manufacturing details like weld codes etc [87]. Based on various design stages in steel design, different perspectives on the information present in the data model can be taken. For this purpose, analytical and design- or manufacturing-oriented views are defined [88].

Based on an ontology with fundamental concepts relevant for the design domain, inference processes, see section 2.2, can be used to validate information provided by an application or user. Hereby, automated collision detection performed on the data model is available. Methods for the visualization of geometries and their relationships are present where goal-specific visualizations are generated using a semantic-aware approach [89].

To summarize, the CIS/2 standard serves as knowledge representation and can hence be classified as simple example of a solution for KBE. With this standard, system integration and interoperability of applications are improved, so that a comprehensive use of a common information base throughout the design process can be achieved. More advanced concepts like fully automatic design operations are not supported.

3.3.2. Boiler Design as Example for Context Awareness

For the design of pressure equipment, complex safety requirements and criteria have to be met. Any design proposal needs extensive validation to evaluate the safety compliance with respect to the given operation conditions. Using KBE techniques Ansaldi et. al. [90] presented a solution where major design parameters are parametric and context-aware, i.e. the values for these parameters depend not only on the shape of the equipment but are also calculated using an electronic representation of relevant safety rules. For evaluation, the solution approach chosen, i.e. the candidate solution, is taken into account and validity checks are performed on the values given.

Integrated into a major commercial MCAD system, shape and other constraints are derived from meta-information made available for the project within an integrated ontology-based database. With information like the boiler capacity, the pressure difference and the type of fluid contained given as part of this database, the safety rules can be applied to calculate, e.g.,

the thickness of the boiler plating. In addition, a dependency graph of the geometric model with respect to external information is modeled, such that changes of certain parameters trigger the reevaluation of the requirement definitions relevant for the given context.

For the design activity of boiler design, hence, a set of context-aware and knowledge-based design activities has been implemented. With the approaches presented, a knowledge-based design scenario is given where the design context is evaluated and considered as part of the design process. Rules are used to express and hence externalize safety regulations previously defined in writing. As a result, the quality of the final design can be guaranteed or improved upon. The number of design errors can be reduced. Yet, within the work presented, the selection of adequate design activities is left to the engineer, i.e. the solution strategy needs to be given by the operator.

3.3.3. Codified Solution Strategies in the Aerospace Industry

For wing design on aircrafts, a complex KBE approach has been proposed by Rocca [91]. Based on a thorough analysis of the modeling steps typically performed during early wing design, relevant design activities are defined as part of a solution strategy, where interactions between these activities are specified. Hence, a sequence of actions to perform for the design of a typical wing is modeled within a KBE application. To avoid suboptimal designs, requirements are defined to prevent the execution of consecutive design activities without sufficient performance. In case of design conflicts, i.e. in case of conflicting requirements, the automatic execution of the consecutive series of design activities is terminated and the solution strategy is aborted. Manual intervention is needed. Conflict resolution needs to be performed by an engineer.

As part of the solution, for each design activity, the input and output requirements are defined. E.g. for the design of a wing section, the wing contour is used as input knowledge. For each activity design steps are given. For the example, the wing surface and the structural members of the wing are created by a series of design steps and are returned as output knowledge. In a following design activity, this information can be used for the calculation of, e.g., the tank capacity. As part of the solution, different third party applications are connected to the knowledge-based system. E.g. the final wing contour is evaluated using CFD methods and these results are then fed back into the KBE system.

Here, a so-called codified solution strategy is applied for knowledge-based semi-autonomous design where an automatically determined series of design steps is executed automatically by the KBE system. Hence, the workload of the engineer is reduced and, using the strategies implemented, the consideration of all relevant design aspects can be guaranteed.

4. Ship Design, Design Standards and the Ship Steel Structure

4.1. The Design Process in Shipbuilding

In the field of naval architecture and ship design, the classification of life cycle phases as given in section 3.1 holds true. As shown by Ohtsubo [92] though, for the design phase a more detailed view can be taken. In this phase, all design processes are focused on the derivation of a precise and complete product, the vessel. For this purpose, functional requirements are identified; subsystems and structural components as well as their relations are defined. A qualitative and quantitative description of subsystems relevant for the overall performance is derived so that in a following step shape, material and manufacturing-related parameters can be defined. Hence, a complete structural definition of the vessel or offshore structure is created.

As part of the design processes, design parameters play an important role. These are parameters that are

- given by the customer as primary requirements,
- given by third-party regulations like classification societies or international agreements,
- derived during the design process and dependent on the vessel's characteristics or the design and manufacturing methods chosen.

Based on top-down design principles, in ship design an interactive and cooperative approach to problem solving is taken. Within this process, different design stages as shown in figure 4.1 can be identified.

Based on preliminarily requirements stated by, e.g., the contractor, *Project Design* is focused on the fulfillment of functional requirements on a global level. In consequent steps, on a more detailed level, adequate means are developed to determine solutions as part of the *Basic Design* stage. The draft solution developed during project design is successively refined and, hence, the problem domain is reduced. On a detailed level, as part of the design stages *Equipment Design*, *Steel Design* and *Production Design* horizontal relationships between subsystems or components are created that are taken into account in subsequent design iterations. Here, a parallel execution of these three design stages is required so that a horizontal integration of all design considerations can be achieved.

Within the design process, a vertical interaction between design stages is given. Hereby, the results obtained from an early design stage are used as input or so-called *Specifications* for the following design stages. In contrast, possible *Requirements* from following design stages like, e.g., equipment design need to be preventively taken into account at earlier design stages. E. g., within project design a preliminary weight calculation is determined. The steel weight

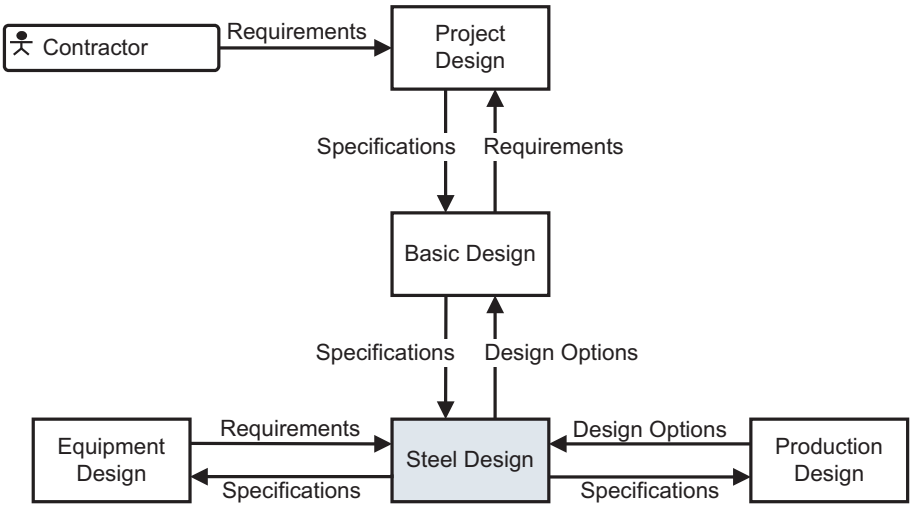


Figure 4.1.: The Design Process in Naval Architecture

defined within this estimate serves as specification for steel design, yet, e.g., the mass of insulation as determined by equipment design can be seen as requirement that needs to be considered during the preliminary calculations.

4.1.1. Project Design

Based on an inquiry of a potential customer or on internal management decisions, a basic set of requirements is given as input knowledge for *Project Design*. These govern key performance characteristics of a ship like, e.g., the loading capacity, speed of operation, operational area and key dimensions like breadth or draft.

As part of the project design stage, a so-called project design, i.e. a preliminary candidate solution, is developed. For this purpose, often comparable vessels with known performance parameters are evaluated, and modified or adapted to the requirements. Variants of this initial concept are evaluated and optimized. A first cost estimate is worked out and the main dimensions of the vessel are determined. The lines drawing and a preliminary general arrangement plan are created. An initial power prediction and weight estimate are derived. For this design stage, creativity and innovative ideas are required. Therefore, knowledge and experience and not plain data play an import role.

4.1.2. Basic Design

Using the specifications developed as part of the project design, these results are refined and improved in the *Basic Design* Stage. Here, the design is updated to incorporate requirements as formulated by the regulations from class societies or internal agreements. More complex computational algorithms or model tests are performed to derive a more accurate power prediction and to optimize the hull shape. The main engine and the engine installation arrangement is defined. Stability, maneuverability and seakeeping are addressed.

The overall steel structure of the vessel is defined where the arrangement and connectivity of

double bottom, bulkhead, side and deck structures is determined and documented as part of the mainframe drawings etc. Based on class regulations, major elements of the ship structure are dimensioned and hence a more precise prediction of the steel mass and its distribution is derived. For critical details like hatch corners or crane foundations, more detailed designs are developed and their interaction with the global ship structure are validated with respect to class regulations.

In this design stage, most design details are not defined explicitly. Yet, based on class regulations, experience from past designs as well as manufacturing cost assessments, generalized standard solutions for common problems are defined. A catalog of standards is assembled that can act as best-practice recommendation for the designer working on the detailed ship structural design, see section 4.1.3. Therefore, technological aspects from manufacturing, assembly and weight calculations as well as strength and fatigue aspects are analyzed or regarded. With the help of this catalog, design solutions for about 80 % of the design of the steel structure are given as shown by [93].

The installation of machinery, electric installations and other equipment also needs to be observed. Requirements of subsequent design phases need to be regarded. E.g., a close cooperation between steel design and equipment or machinery design, see section 4.1.4, is required. From a perspective of steel design, sufficient space needs to be provided for the installation of piping and HVAC installations. In addition, depending on the vessel type additional aspects may need to be regarded. E.g., for a passenger vessel, evacuation procedures and fire protection might be of high importance. As part of the design of an FPSO vessel, autonomous operation and station keeping need to be addressed.

4.1.3. Detailed Steel Design

With the end of the basic design stage, the main characteristics of all main structural members of a vessel are fully defined. Also, typical solutions for most fine-grained design problems are defined universally in the standards catalog as best-practice recommendations. The innovative task of knowledge creation is finished.

In detailed ship structural design, these standardized solutions are applied to all occurrences of the design problems given. Depending on the context, the selection of a suitable solution from the catalog is required. E.g., the connection of a stiffener in the double bottom with intersected plates is designed in detail. The corresponding types for cutout and collar plate are selected and dimensions for these parts are defined within the CAD model.

With the steel model being completely defined, more precise predictions regarding weight can be made. Preliminary assessments performed during basic design are validated and can be improved. Detailed drawings are generated that serve as input information for internal communication, production design or the actual manufacturing process.

Within this thesis, the knowledge-based application of standardized solutions for given design problems is analyzed in the following chapters. As the selection of suitable solutions, i.e. the process of problem solving, may not only depend on the steel structure but also on loads, room designations and equipment or machinery located nearby, special considerations are placed on an automatic consideration of such aspects.

4.1.4. Equipment, Machinery and Accommodation Design

Equipment and machinery design is concerned with the design and definition of the equipment relevant for the ship's operation. Under the term equipment and machinery, here, all means required for the actual ship operation are summarized. This includes systems for anchoring and mooring, life-saving appliances, the engine installation or cargo handling equipment. Staircases, guardrails and elevators are required to support the movement of the crew. The electrical installation of the engine control room or the bridge are required for the control of the vessel. Also, suitable accommodation arrangements need to be determined here.

The complexity of equipment, machinery and arrangement is highly dependent on the vessel designed. While for a freight-carrying vessel, normally only limited efforts are required for design, the opposite is true for a passenger ship. Often, equipment and arrangement not only occupy a certain space for installation, but also require specifically designed foundations etc, i.e. the connectivity of the equipment must be ensured. For this purpose, requirements regarding the design of the supporting steel structure are reported back to the basic and steel design phases. As a result, detailed drawings about equipment systems, machinery or the arrangement can be created. Bills of material can be generated to support the procurement process and weight calculations can be updated.

4.1.5. Production Design

Based on a mostly finished data model, production design is responsible for a detailed definition of the production process of the vessel. The manufacturability of solutions developed as part of the detailed steel design or equipment design phases is evaluated. For this purpose, the solutions are analyzed with respect to the manufacturing processes available and technical regulations applicable. Space considerations, crane capacities and maximum allowed dimensions are considered. Tests for quality assurance are defined.

Optimal manufacturing technologies are sought. E.g., the welding process applied may determine the bevel type and the weld preparation used for a weld and vice versa. For each part and each manufacturing step in the build process, workspaces are allocated. Details like weld sequences are defined if required. Also, detailed manufacturing drawings are created that are used throughout the production process.

The previous stages focus on the process of deriving an optimal solution. With the knowledge created for the problem domain, solution strategies are derived and evaluated. In contrast, in production design as a final design stage the technical realizability is implemented. Incorrectly implemented specifications or ignored requirements may be found here and can hence be reported for correction.

4.2. KBE in Ship Design Today

In ship design, commercial applications and custom software solutions are used throughout the actual design process, i.e. a computer-aided approach towards design, engineering and manufacturing is taken. Within a shipyard, often, a heterogeneous software architecture is

given where interaction and integration are issues that are not fully solved.

In recent years, more advanced features have been added to commercial applications to support the design process and to ease the workload of the engineer. Here, first steps toward knowledge-based engineering as introduced in section 3.3 have been taken, yet no full knowledge-enabled solutions are given. In this section a short overview over the capabilities of selected major commercial applications in the domain of naval architecture with respect to KBE is presented. Also, examples from the current state of research performed on this topic are given. For a more concise review of approaches possible and results obtained see e.g. [94] [95] [96].

4.2.1. Commercial Applications

Aveva Tribon Hull and Vantage Marine In naval architecture, *Aveva Tribon Hull* [97] can be seen as the market leader in the field of structural design, where next to powerful methods for the definition of typical ship steel structures, a tight integration with the following assembly preparation and manufacturing methods is given. Within the system, referential definitions are used for a topological representation of the product model, so that design changes thus lead to consecutive updates of all elements directly or indirectly concerned.

In the field of KBE, Tribon Hull uses a simple pattern-matching approach towards the definition of bracket connections. Based on the governing dimensions of selected profiles or plates suitable design solutions are selected from a known list of candidates. The determination of a location of installation as well as the selection of the final solution needs to be performed by the engineer. More advanced selection methods based on context or e.g. functional aspects are not supported. KBE methods for other design elements are not available.

For piping and cable installation, schema driven modeling can be supported. The general functionality of the cable or piping installation can be defined as schema where all logical aspects of the installation like connectivity as well as requirements regarding pipe diameters or materials are defined as part of such a schema. In a following step, the physical representation of the pipe or cable installation is defined using the schema. Here, pipe diameters for specific pipe segments follow the definitions given in the schema. Changes to the schema lead to updates in the derived installation design.

With *Vantage Marine* as successor of Tribon Hull, the basic functionality of the application is not changed. More advanced methods for integration and extension of the system are provided, though. The *.Net* programming language is supported.

NX from Siemens PLM Software NX from Siemens PLM Software [98] was originally developed as a general purpose, fully three-dimensional MCAD system. As of 2007, the NX Ship Design package extends the functionality provided by NX and provides a set of tools specific to the design process in naval engineering. As required in detailed design, e.g., design methods for the rapid definition of planar steel structures are provided. As, e.g., solutions for the definition of multiple stiffeners within a single design activity are given, series effects can hence be explored. Also, with a fully associative modeling approach, changes of the resulting structure can be performed easily.

For integration, NX offers programming interfaces for most major programming languages

where full access to the data structures of the product model is given. Read and write operations are supported. For KBE, NX - and hence NX Ship Design - is fully integrated with the so-called Knowledge Fusion component.

Using an object-oriented programming language called Intent!, design knowledge can be expressed as rules and facts within Knowledge Fusion. With this approach, a fully rule-based solution strategy can be developed that directly operates on the product model. Also, external sources can be queried; optimization strategies can be perused.

Intergraph Smart Marine 3D Smart Marine 3D, formerly IntelliShip, from Intergraph [99] is a software package with a focus on basic and detailed design. Methods for the definition of hull forms, the steel structure as well as outfitting like e.g. piping, are provided.

Using a top-down design approach, the reuse of existing information is stressed. For this purpose, existing data can be used and custom reference properties can be defined. For the domain of detailed design, rules can be written to create, evaluate or modify data within the product model. Using Visual Basic as language for the definition of the consequence of a rule, design methods can be defined. The topology of the structure can be evaluated and user input can be regarded. Also, parametric modeling features are available for the definition of standardized parts.

For more complex design scenarios, though, the level of expressibility available for rule formulations is limited by the programmatic approach taken. A full rule-based solution with features like automatic agenda filtering is not feasible.

4.2.2. Research on KBE

Macros for Automation in FORAN As shown by Tronstad et. al. [100], for automation the application of macro-based solutions can be applied. With macros, a specific, fixed definition of the design process is given. The domain of intercourse as well as required input and output data is given. Where required, the macros are developed such that user-input is asked for.

As part of the research, the automatic design of certain elements of the steel structure like openings in plate structure and equipment, e.g. macro-generated pipe connections and macro-supported pipe routing facilities, has been considered. Based on FORAN, the approach presented can be used to define simple design activities with a specific and well understood design logic where the focus of development is placed on an electronic representation of the design actions performed. With this approach, optional parameters are difficult to process and more advanced solution strategies or iterative design approaches are hard to implement. Also, access to external data like e.g. a standards database is limited by the often reduced set of functionality provided by the macro language. Therefore, knowledge-intensive design activities are not prime candidates for such an approach.

AJISAI - A System for Conceptual Structural Design For conceptual structural design, an approach based on Catia V5 from IBM Dassault Systems is presented by Toyoda et. al. [101]. Extending the AJISAI system developed by Ishikawajima-Harima Heavy Industries (IHI) and using a set of parametric standard components, a preliminary structure of the basic ship structure is modeled. As standard components, building blocks like complete bulkhead

structures, double-bottom sections or superstructures are used. Best-practice recommendations as well as the reuse of known building-blocks of guaranteed quality are the focus of the research.

Using knowledge-based engineering functionalities delivered as part of Catia V5, parametric engineering techniques are applied for the configuration of these components. E.g. for a double-bottom section, the number of stiffeners or the initial plate thickness are required as input parameters. Adequate means for data input or data retrieval from the model are implemented as part of the system.

As a result, the generated structural model provides a definition of an initial structural design. For each element generated, meta-data with additional information about e.g. the design intent and the design steps performed for this element are provided. These attributes are available for further design steps. From a perspective of KBE, the system developed uses codified solution strategies for the automatic configuration of a given parametric set of components. Here, the solution strategies are developed straightforward, i.e. conflict resolution strategies or iterative approaches towards problem solving are not defined.

DelfPipe for Automatic Pipe Routing In outfitting and using a schema-based design approach, today, the physical routing process needs to be performed manually. Depending on the complexity of the design of the vessel and the compartments considered, this can be a time-consuming task. Different approaches have been presented e.g. by Kang [102], Kuo [103] or Asmara [104].

For the latter, the detailed design process of pipe routing is automated within the system DelfPipe, where the schema and the existing product model is taken as input knowledge. A single design task, namely the design of all pipes with a minimal pipe length and in given locations, is performed. For this purpose, information about the allowed installation ways and their dimensions needs to be given. The product model, i.e. the shape and topology of the steel structure, are analyzed. Using search strategies, optimal solutions are determined, such that the pipes generated fulfill the requirements as defined by the schema, are of minimal length and do not interfere with each other or surrounding elements. For this purpose, a constraint-based approach towards problem solving has been chosen, were design methods and strategies are applied as codified knowledge.

Quaestor: A KBE Workbench For conceptual work, a knowledge based management system named *Quaestor* has been developed by [Q]nowledge [105], a subsidiary company of Marin Research Institute [106]. Within the system, a knowledge base can be defined where data is represented by facts, their properties and the corresponding relations; various means for the definition of calculation procedures are given. Using these algorithms and acting on existing knowledge, Quaestor can generate new knowledge which is automatically added to the knowledge representation. Changing facts may be accounted for by rerunning the solution processes; iterative optimization is hence made feasible. In most cases, an interactive mode of operation is applied in so far, as the engineer is responsible for the selection of a single step to execute, evaluates the results returned by the algorithms and hence commences further design modifications.

Therefore, the system is positioned to provide a development environment for the rapid definition of knowledge-based concept exploration models where a key focus is placed on the

way, knowledge is used during design. More advanced solution strategies or automatic rule-based approaches are not directly supported. The handling of geometric entities is currently not a key capability.

4.3. Standardization

Design as a goal-driven activity is focused on reaching a given set of objectives in a cost- and time-efficient way without violating any of the given requirements and constraints. For this purpose, a search-oriented strategy is employed as shown in section 3.2. Within the domain of detailed ship structural design, standardization as concrete and detailed design activity can be used to support this process.

4.3.1. Standardization - A Definition

For a set of identical problems, the repetitive execution of a search strategy for each problem individually is cost intensive and may result in design errors, as invalid actions are performed within the problem solving methods. Standardization is one approach to solve this problem. According to Kienzle [107], standardization can be defined as follows:

Definition 10. *Standardization is a means to describe an authoritative solution for a specific problem formulation based on the current state of technology where all relevant domain perspectives and the requirements of all parties involved are regarded.*

Standardization is therefore an optimization of a given problem with respect to economic considerations and the technological performance required. In other words, for a given problem, the domain of allowed solutions is restricted by a standard.

In naval architecture, in most cases, standards are focused on a regulation of technical or organizational aspects on a fine-grained, detailed level, where standards are defined in a self-contained way, i.e. all relevant information required for the understanding of the standard is stated explicitly or is given by other standards referenced. For each standard, technological developments or additional experience gained may change the state of technology. Economic changes may require a reassessment of the standards developed.

As a result, standardization augments the innovative and constructive design process of a complex product. While innovation is performed by a creative goal-driven synthesis of this knowledge, for a bottom-up design process, standards are taken as a prerequisite for a methodical approach towards component assembly [108].

4.3.2. Standardization in the Design Process

Within the process of standardization two important phases can be distinguished, namely the phase of standards development and the application of these standards. Based on the product life cycle model as shown in section 3.1, the development or selection of standards is a fundamental task of the conceptual and embodiment design stages. In ship design, based on a fundamental understanding of the design problem at hand, solution strategies as

applicable for the detailed design stage are identified during project and basic design. Hence, encoded instructions for the application of the domain knowledge are developed.

The application of standardized solutions to a design problem is a design activity that operates on a well-known set of characteristics and assigns concrete values to a known set of characteristics, see section 3.2. From a perspective of design methodology, see section 2.3.3, the application of standards in engineering represents a simple configuration design problem. The application of a fixed or parametrized set of components in a fixed assembly situation and in a known design context is defined:

Definition 11. *Design Context defines the essential set of external requirements and internal constraints relevant for a design problem. Constraints and requirements are classified as essential if changing requirements or constraints may lead to changes in design solutions in the defined domain of applicability.*

During standards development, therefore, key factors need to be identified that influence the design. A complex context definition may lead to a large domain of applicability, yet makes the exact and unambiguous formulation of the standard difficult. In contrast, a simple context model may lead to the situation where important requirements or constraints are neglected. Therefore, a compromise with respect to the complexity of the context and the field of applicability needs to be found. For an easy to understand and unambiguous definition, factors of minor effect need to be ignored. The formulation of a standard should be precise, unambiguous and not subject to individual interpretation. The scope should be as extensive as needed, yet not more [8].

In detailed ship structural design, e.g., the design of an endcut for a bulb profile is determined by the function, the profile has to fulfill. If local stiffness and aspects regarding vibration are the only requirements, a sniped solution can be used. In contrast, if global loads should be distributed, the endcut needs to be designed such that, e.g., a bracket connection can be used. Therefore, for endcut design the design context is made up out of the functional characteristics of the profile and the plate, the profile is located on, as well as out of information about, e.g., the neighboring design elements, their properties etc.

4.3.3. Shipyard Standards

As part of the design process in the marine design process, shipyard standards are focused on the technical side of engineering as problem solving activity. These standards therefore serve as technical best-practice recommendations for a selected set of design problems. Often based on international and national standards and regulations, e.g. from ISO or class societies, technical solutions with optimal performance are defined. Hence, a set of company or shipyard standards is developed.

For a set of existing standards, different usage scenarios and formulations can be present. Shipyard standards can be defined so that they can be used as tools to select suitable solutions for a given problem. Different technical solutions for a problem may be compared, so that the standard acts as decision support. For stock parts from third party vendors, standards can define a list of components or semi-finished products available in stock. Finally, standardized manufacturing or assembly procedures can help to determine a cost-efficient design solution.

Shipyard standards play an important role in the naval engineering design process in so far as:

- experience from previous projects is distilled by standards,
- solutions of a satisfactory quality are used if standards are adhered,
- the overall design quality can hence be improved,
- the design can be aimed at an optimal usage of the production process,
- series effects can be explored,
- cost savings can be achieved.

In most cases, standards compliance is mandatory, i.e the solutions as defined by standards need to be followed. Exemptions from the given standards are only allowed if requirements can not be met by a standardized solution.

Focusing on shipyard standards from the domain of detailed ship structural design, for problems of limited complexity and for a given context, a catalog of standardized solutions is mostly concerned with an explicit description of the location, orientation and shape of geometric details. As an example, figure 4.2 shows the design of a bulb profile against buckling, a buckling stiffener, or local vibration. As part of the standard, these solutions are defined such that strength and fatigue as well as aspects regarding manufacturing and assembly are fulfilled implicitly. Welding instructions are mentioned where applicable. The limitations imposed by the context are formulated in writing.

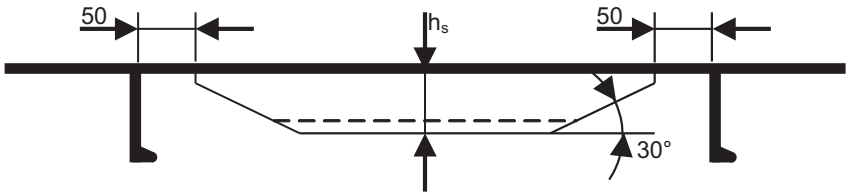


Figure 4.2.: Buckling Stiffener as Example for Standardization

In addition, for more complex composite design constructs, a sequence of design activities to perform may be given. Hence, a solution strategy, i.e. a design workflow is defined where for each step to perform, one or more design standards may be relevant. Here, the engineer is responsible to search for, select and apply the relevant standards.

In the industry, such standardized procedures for complex design constructs are not applied commonly as the number of occurrences where such procedures are applicable is limited. Also, the design context can be hard to define hence making such definitions impossible. In case of design rules that are shared implicitly within the engineering personnel, an explicit standardization is difficult to achieve.

As standard solutions are mostly defined for design problems that occur in high numbers, standardization within the field of naval architecture is mostly used in the following domains:

- Ship Structural Design
- Equipment
- Machinery

For the last two domains, international and national standards play an important role. As these standards cover most of the existing design problems in regular designs, shipyard- or project-specific regulations are a minority. In contrast, in ship structural design a high number of identical design problems exist for most types of vessels. Detailed and fine-grained solutions of limited complexity are common. These are expressed as standards which are adapted to changing requirements or the project at hand, when needed.

As the search for applicable standards has to be performed manually, the application of these standards requires a thorough understanding of the shipyard standards available and the problem domain given. Also, often, the evaluation of the applicability of a standard can be difficult as the input parameters for a design solution as defined by a standard may not be completely known.

E.g., the profile arrangement on a deck structure is, amongst others, determined by global and local loads and vibration aspects. Focusing on the last issue, the behavior of such a deck structure depends, e.g., on the deck load due to machinery, equipment, insulation etc. As the design of the profile arrangement is often developed at a time, these details are not known, the designing engineer has to work with an incomplete set of information at design time.

4.4. Views on the Ship Steel Structure

As shown in section 4.1, in the later design phases design activities are mostly based on an evolving electronic knowledge representation of a vessel where the results of previous design activities are represented as part of the product data model. Depending on the type of design activity performed and the design objective given, different perspectives or views on the data model of the steel structure can be taken.

As introduced by, e.g., Lorenz [109] or Chandrasekaran [61], for ship structural design, three major perspectives can be identified, namely:

- a topological view,
- a hierarchical view,
- the functional view.

Topological View If the focus is placed on the definition of shape and structure, a topological view can be adopted, where a network of connected physical objects is identified [110]. Depending on the granularity chosen, different levels of detail are represented. On a coarse scale, the topological relations of a complete bulkhead structure with neighboring elements can be given as a single relation. On a more detailed level, the definition of the contour of a plate and the surrounding elements is defined by multiple relations.

Depending on the modeling approach chosen, different characteristics of the topology might be considered. In addition to the simple statement that two elements are connected, more advanced constructs might be used to define further information. For example, applying a topological view on a vertical plate stiffener which is located on a plate and that is also connected to neighboring design elements like brackets etc., the topological relations of said stiffener can be classified by more elaborate designations like *connectedTo*, *installedOn* or

nearby. By such more advanced topological representations, therefore, additional knowledge about the product can be expressed. Depending on the type of statements used, this knowledge is not only focused on pure topological statements but encompasses also information from further domains. As example, the relation *installedOn* substantiates the statement about a given connection and makes a statement about the order of installation.

Based on connectivity, a topological view can be used to identify relationships. Starting with a known set of connectivity information, e.g., the execution of strength calculations can be eased. Also, change management can be supported where a change of a certain geometric aspect of an element definition is propagated throughout the known topological network.

Hierarchical View In the field of mechanical and naval engineering, CAD systems are tailored towards the design tasks performed. Often, a preconfigured view as suitable for common design workflows is taken so that common design activities for the definition of parts and assemblies are clearly presented and directly supported [111].

Using a general design methodology, a hierarchy-oriented view uses a top-down approach towards element classification, where all physical members of the product model are assigned to different levels of detailing. This information is represented using a tree structure. On each level within the tree structure, all subordinate elements are included. With the tree structure, navigation and identification of certain parts within the ship structure is eased. Also, a topological or semantic distance between parts can be calculated based on the distance to the nearest common category level.

In figure 4.3, a representation of the hierarchical classification as applied to ship structural elements is shown. Excluding superstructures, all parts of a vessel are classified as subelements of the hull structure. The hull is made up out of multiple sections, where each section consists of multiple so-called assemblies like, e.g., a floor plate, see the example, or a complete tank encasing. Often, for an improved manufacturability assemblies are defined such that primary two-dimensional objects are given. Within an assembly, individual elements like plates, stiffeners, collar plates or brackets are defined.

During manufacturing, in contrast, a bottom-up approach is used. Elements are manufactured that are assembled into assemblies. These assemblies are used to built sections out of which the hull is built.

Functional View Each design activity performed as part of the steel design process is based on functional requirements where the result is a solution that provides certain functional capabilities. For example, the installation of a buckling stiffener leads to an increased stiffness of a plate structure with respect to buckling, hence buckling prevention can be seen as primary functional effect of this stiffener.

If the focus of an analysis is placed on functional requirements or functions provided by parts of the steel structure etc. a so-called *Functional View* is taken. This view concentrates on the relationship of functional requirements and the outcome of a design activity, the so-called functional result that is influenced by the functional requirements given. Using a functional perspective, the detailed technical solution chosen, i.e. information about shape, topology etc., is of no importance as long as the requirements are fulfilled by a solution. As the relationship of multiple functional requirements and their appropriate design solutions are

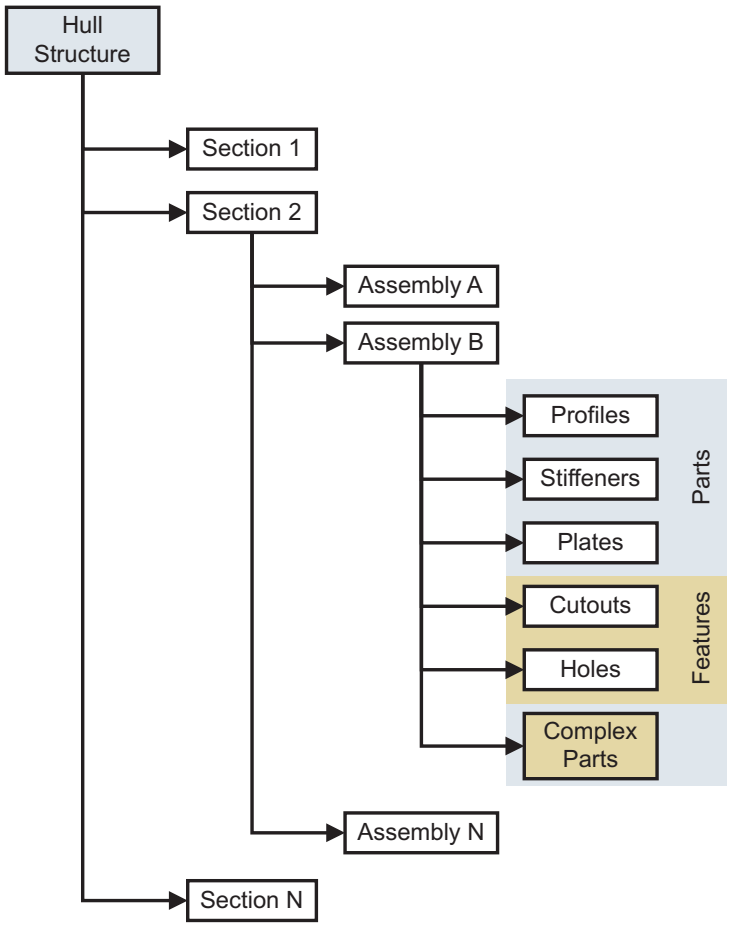


Figure 4.3.: The Ship Structure from a Hierarchical View

graph-oriented, different valid solution paths with different intermediate requirements can exist, if all feasible solutions are taken into account.

From an element-focused perspective, each element in a design provides *Primary Functions* and *Auxiliary Functions*. E.g., as primary function of a bracket connecting a stiffener and a plate load transfer can be named. An element can provide one or multiple primary functions. Also, a design solution can provide additional auxiliary functions. E.g., a pillar in a cargo deck can be also used as vertical cable run between decks. In this case, load transfer and vertical support are primary functions; the cable routing option is an auxiliary function.

As the objective of ship structural design is to define the ship structure according to a set of functional requirements, for this purpose, physical objects with a certain shape, contour and topology are designed or selected. Out of these elements the ship structure is assembled. Hence, in reality a mostly shape oriented perspective is taken on the structure where functional requirements are satisfied by the geometric assembly. An explicit formulation of the required functions is not given, i.e the design considerations during design of a certain object is not saved within the product model nor is it directly visible from drawings depicting the solution.

The issues caused by such implicate definitions of functional results are shown in figure 4.4. Here, using a midship section of the steel structure of a medium-sized special purpose

vessel as example, a selection of primary functions as provided by the structure is shown. Local reinforcements are defined within the structure to support equipment or machinery; design elements are given to provide a watertight compartmentation of sufficient strength. Openings are defined for pipe installations or as a means of weight reduction.

Therefore, as the reconstruction of a functional view from geometric representations or drawings is only feasible due to an inference process, an engineer needs to develop his own thoughts regarding the purposed design intent based on documentation, experience etc. Hereby, misinterpretations may occur which may lead to incorrect design modifications etc.

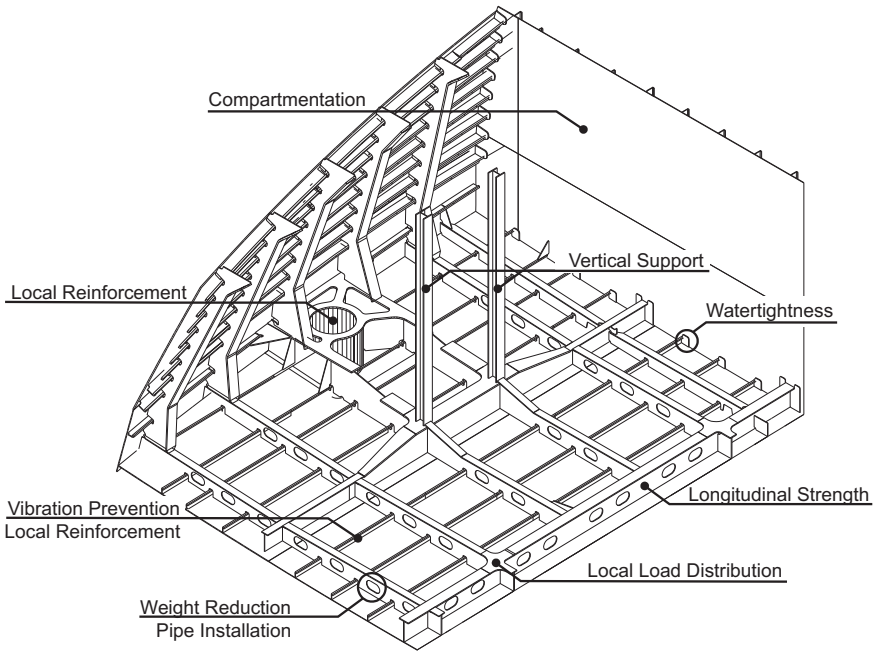


Figure 4.4.: Primary Functions of Ship Structural Elements

4.5. Classification of Structural Design Elements

The primary functions of a vessel's structure are provided by the major elements of the structure. For the ship hull these are mainly [112] [113]:

- longitudinal girders,
- transverse frames, bulkheads and floor plates,
- horizontal decks,
- hull plating.

The general characteristics of these elements are defined in the design stage of basic design, see section 4.1.2.

In detailed ship structural design, solutions for design problems with local scope are developed. Details of the ship structure are defined. For this purpose, different types of objects are used. In the following a categorization of these objects as follows is used:

- Parts
- Features
- Complex Parts

Due to brevity's sake, a detailed explanation of the parts, features and complex parts used in detailed ship structural design is not given. Reference is made at this point to other sources in the literature, see e.g. [114] [115] [116] [117] [118].

Parts denote real physical, atomic objects. A part consists of material. A part is not assembled but is manufactured from a single piece of material by certain steps of manufacturing. Examples for parts are, e.g., brackets, collar plates or single stiffeners made out of stock profiles. Welding or other joining technologies are not used.

Features define geometric modifications performed on an individual part. The major characteristics of the contour and shape of the part are not changed, though. E.g., a notch defined for an improved weldability of a certain plate modifies the contour of this plate part. An endcut definition for a stiffener is used to specify the correct preparation of the ends of this stiffener. A flange defined for a certain type of part is defined using a feature definition. As a feature is normally dependent on functional and shape aspects of the part it acts on, features are only valid for a certain set of parts.

Finally, *Complex Parts* are assembled out of primitive parts and features where the number of parts involved is low. These can be seen as micro assemblies. E.g., a stiffener crossing or a girder are complex parts, where the girder element is built out of two primitive parts, web and flange namely. The distinction between complex parts and assemblies is difficult and subject to personal interpretation. In this thesis, the term complex part is used for assemblies that offer a high potential for reuse. Complex parts can be used as a ready-made design solution for similar design problems within a given project where an individual adaption to the design problem is not needed.

5. Test Cases

For the development of knowledge-based algorithms as presented in this thesis, a selection of design scenarios is used. Each so-called *Test Case* is selected so that a common type of KBE design problem can be represented. For this purpose, typical design problems from the domain of ship design are used. A hierarchical view is adopted, where examples from all three major categories of parts, i.e. from parts, features and complex parts are present.

The selection of such test cases is grounded on an analysis of suitable fields of application for knowledge-based engineering. Based on a first evaluation of commercial aspects, see section 5.1, the issue of feasibility and complexity of the development is analyzed in section 5.2. For the domain of ship design, a qualitative review of design errors and quantitative statistics about typical solutions used in naval architecture as given in section 5.3 and 5.4 respectively introduce potential test cases. Using the data collected, a selection of design problems suitable for knowledge-based standardization can be derived, see section 5.5. Here, the design examples selected as test cases are introduced.

5.1. KBE from a Commercial Perspective

The development of design standards as well as the design and implementation of knowledge-based computer algorithms which support and execute standards-compliant design activities is only suitable for certain cases. As shown in figure 5.1, in all cases the application of KBE solutions results in higher running costs during the development phase. After deployment, i.e. after the design processes have been migrated towards these KBE solutions, a reduction of costs can be achieved if the system works as expected. As shown by the two vertical lines depicting two service times of the KBE solution named I and II and a general development of costs given by the curve A, it can be seen that a sufficient time of application is required to reach a positive return of investment. Similarly, the savings rate, i.e. the absolute reduction of running costs compared to a conventional approach has a major impact on the economic profitability, see the sketched line B which shows an unprofitable scenario for the timespan considered.

Therefore, for the deployment of a KBE solution in a commercial environment, possible use cases for standardization need to be evaluated with regard to the following factors:

- Number n of design problems of a given type,
- Defect Rate r_D of incorrect design solutions,
- Cost c_E resulting from each design error.

As the process of standardization requires considerable investments for each design problem, a sufficiently large number n of occurrences of the design problem evaluated needs to be

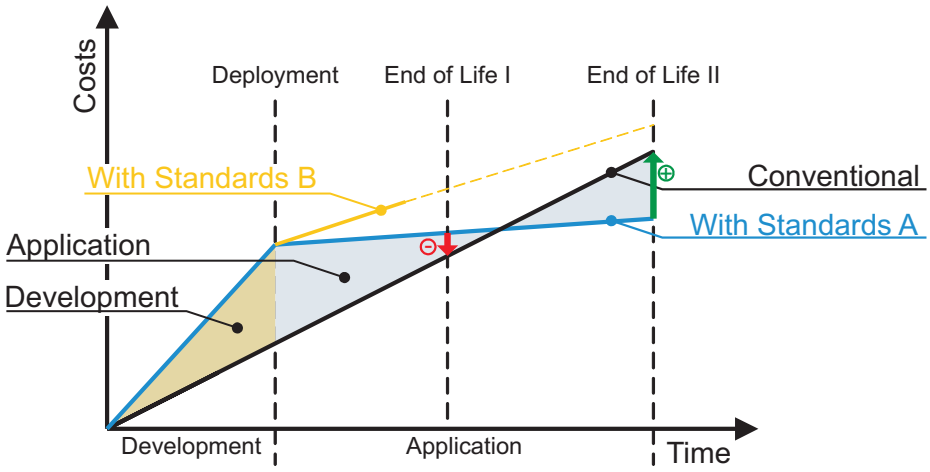


Figure 5.1.: Economic Rentability of KBE Solutions

given. If possible, synergy effects over multiple design projects should be given. The design activity should be used - preferably unchanged - for multiple vessels designed.

A high error rate r_D for a specific design problem highlights issues in the current design process. The usability of the software used or standardization activities already present offer potential for improvement. Training of the engineers and communication strategies can be reconsidered. In this case, the application of IT-based automatic systems can be used as a means to reduce the defect rate. In contrast, if the current design methodology leads to mostly error-free designs, knowledge-based standardization procedures can ensure that the error rate is kept constant over time independent of team changes. The build quality is not improved, though.

Finally, the cost c_E for the correction of a design error consists of direct and indirect costs. Direct costs contain the labor time needed for the design changes. In case of an early discovery, only the CAD model needs to be changed. Changes to related design solutions may be required. If the error is identified during production, significant costs may arise. Parts need to be manufactured again. As other manufacturing tasks may be delayed as well, indirect costs may occur, too.

Therefore, from a commercial aspect, a situation where a single specific design problem accounts for a significant number of design errors with serious monetary consequences is a prime target for the application of KBE strategies.

5.2. Feasibility of Development and KBE Aspects

From a development point of view, the aspects mentioned in the previous section are valid, yet may be perceived with a different connotation. Also, other factors need to be taken into account, namely:

- Reusability of a design solution,
- Applicability of standardization methods,
- Complexity of the design problem,

- Availability of context-sensitive knowledge,
- Limitations of the CAD system functionality.

From a perspective of standardization, reusability is influenced by the number of occurrences and the error ratio. For an improved reusability, no dependency on project-specific definitions should be given. Explicit and concise formulations should be applicable for the formulation of requirements and constraints so that methods of standardization can be used.

The complexity of the design problem should be reasonable. For trivial design problems, a full knowledge-based algorithm can be seen as overkill. For very complex problems, an unambiguous and complete problem formulation using KBE approaches may be difficult or impossible to obtain. With increasing complexity the number of requirements and constraints grows exponentially. Boundary conditions can be interrelated, so that only iterative solution strategies may lead to optimal solutions. Solvability can not be guaranteed.

For the solution of design problems, knowledge is required, see chapter 2. The knowledge may directly belong to the problem at hand. E.g., for a bracket connection the dimensions of the profiles to be connected are parameters of directly involved elements. The information is present as part of the CAD system's data model. Also, further information may be required. For the aforementioned bracket, a flanged bracket may be recommendable if the space available is restricted. Information not directly present in a shape-based data model is required. Therefore, the availability of information and knowledge required for the design task is of key importance. A knowledge-based solution can only work automatically if this information is already known at the time, the design task is performed, and if it can be integrated into the design system. Interfaces between different software systems may be required.

Finally, a KBE solution is targeted towards the software infrastructure available on a shipyard. The capabilities of a KBE solution are influenced by the software infrastructure given in two ways. On the one hand, any kind of system integration is limited by the capabilities offered by the interfaces of the CAD system and other software components used on the shipyard. On the other hand, the functionality offered by the design applications also influences the selection of suitable areas of application. Often, the focus is placed on areas, where the current approach offers suboptimal solutions. The design workflow and hence the labor time needed can thus be reduced significantly.

5.3. Cause and Effect of Design Errors

"Where wood is chopped splinter must fall!" As expressed by this popular saying, in ship design errors are made. Errors that have varying consequences and that normally can be classified by:

- the source of the error,
- one or multiple causes why the error has taken place,
- the error type.

Applied to the field of naval architecture, a deeper understanding of the existing situation as given on shipyards today can be gained with the help of these three categories. Approaches to the introduction of improved standard-compliant design methods can be found.

Focusing on the first classifier, in table 5.1 the point of origin of design errors within the typical design process in naval architecture is shown, see the second column. As design errors are seldom detected instantaneously, the design stage where the error was detected is shown in the rightmost column. In this column, the sum of entries results in more than 100 %, as such errors may be detected in more than one design state simultaneously.

Design Stage	**Source**	**Detection**
Basic Design	7 %	1 %
Detailed Design	66 %	6 %
Manufacturing Preparation	17 %	31 %
Drawing Generation	8 %	26 %
Manufacturing	2 %	51 %

Table 5.1.: Source of Design Errors and Detection Rate

This data is based on an extensive review of the experience seasoned engineers have gained. For this purpose more than 20 detailed interviews with the engineering personnel of major shipyards in Germany were performed [118]. As part of each interview, questions regarding the individual perception of the design process and problems perceived therein were asked with the focus on gathering further insight into the number of design errors typically encountered during design. In total, approx. 180 errors were identified by the interviewees. As a result, information regarding sources of such design errors, the location of detection within the design process and the typical time needed for correction could be determined.

From table 5.1 can be derived that the majority of the design errors is caused during detailed design. Manufacturing preparation is the second largest contributor to design errors. Detection of errors present in the data model is poor. In detailed design, defects are detected seldom. About 50 % of all errors are not detected before manufacturing. The design stages manufacturing preparation and drawing generation account for more than 50 % of all errors found; less than 10 % of all design errors are detected while the actual design process takes place.

For design defects, different causes can be identified. With a single defect possibly being alloted to different defect categories at the same time, human errors and issues regarding design knowledge are the main reasons for design errors, see table 5.2. In contrast, errors in the design software used as well as other unclassified causes account only for a minority of all design defects reported. Therefore, an approach based on KBE with a standards compliant, controlled and automated design process which focuses on theses categories can be a major contribution to a reduction of design errors.

In detail, human errors (73 %) can be caused by oversight or general mistakes. Relevant design criteria may not be identified and are hence ignored. General mistakes like invalid data entry etc. can be present. Usability issues in the software used may facilitate these mistakes. Without KBE techniques, these errors are not identified in time as often no automatic validity checks for the data entered exist. For errors caused by informational deficits (28 %), information is either not known (10 %), incorrect (10 %) or outdated (7 %). In all three subcases, incorrect design decisions are made. Knowledge management can improve the situation.

Cause of Defect	Percentage
Human Error	73 %
Information	28 %
Software Error	13 %
Other	8 %

Table 5.2.: Causes of Design Errors

Focusing on the type of design defect, for each design error encountered in an interview, questions regarding the category of such errors where asked. Based on this subjective data, a categorization of design errors as shown in table 5.3 can be made.

Category	Percentage
Design Knowledge	30 %
Manufacturing	20 %
Organizational	19 %
Material	7 %
Other	25 %

Table 5.3.: Categorization of Design Errors

In the category of *Design Knowledge*, the design problem is not understood in full context. Hence, defects may stem from the fact that, e.g., the orientation of a profile is not taken into account. Design elements used for a solution may not be from a catalog of allowed and standardized parts or features. Design errors from the *Domain of Manufacturing* can show, e.g., missing tolerances or improper weld preparations performed. A suboptimal manufacturing process is given. Missing or incorrect position numbers may lead to *Organizational Problems* during assembly. Preferred material dimensions may not be considered consistently. Hence, organizational factors may not be regarded in full detail. *Material-related* design errors are based on, e.g., an incorrect selection of not-preferred profile types or a missing adherence to preferred plate dimensions. Manufacturing costs may be increased unnecessarily.

It can be seen that the majority (69 %) of all defects can be assigned to the categories of design knowledge, manufacturing or assembly preparation and organizational aspects. In all these categories, KBE applications may help.

5.4. Quantitative Statistics for Selected Design Problems

As shown in section 5.1, sufficiently high numbers of similar design problems are required to justify the application of knowledge-based engineering methods. As an example, for three current designs, statistics about selected parts of the steel structure are given in table 5.4. Here, the designs of a medium sized container vessel with $L_{PP} \approx 200m$, an offshore supply vessel of $L_{PP} \approx 90m$ and a seagoing tug with $L_{PP} \approx 75m$ were evaluated. The numbers presented are based on an evaluation of the corresponding CAD models.

Vessel	Plates	Profiles	Brackets	Clips
Container Ship	14380	15350	6265	5150
Offshore Supply Vessel	9400	12480	6090	1500
Seagoing Tug	5800	8970	4760	650

Table 5.4.: Statistics about Major Parts of the Ship Structure

For each of the designs, high numbers of plates and profiles are given where the number of plates is correlated with the size of the vessel. A weak correlation with respect to the number of profiles is given for the number of plates. The number of brackets is roughly constant across the designs. With regard to the usage of clips, differing numbers are given. Here, vessel size seems to play an important role. Clips are used for large vessels where high loads are present. This is an effect of the requirements as defined in the relevant class rules.

For cutouts, four sections of the selected container vessel were analyzed in more detail. Each section consists of a complete slice of the ship hull in the midship region with a weight of approx. 600 t. Using the CAD model available, as a result, the types of cutouts used as well as respective numbers can be calculated as shown in table 5.5. Cutout types are denoted by an identifier. An identifier of *9999* represents a special case where the cutout definition algorithms of the CAD system are used for purposes other than intended. The number of cutouts of a specific type is given for each block and as sum.

Cutout Type	Block A	Block B	Block C	Block D	Total
900	158	48	106	28	340 (49 %)
910	47	88	0	21	156 (22 %)
920	20	30	26	10	86 (12 %)
940	6	36	0	0	42 (6%)
950	2	0	3	0	15 (2 %)
9999	0	45	0	0	45 (7 %)
Other	8	0	7	0	15 (2 %)

Table 5.5.: Statistical Analysis of Cutout Types

As a result, for the majority of all design problems, a set of standardized design solutions is used. For almost 85 % of the definitions found, the cutout types *900*, *910* and *920* provide a feasible solution of sufficiently high quality. These are shown in figure 7.13 and 7.15 on page 96 f. The standardized types *940* and *950* are used in about 8 % of the cases. For the remainder, the special case *9999* as well as custom definitions are used.

5.5. Selected Design Problems as Test Cases

For the presentation of the solutions developed as part of this thesis, a set of suitable design problems is selected. Taking a function-oriented view, the set of design problems was selected such that:

- real-world problems in the domain of ship design are approached,
- common design issues are investigated,
- different aspects related to knowledge-based design algorithms can be highlighted,
- a set of problems of increasing complexity is given,
- the limitations of the CAD system chosen are covered.

As reference CAD system *Tribon Hull* was chosen.

5.5.1. Bracket Placement as Basic Example

The selection and placement of a single part from a catalog of standardized solutions in a well-defined design context can be seen as simple case of knowledge-based engineering. For this purpose, the design of a bracket serves as example and is used to introduce fundamental concepts as well as formulations of KBE algorithms.

As brackets are only used to provide load transfer between members of the ship structure, secondary or auxiliary functions do not exist. Hence, for each bracket design activity, the structural parts to connect are of importance, only. As examples, typical use cases for bracket placement are given in figure 5.2.

In the two figures shown left, the use of brackets for the connection of a profile and a plate structure is shown. In the first example, a flanged bracket is used to connect the profile with the in-plane oriented wall structure which is located on the opposite site. In the second use case, a so-called attached bracket is applied for the direct connection of two profiles, i.e. the bracket is not placed on top of the respective flanges, but next to the profile's sides. In case of ship structural elements with differing orientations, a bracket solution as shown in figure 5.2 third from left can be applied, where any loads acting on the profile oriented in parallel with the drawing pane can be transfered into the orthogonal profile located below and vice versa. As last example, a peaked bracket is shown on the right.

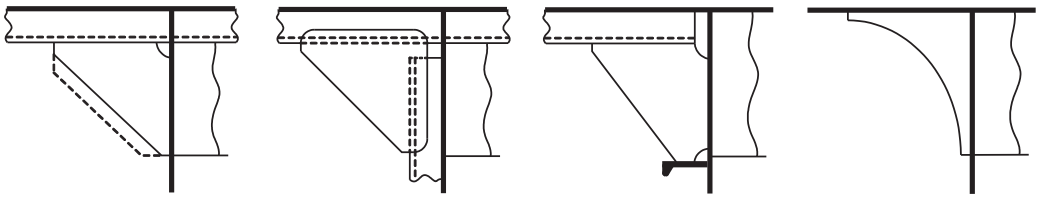


Figure 5.2.: Typical Use Cases for Bracket Placement

As test case, the placement of a single bracket for a profile-profile connection is chosen, see section 7.1. In this test case, a clear single-objective solution algorithm can be used, as for the design standard analyzed, an unambiguous correlation of required bracket load and size is given.

5.5.2. Functional Design for Notch Selection

Feature definitions are more complex cases of knowledge-based design where the selection of a suitable solution is followed by a modification of the part the feature acts on. As test case, the selection, dimensioning and placement of corner cutouts is used. Here, the design problem is characterized by:

- the definition of a single feature,
- with multiple independent design objectives.

As part of the design activity, a suitable notch shape is selected. The diameter of the notch needs to be determined and the contour of the plate is changed accordingly. As notches are mostly placed for an improved weldability or as water opening, different design objectives can be given.

5.5.3. Collar Plates and Cutouts as Complex Parts

Finally, for the design of complex parts a standardized combination of parts and features is used. More than one standardized design element is involved in the design process. As test case, the detailing of a profile-plate intersection using cutouts and collar plates is chosen. With the high number of profiles given in every ship structure, this can be seen as a common design problem. Compared to bracket design, the design task for such intersections approaches more advanced problems, namely:

- a complex part that consists of one part and one feature,
- multiple primary and secondary design objectives.

For the definition of the complex part, the properties of the cutout as feature and the collar plate as part - if required - need to be derived. The interrelations of these items are of importance, i. e. a specific collar plate is only valid for one or a limited number of cutout definitions. Different design requirements like, e.g., space requirements or restrictions regarding the watertightness of the solution are present. Depending on the design problem, different objectives must or should be fulfilled. If multiple objectives are required, an optimal solution needs to be found.

6. Application Design and System Architecture

For the application of knowledge-based algorithms and to achieve an improved standards-compliant detailed design, a software system has been developed. In this chapter, the overall system architecture is presented, see section 6.1. Important components of the system are identified. The interaction of these components is analyzed. In section 6.2, a detailed introduction of the core solver components is presented. Finally, strategies applied for the definition of an electronic knowledge representation for standards and design knowledge are presented in section 6.3.

6.1. General System Architecture

6.1.1. Cooperation of CAD and KBE Systems

For the development of knowledge-based solutions, the data available as part of the product data model of a CAD system is of key importance. Also, for automated design tasks and design patterns in knowledge-based applications, it is important that results can be reused. Without adequate options for reuse, the information generated is of less value. Significant contributions to the design process can thus not be made.

For a given software infrastructure, therefore, some cooperation between CAD and KBE system is required. KBE solutions are developed such that the scope of the results is compatible with the CAD software and its product data model. To achieve optimal results, for the system architecture a decision regarding the level of cooperation needs to be made. Two general approaches can be distinguished, namely:

- Integration,
- Interaction.

In figure 6.1 (left), a complete separation of a CAD system and its product data model from the knowledge component is shown. The KBE system is developed as independent separated solution. Means for data exchange are provided using defined *Application Programming Interfaces* (APIs). The data of each system is hence made available for use by other applications. Using multiple interface implementation and ignoring platform specific solutions, a single KBE system can interact with more than one CAD system.

If the knowledge component is developed as an integral part of the CAD system, a situation as shown in figure 6.1 (right) is given. The KBE solution is realized as a built-in module. Developed as extension of the CAD system of choice, direct access to the CAD data is given. With a fully integrated approach, the complete capabilities of the CAD system can be used.

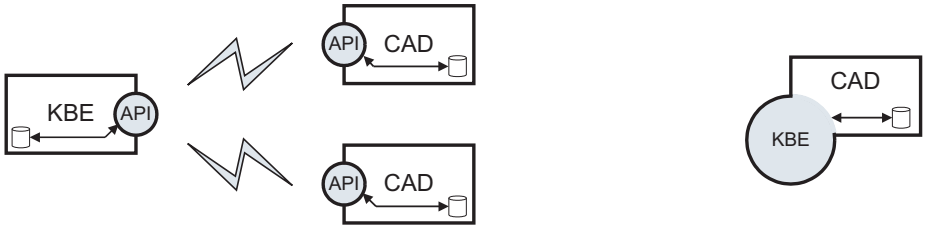


Figure 6.1.: Interaction (left) vs. Integration (right) of CAD and KBE Systems

An explicit communication layer for the interaction of the components is not required. For the development, the flexibility of the solution is limited, though. Only programming languages supported by the CAD system are supported. Useful third-party components written in other languages or the usage of different data models are not applicable.

In contrast, with an interactive approach full freedom regarding the development of the knowledge-based module is given. Often, only changes in the communication layer are required to incorporate changes to the CAD system. The development of adequate communication procedures and, if required, data transformation algorithms is time consuming, though.

For the architecture presented in this thesis, the middle ground has been chosen. The knowledge solver has been developed as an independent component. As the availability of applications for KBE that are compatible with the programming languages supported by the CAD system of choice is low, full benefits of all solutions available on the market could thus be taken. Supplementary knowledge models are defined as subsets of the functionality of the CAD system chosen, because the capabilities of the CAD system have a major influence on the design patterns implemented. The solution is therefore closely tied to the CAD software package. Algorithms were developed with an application-focused point of view.

6.1.2. The CAD System of Choice

As test-bed for the development of KBE-algorithms, the commercial CAE package *Tribon* from *Aveva Marine* has been chosen [97]. As major factors for this decision the following can be named:

- in-house competence and experience,
- third-party requirements,
- extensibility,
- market share.

Tribon offers various methods to access data stored within the system. Versatile and feature-rich capabilities for data extraction, data manipulation and creation exist as shown in section 6.3. Previous experience available was another factor for the selection of Tribon M3. With a thorough understanding of the functionality available at the Chair of Naval Architecture, development could be eased. Finally, the decision was supported by the fact that Tribon is one of the leading software vendors in the domain of commercial ship design.

The development was performed using the *Tribon M3 Hull* application. The successor *Aveva Hull Detailed Design* [119], which amongst other improvements offers an improved database back-end, has only been released after the author's work had started. It was thus not considered.

6.1.3. System Components

In figure 6.2 the general system architecture is shown. Core components of the KBE system are drawn. Data streams and information exchange are depicted by arrows.

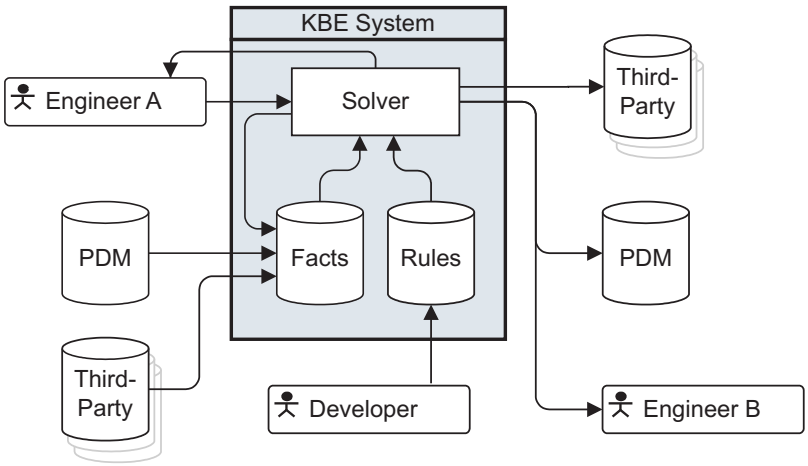


Figure 6.2.: General System Architecture

As knowledge-based engineering is focused on the IT-based application of engineering algorithms, information about the algorithms or rules and the design problem or facts needs to be given, see the corresponding databases as part of the KBE system.

Case-dependent knowledge about the complete design problem as well as method-specific, case-independent knowledge is stored as *Facts* within the system and represents the information base the system is working on. The first is comprised out of data about the design problem like shape, topology or manufacturing related details like weld definitions. Method-specific, case-independent knowledge is focused on the definition of shipyard standards, their field of applicability etc.

In addition to the product data model imported from the PDM or CAD system, the import of additional information from third-party sources may be necessary. E.g. information about loads acting on the ship structure are required. Results derived from FE-based calculation may be provided as separate data source. The order of assembly might be of importance to determine feasible design solutions. For this purpose, interfaces as described in section 6.3 are used. Using a modular approach, changing requirements can thus be encompassed by flexibly adding additional sources. Alternatively, at runtime user feedback can be used as additional input source. An interactive mode of operation is applied.

For the execution of the algorithms, a suitable development and runtime environment is required to execute automatic problem solving strategies. For this purpose, the *Solver* component provides the functionality required to execute the problem solving methods, i.e. the

key logic describing the conditions, requirements and solution steps. These are stored within a shipyard- or project-specific *Rules* database which is maintained by the developers responsible.

Results achieved are made available for reuse within the system. They can be added to the *Facts* knowledge base in so far, as data models present are updated accordingly. Also, results can be made available to the engineer who is responsible for the evaluation of the results. With an explanation about the reasoning steps involved, the feasibility of the design solution presented can be judged. Results can be exported into any connected databases. Here, databases for export can be identical to databases used for import or can be of a different nature.

From a functional perspective of software development, therefore, the following functionalities need to be addressed by the software system:

- Data Storage Facility for the product data model and the known standards. This includes:
  - the persistent storage of data,
  - the integration of existing knowledge from third-party sources, i.e. the import of or access to existing product data models etc.
  - Editor for native data, i.e. data that is defined and maintained within the system
- Formulation of design algorithms
- Solver as runtime execution environment for these algorithms and PSMs
- Output of the results

As the implementation of interfaces for the import of data and the reuse of results is time-consuming, a fully working solution for a complete, automatic interaction is therefore difficult to achieve. It is highly dependent on the capabilities of the external software systems present as data provider or receiver. Therefore, the focus of this thesis is mainly placed on the internals of the KBE-system. An external reuse of results is not considered in detail.

6.2. The Solver

For the execution of codified design algorithms, a solver component is needed. This component parses the algorithm formulations which are written in a solver specific dialect and applies these algorithms on the knowledge base. Therefore, the selection of a solver component has a major impact on algorithm and knowledge model definition.

6.2.1. Solver Components

With engineering seen as problem solving method and depending on the problem type, as introduced in section 2.3.2 two different types of electronic solution strategies can be distinguished, namely via the application of:

- constraint type engines,
- solver type engines.

A constraint type engine is mainly focused on constraint and pattern matching. I.e. for a proposed solution, a constraint type engine is capable to determine, whether the solution matches the requirements given. A solver type engine in contrast is a means to generate solution proposals. If the quality of these proposals can be determined, search strategies can be applied to find optimal solutions [120] [121].

A constraint engine alone can be applied for simple problems with straightforward approaches towards a solution. Often, both types of engines are combined in, e.g., a *Rule Engine*. With a rule engine, discrete aspects of a design problem are separated and formulated explicitly via rules. For each rule, a constraint type engine is used. The solver type engine, then, is responsible to determine the execution order for the rules. Due to the clear separation of constraints and solution steps, an intuitive development process is given. Therefore, for the work presented, as solver component a rule engine is applied.

6.2.2. Drools as Rules Management System

For an automated design of standardized solutions for detailed ship structural design, the general structure of a solution is known, see section 2.3.3. The number of dimensions in the solution domain is limited. Iterative multi-objective and multi-domain solution strategies are not required. Therefore, often a direct rule-based approach can be chosen.

For an application in the domain of engineering, as requirements for a rule-based solver component,

- selective rule activation,
- rule execution ordering,
- custom search strategies.

can be named. *Rule Activation* can be used to select and activate only a set of rules suitable for a given problem. *Rule Execution Ordering* is used to control the importance of rules. It is a means to guide the solution process. Different aspects of the rule base can be emphasized. *Custom Search Strategies* allow the extension of the rule-based reasoning processes performed.

For this purpose, various commercial and open-source rule systems are available on the market, each with individual strengths and weaknesses. A comparison of the major players in this market can be found in [122]. Here, as main component of the KBE system developed *JBoss Drools* has been chosen [123]. Drools is a open-source, rule-based system written in Java that augments the functionality of a rule management system with *Business Process Management* (BPM) capabilities.

The main components of Drools are shown in figure 6.3. Within Drools, the *Working Memory* contains the facts the rule engine reasons about. Data is added to the working memory as Java objects. The *Production Memory* contains one or multiple sets of rules. As part of the solver component, the *Pattern Matcher* is responsible to determine which rules are relevant for the current state of the working memory. Based on this information, Drools determines the rules to be executed. An *Agenda* is derived where the sequence of rules to be executed is defined. In a second step, each activated rule is read and any actions defined are performed. As result, new information may be created and is stored within the working memory.

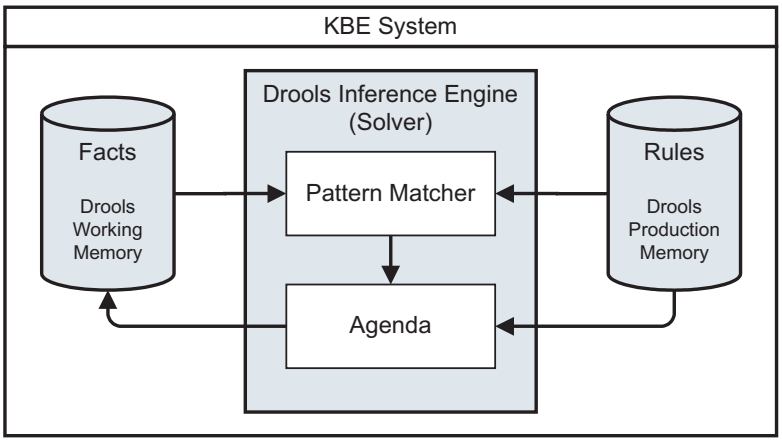


Figure 6.3.: Components of the Rule Engine *Drools*

The integrated rule system offers capabilities like

- rule partitioning and rule activation,
- conflict resolution,
- dynamic agenda and truth maintenance.

For a complex rule base with many rules, categories can be assigned to rules using rule partitioning. So called *Agenda Groups* are defined. Rule activation allows to enable only rules of a given category. Rules defined for different design purposes that may cause conflicts or incorrect design results can be disabled for a given design task.

Normally, the execution order of rules is arbitrary. Depending on the information available, pattern matching algorithms are responsible for determining the order in which suitable rules are executed within an agenda group. Conflict resolution is a means to influence this order of execution. Rules of higher importance or *Salience* are preferred. The solution strategy is changed, so that these rules are executed first.

If a certain rule changes the knowledge base, truth maintenance is used to reassess the agenda given. If needed, the agenda is updated. Changes to the working memory dynamically change the solution strategy.

With the rule functionality focused on determining results for a single problem, *Business Process Management* methods can be used to integrate multiple rule configurations into a design workflow [124]. E.g., in the field of naval architecture, the sequence of design activities performed for the definition of a complete floor plate structure can be defined. Here, each design activity is formulated as an individual rule-based problem. Based on requirements and the design problem at hand, necessary design tasks are executed. Design loops, if needed, can be performed.

For the work presented, the *Drools Flow* component has been used. This component is a workflow or process engine that allows the advanced integration of processes and rules. A process or a workflow describes the order, in which a series of steps needs to be executed. Each step is defined as a set of rules. Data present in the working memory as well as input from external sources can be used to control the execution order [125] [111].

6.3. IT-based Representation of Design Knowledge

6.3.1. The Knowledge Model

Within the rule-based system, for each design problem relevant data needs to be present. For this purpose the working memory is populated beforehand and relevant rules are activated. Different data sources may be used to provide data. From a knowledge perspective two different types of knowledge can be identified, namely:

- quasi-static, method-specific information,
- problem-dependent data.

Quasi-static information consists of all data that is not dependent on the design problem at hand. E.g., a list of profile types and dimensions in stock is valid for multiple design problems within a given project. Parts and features defined in the shipyard standards also remain unchanged for a design project. A persistent storage solution is hence required which allows for the reuse of such information. Solutions for the management of and changes to the shipyard standards etc. are required [126].

Also, for each design problem, problem-dependent knowledge exists. The design configuration like shape, topology and connectivity etc. differs from design problem to design problem. This information is of a dynamic nature. Reusability is not given. Often, this information is provided by third-party applications like the CAD system.

Implementation Details As introduced above, the working memory as temporary storage solution needs to be populated with all relevant data, see figure 6.4. Information is imported from various sources. Transformation operations might be performed. Hence, before execution, the information is made available to the reasoning process [127].

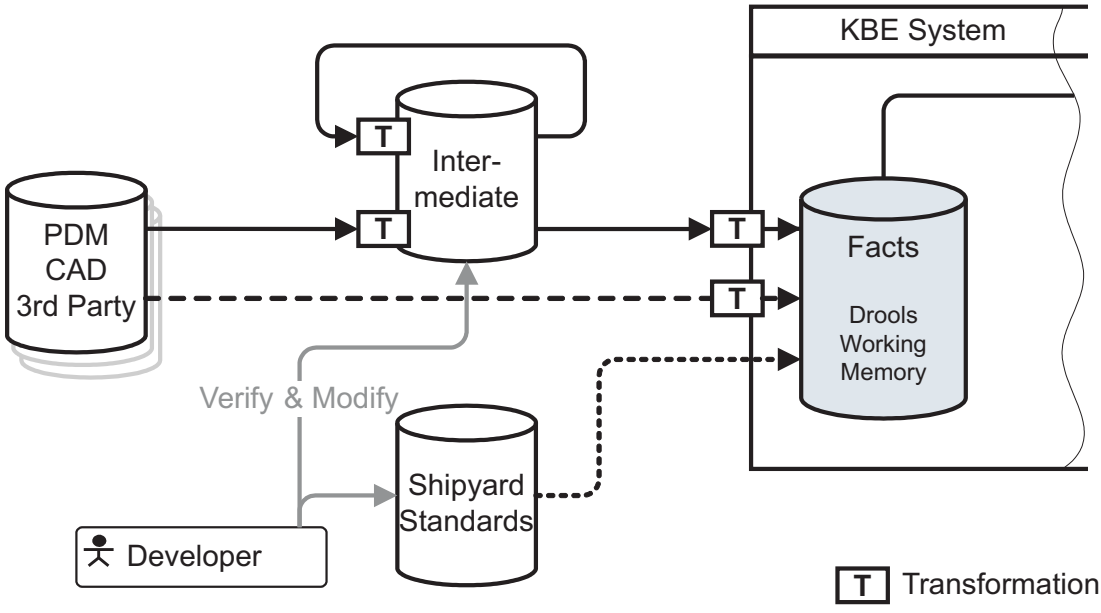


Figure 6.4.: Layers of the Knowledge Model

For the implementation in software, three different approaches are considered. Firstly, information can be directly imported from an external data source, see the dashed arrows. Here, a single interface component is required which is responsible for any fully automatic data transformation or translation needed. In contrast, an intermediate data model can be used if more complex transformations with manual intervention are required, see the solid lines. Here, data from external sources is imported into an intermediate persistent storage. After having performed any data preparation steps necessary the data is then loaded into the working memory. For both operations, data transformation procedures might be executed. Manual intervention by the developers can be required. Finally, information developed solely for the KBE system like, e.g., an electronic formulation of shipyard standards can be directly loaded into the system, see the dotted line. No transformation is needed.

An intermediate data model is applied in case of complex information of a quasi-static nature. For the automatic execution of the system an automatic import of changing data is required. Therefore, problem-specific data needs to be handled using the first approach.

All knowledge models are defined as ontologies using the *Web Ontology Language* OWL standard, see section 2.2.3. The main considerations for the selection of this language were as follows:

- Expressibility,
- Editing,
- Reasoning and Transformation,
- Tool and Community Support.

OWL supports all common modeling constructs used for data model design. Type hierarchies can be defined. Inheritance is supported. *Data-Type Properties* allow the assignment of specific values for certain aspects of a component. With *Object Properties* links between different object instances can be defined [128].

Powerful editing environments for the definition and management of class structures are available as shown in figure 6.5. For example, in contrast to relational databases or STEP databases, the direct editing of instance data, see the bottom window, is supported. This provides a single development framework for the definition of the structure of the knowledge model, see the left column. Details of, e.g. a stock plate type can be defined as shown by the center window. The definition of relations between object instances, i.e. a definition of a graph-oriented network, can be performed comfortably.

As OWL is based on first-order description logics, support for automatic inference is available. This allows for semi-automated consistency checks of the standards developed. Inconsistencies can be detected directly within the model. For instance data, constraints to be fulfilled can be defined and validated. For data transformation, the SPARQL query language can operate on existing knowledge bases. Transformations can be performed [129]. New data structures can be created [130].

With open-source libraries like *Jena* [131] for the programmatic access to the data or *Pellet* [132] for computer-based reasoning, the integration of OWL data models into applications is eased. These libraries are developed in Java, hence no language barrier is given for an integration into Drools as business rule engine.

The use of Java objects restricts the use of certain modeling capabilities defined as part of

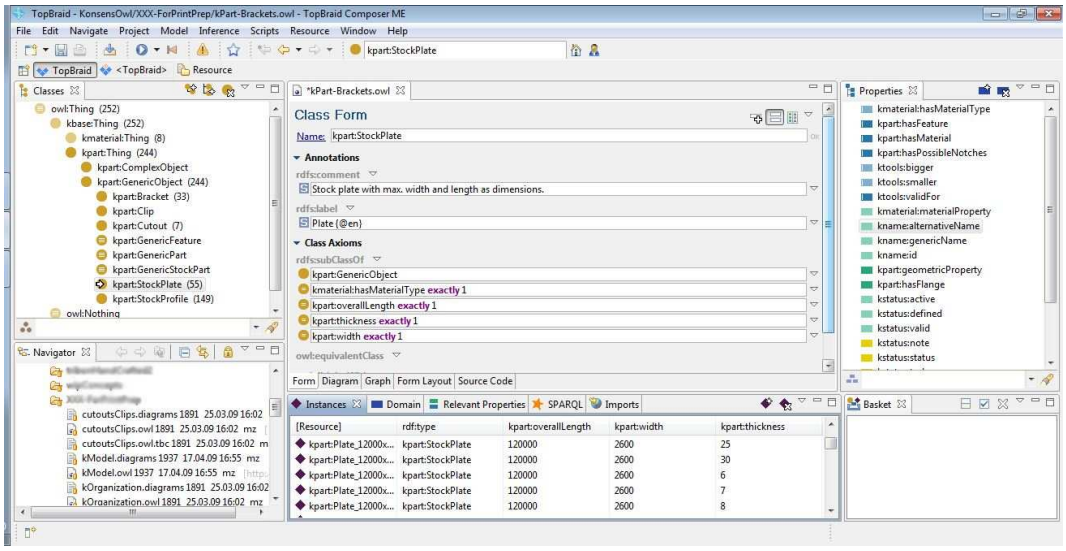


Figure 6.5.: Topbraid Composer as Editing Environment

the OWL standard. E.g., multiple inheritance is only supported via interface definitions. Transitive properties and inheritance of properties are handled differently. From a Java application, a native, object-oriented access to data stored in an OWL database is by default not possible. For this purpose, two different solutions have been developed, namely:

- direct bidirectional access via proxy objects,
- native Java objects stored in an intermediate persistent database.

Following the guidelines developed in [133], a generator for the automatic generation of a Java Proxy API has been developed. The data stored in ontologies can be accessed natively via Java classes, methods and attributes. Operations for data retrieval, search and editing are provided. More information about the technical details are given in the appendix, see chapter A. The classes generated act as proxy objects. These objects operate on a knowledge base which is defined by an ontology. The ontology can be file-based or stored in an appropriate database. Access to remote knowledge repositories is supported. The Jena framework is employed to provide programmatic access to the ontology. Calls to the methods of the proxy objects, i.e. calls to the Java API, are translated into the corresponding method calls from Jena. Any editing actions result in direct changes of the knowledge model. In case of constraints defined in the ontology - these are not directly implementable in Java - corresponding tests are implemented in the wrapper methods. E.g., if a bracket has a constraint which states that a maximum of two profiles can be connected to this bracket, a corresponding method named *addProfile()* aborts with an error, if two profiles are already defined for the bracket instance the method is called upon [134].

In the previous approach, persistence is provided by the ontology model. For large models, performance problems occur with the current implementation. As an alternative approach with drastically improved performance, an additional persistence layer has been added, also see figure 6.4. For quasi-static information, objects are imported from the ontology into an object-oriented database. As database engine, *DB4o* [135] is used. The generator mentioned above has been extended to provide the corresponding Java code. Automatically, a migration script is generated. With the help of the migration script all data contained in a given

ontology is automatically copied to an object oriented-database instance. This data can then be loaded directly into the working memory. The intermediate database and the underlying ontology are decoupled. If the ontology is changed, the import process needs to be executed again.

Modeling - Ontologies, Classes and Instances Where possible, a modular approach is chosen. Basic functionality or data structures are defined in separate knowledge models or ontologies, see figure 6.6. Different *Namespaces* are used. With these ontologies as building blocks, the reusability of the concepts developed is improved. A set of core and supplementary ontologies is defined.

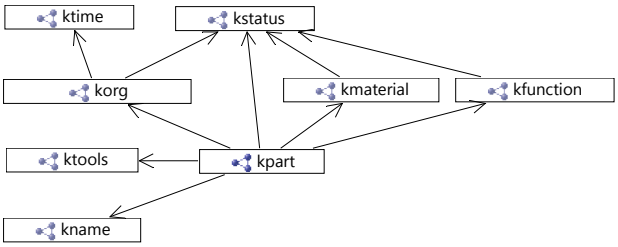


Figure 6.6.: Modular Ontology Structure

E.g., two ontologies named *kstatus* and *kname* provide relevant properties regarding the validity of a solution. The current status of a standard can be documented. It can be marked as approved or obsolete. These primitives are reused in more elaborate knowledge models. The *kdocumentation* module uses basic concepts about date formats etc. from *ktime* to represent organizational aspects. Functional primitives like functional capabilities of a certain solution are defined as part of the namespace *kfunction*. Finally, the main modules like *kpart*, the ontology representing actual information relating product data, is based on the supplementary concepts [136].

For modeling, classes are defined to denote the characteristics for each type of a standard object. The instances of a class then assign concrete values to these characteristics [137] [138]. As shown in figure 6.7, e.g., the shape of a certain bracket type may be defined by two lengths L, l and the plate thickness t. Instances of this type then define specific values for the three variables. E.g. the bracket *A200* uses a length of the connecting flange of $L = 200$.

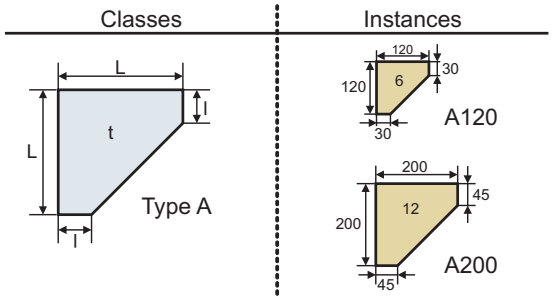


Figure 6.7.: Class Definition and Class Instances

Modeling - The Standards Database For a standards compliant design an electronic representation of the relevant design standards is needed. Using a syntax similar to UML, the main class hierarchy is shown in figure 6.8. Classes are drawn as boxes with the class name and namespace written on top. Properties follow as list. Here, blue icons denote object properties while green icons reference datatype properties. In square brackets, information about the number of entries required or allowed for a single property is shown. For the class *kpart:GenericPart*, e.g., each instance needs to have exactly one reference to a material definition of type *kmaterial:GenericMaterial* [139].

Using a top-down modeling approach, the class *kbase:Thing* defines an ID and an optional generic, descriptive name. These attributes are available to all objects in the knowledge model and can be used for reference. Everything related to the definition of standardized parts and features is defined below the *kpart:Thing* class. This class provides a logical partitioning of the class structure. *kpart:GenericObject* is the root class for stock parts and parts or features as introduced in section 4.5. Complex Parts are defined as subclasses of *kpart:ComplexObject*.

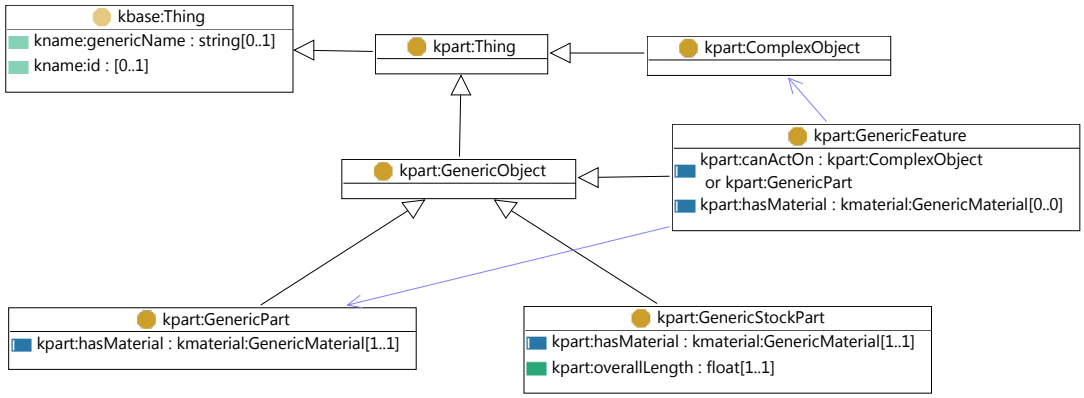


Figure 6.8.: Main Class Hierarchy for the Standards Data Model

As parts are materialistic objects, a reference to allowed materials is defined. This is formulated via the *hasMaterial* object property. For features, no material definition is allowed, see the [0..0] restriction. The domain of applicability of features is defined via the *canActOn* reference. Multiple references to parts or complex parts are allowed. E.g., for the definition of a cutout, plates and girders of sufficient height are added as permissible parts via this object property.

With *kpart:GenericObject* being the root class for the representation of singular entities like stock parts, parts and features, a typical data model can be defined as shown in figure 6.9 in extracts. In this model, some main particulars for the representation of bracket related knowledge are given.

For each standardized bracket instance, common shape related information like the thickness needs to be given, see the *kpart:thickness* property. Depending on the actual contour of the bracket, further shape related data is defined in subclasses of *kpart:GenericBracket*. The field of applicability with respect to suitable profile types is represented by the *ktools:validFor* property. A list of stock profiles available is given by subclasses of the *kpart:GenericStock Profile* class. In this class, common characteristics for all stock profile types and profiles dimensions are defined. These are material, height and length overall. Information about

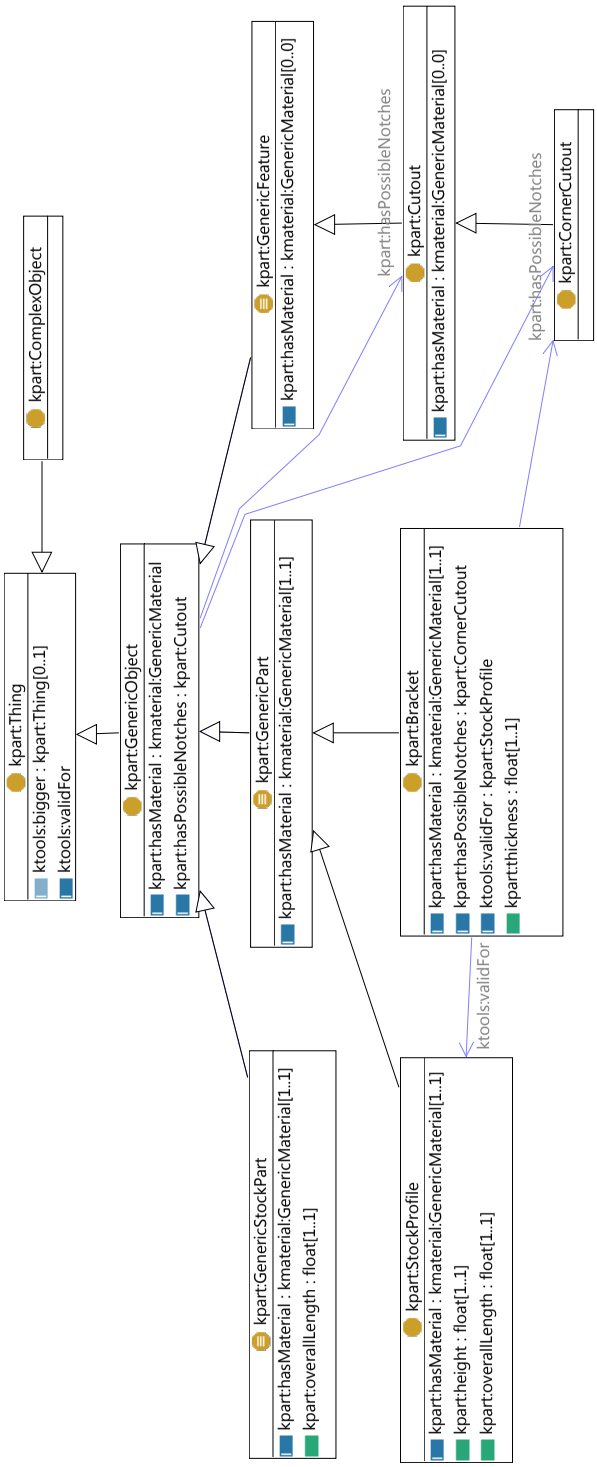


Figure 6.9.: Core Data Model for Stock Parts, Parts and Features

the specific shape and its dimensions of the cross section is defined in the child classes. Information about suitable cutouts in the bracket, e.g. for an eased welding operation, is given by *kpart:hasPossibleNotches*.

Following the approach formulated in STEP AP218, standardized solutions can be restricted to certain projects or vessel types. For this purpose, the property *kstatus:isValidFor* is defined as shown in figure 6.10. The rule-based system can thus use the type of vessel during the solution process. Also, certain standards can be restricted to a specified list of shipbuilding projects.

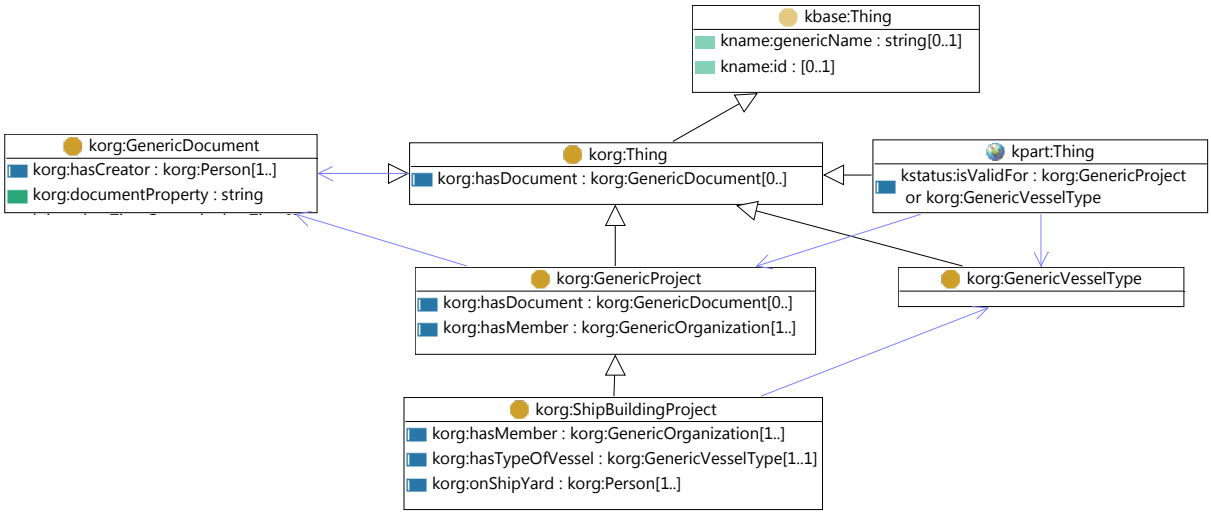


Figure 6.10.: Documentation and Project Specific Definitions

Also, for standardized solutions, additional documentation can be attached via the *korg:hasDocument* reference. For this purpose, extensions of the *korg:GenericDocument* class are defined to represent documents, images or external links. These are not shown in the figure for brevity's sake. Using aspect oriented programming techniques, any object within the knowledge model that is derived from the *korg:Thing* class like, e.g., *kpart:Thing* in figure 6.10, can thus reference documents. This information can be used by the operator to gain further insight into the problem at hand.

With this approach, information about the shipyards standards as shown in figure 6.4, page 62, can be represented electronically. An knowledge-based evaluation of the information base is feasible. More information about specific formulations used is given in the following chapters 7 and 8.

Modeling - Problem-Specific Models In this thesis, for the design data, i.e. for information provided by the product data model, custom problem-specific models are used. Different models are developed for each test case. Task-specific and hence simplified data models were chosen, as a full-featured data model like *STEP AP218* or the *Enterprise Reference Model* from *Atlantec Enterprise Solutions* [140] would lead to a significant increase of the complexity for the rule formulations used. A focus is placed on functional modeling, i.e. the functional relationships of the elements imported are of prime importance. With this approach, a condensed and problem-focused view on relevant attributes can be obtained [141] [142].

An important factor to be considered is the use and reuse of non-geometric information. E.g. information about room definitions and their designations can be required. As a result of a reasoning process, data about, e.g., permissible stress levels within the ice girder may be given. Specific Loads may be calculated. Often, this kind of information is not part of the PDM model imported nor does the PDM or CAD systems provide the capabilities to store this kind of information. Results can therefore not be completely transferred to the PDM.

In case of the CAD system of choice, *Tribon*, the data model used is geared towards the representation of shape, topology and manufacturing data. Any information regarding, e.g., room definitions and designations, strength and stress related information or data about functional attributes like bulkhead definitions are not represented. Results from other applications like FE applications or class tools are not integrable.

Using concepts of the knowledge model, for this purpose a simple data model for meta-information has been developed, see figure 6.11. The data models provided by the CAD system can thus be augmented. Each object of type *kmeta:Object* is stored under a unique identification number which is used for data retrieval in subsequent design activities. Here, for Tribon, the block, panel and part names are used to generate the IDs. Relevant facts can be stored as meta-information using one or multiple key-value-assignments via the *kmeta:hasValue* statement. For each assignment and attached to a *kmeta:Object* instance, the value to represent is stored in conjunction with a key identifying its purpose, see the *kmeta:value* and *kmeta:key* properties. Type safety can be improved by using an optional type identifier.

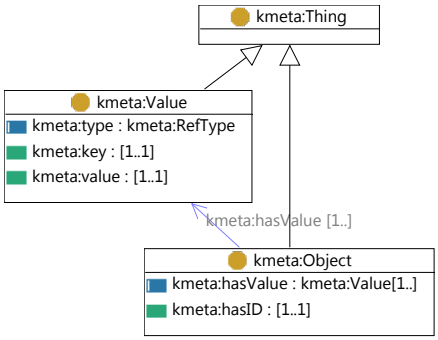


Figure 6.11.: Storage of Meta Information

6.3.2. Interoperability and Knowledge Import

As shown in figure 6.4, the import of existing product data into the working memory can either be performed directly or via an intermediate model. Different interfaces to the CAD system's data can be used. For Tribon, the design system selected, two different approaches can be taken [143]. These are:

- via export functions,
- by programmatic access.

To reuse product data, export functions as defined in a CAD system are applicable. In Tribon, for this purpose a series of data formats are available. For each format, an *XML*

Schema (XSD) is used to define the structure of the data. Attributes of elements as well as their relationship are specified. For the representation of steel structures, the *Tribon XML Hull Basic Design* format *TXHBD* can be used.

As this exchange format is developed with a focus on manufacturing and assembling tasks, the complete information present in the data model is not accessible. E.g., the topology is not available. The format is focused on the export of shape, manufacturing and material information only.

For programmatic access, the *Data Extraction* DEX Library can be used to access data from the product data model. Only read-access is provided. Also, the *Vitesse Framework* is a Python-based development environment that also allows access to information of the model. Information can be retrieved from, changed in or added to the model [97].

With an evolutionary approach adopted for the development, further understanding of the reasoning process has resulted in changing requirements regarding the knowledge model. This has been accommodated by an extension or change of the import solutions. Also, additional capabilities added to the rule engine Drools introduced new approaches to handle the import process. Therefore, multiple approaches were investigated and tested. For the test cases presented in the following chapters, three different solutions with individual strength and weaknesses have been explored, namely:

- the import from TXHBD data via an intermediate model,
- a direct import from TXHBD hull data,
- the native access to the product data model.

TXHBD Import with an Intermediate Model A first approach to data import is based on an intermediate OWL data model. The data is provided in an TXHBD-compliant XML file. The file is imported and converted to OWL. For this purpose, an import facility has been defined. A two-step approach has been adopted:

1. Generate OWL classes and properties.
2. Import the instance data.

Using the XSD schema definitions provided by TXHBD for the declaration of the data structures, corresponding classes and properties are defined as part of the knowledge model. Mapping relationships, i.e. the transformation of XML definitions to OWL definitions, are created. This task is performed once, see figure 6.4 on page 62. The class structure as well as the attributes defined can be changed manually. The data structures can be optimized for the reasoning process required.

For each design problem at hand, the knowledge model is populated from the corresponding XML file. According to the given mapping, corresponding class instances are created. Data is assigned to datatype properties of an object. Object properties, i.e. references between objects, are created.

TXHBD does not provide topological information. As this thesis is not focused on integration aspects, currently a manual operation is required. Topological relationships are added by the user, a process which is feasible only for simple problems. This includes, e.g., information about connectivity and intersections. If required, additional transformation or data preparation steps are executed. E.g., for each object a unique name is generated, which is

used for meta-data storage. Such transformations are defined within the ontology model. For the definition of these transformations *SPARQL* [144] is used, see [145] for typical examples of such constructs.

Direct Import from TXHBD With version 5 of Drools released early 2009, native import capabilities were introduced. *Smooks* [146] as transformation engine is directly integrated into the business rule management system. The working memory can thus be populated from the XML data source directly. Transformations can be defined. Here, based on translational mappings, the structure of the object model created in the working memory can differ from the structure of the XML data.

The intermediate model required by the previous solution is not needed. A more comfortable, direct approach to import data from TXHBD files is given. Yet, topological information is not present. Without an intermediate model, the manual addition of topological data is not possible. Therefore, the approach is only usable for cases where no information about topology is required or where this information can be obtained in a different way.

Native Access Native access to the product data model is probably the most efficient and powerful approach. Only with a native access, a bi-directional access can be realized. This approach has only seen limited testing as part of this thesis, though.

As part of the collaborative research project *QualiShip* [147], a wrapper on top of the DEX has been developed. *Data Access Objects* are defined which act as abstract interface to the product data model [148]. A rule execution may lead to updates of the knowledge model and might hence change the Tribon data model. Details of the *Tribon* database are not exposed.

An object-oriented view on the product model is provided. Attributes of the model can be accessed directly. The topology is available. For each access method, DEX queries are implemented which perform the corresponding extraction of the product data from the model of the CAD system. Helper Objects are present to facilitate search operations on the product model.

At runtime, relevant data is retrieved from the CAD system. For a given design problem, the corresponding data access objects are added to the working memory of the KBE system. If updates are performed on the data model, the corresponding information is updated in the CAD system. If the CAD system is not capable of representing a certain type of information, see above, the data is persisted in the external database for meta-data. On a following data access, the information of both data sources is merged and presented holistically.

7. Atomic Design Activities

Based on the theory of generic design activities as introduced in section 3.2, the design task as performed within the design process in naval architecture is given by a sequence of individual design activities where for each such activity a single, self-contained or *atomic* objective is given. In this chapter, with a focus on the application of problem solving methods in the domain of ship structural design, for each test case identified in section 5.5, the corresponding KBE algorithms are shown. Based on a requirements analysis, the design problems for the given test cases are defined such that the design problem is completely covered. For the definition of the design standards to be used as part of the test cases, an electronic knowledge base is developed. Data models are developed for the representation of the ship structure.

As the field of application as well as possible configurations are known for each design activity presented in this chapter, a clear and unambiguous recipe regarding the selection of suitable and valid solutions is given for all possible solutions. Therefore, the design problem can be classified as a simple configuration design task as introduced in section 2.3.3. The design activity hence operates in a fully defined context.

For the work presented, the development of approaches towards an adequate formulation of a knowledge-based definition of the solution process is of key concern. Therefore, in the results presented, not all design aspects and design criteria as relevant for the full determination of a design solution are presented. In contrast, a focus is placed on the presentation of applicable modeling techniques. Typical problems encountered as part of the rule formulation process are highlighted.

For the formulation of algorithms, the rule engine *Drools*, see chapter 6, is used. As language for the definition of the rules, algorithms are formulated using the so-called *Drools MVEL Dialect*. In MVEL, a rule needs to be defined according to the following structure:

```
1  rule "Name of the Rule"
2  attributes
3  ...
4  when
5    conditions (LHS)
6    ...
7  then
8    actions (RHS)
9    ...
10 end
```

For each rule, a custom name as well as rule-specific configuration attributes are defined as part of the rule header. As part of the *Left Hand Side* (LHS), the conditional parts of the rule are given. If all requirements defined here are fulfilled, the *Right Hand Side* (RHS), the so-called action is executed. For more information about the dialect see [149].

7.1. Bracket Design

In ship design, brackets are mostly used to create a fixed connection between two profiles profiles or to connect a profile to a plate structure. For these parts, load transfer and load distribution within the ship structure can be named as primary functions. In the following, a KBE solution is developed which is based on the evaluation of a given design standard. With the focus of the work being placed on the connection of profiles with a bracket, other types of bracket connections are not considered. For verification and development, the test case as introduced in section 5.5 on page 53 is used.

7.1.1. The Design Standard

On most shipyards, for common use cases established and standardized solutions are used for the design of brackets. As part of the work presented, the detailed design standards for a series of multiple vessels built at a major German shipyard were analyzed. For the development of KBE algorithms for bracket design, this catalog of standardized solutions as defined for medium-sized container vessels is considered in the following.

As part of the detailed design standard considered, a set of types of brackets for bracket design is given. For each type, the overall shape as well as governing dimensions are given. Where needed, material grades are considered. A list of instances is defined for a bracket type, such that the diversity of parts is restricted to a fixed, known set. For each entity within this set, the field of applicability is defined. A list of compatible, i.e. feasible and valid profile types and dimensions is given as shown in table 7.1.

		Profile				A, A36	Profile				A36
		Bracket				same	Bracket				A
Profile	**Equivalent**	**ID**	**l**	**t**	**f**	**a**	**ID**	**l**	**t**	**f**	**a**
HP120x8	FL120x(15-20)	KL120	200	8	-	3.5	KL140	200	8	-	3.5
		BCB120	200	8	-	4.0	BCB140	200	8	-	4.0
HP140x8	FL120x15	KL140	200	8	-	4.0	KL160	250	10	-	4.0
	FL150x10	BCB140	200	8	-	4.0	BCB160	250	10	-	4.5
...	...										
HP400x14	-	KLK400	575	14	120	6.0	KLK430	600	15	120	6.0
		-					-				
...	...										

Table 7.1.: Bracket Table for the Assignment of Profiles and Brackets

In total, a set of four bracket types is defined within the standard, see figure 7.1 for a graphical representation of these types. The types *BCB* and *KL* are unflanged brackets. Flanged bracket types are *BCBK* and *KLK*. In total, 82 different bracket variations are defined that cover bulb and flat bar profiles with a height ranging from 120 to 430 mm.

As part of the bracket table, an assignment of bulb profile and compatible bracket sizes is given. For each bracket, governing dimensions like leg length l, thickness t, flange width f and weld height a are defined. Depending on the material grade used for the profile, different

dimensions might be required for the bracket. E.g., for a bulb profile *HP140x8* of material grade *A36*, a bracket like *KL140* can be used if this bracket is also manufactured from *A36* grade material. In contrast, for normal steel, the larger bracket *KL160* needs to be used. For large profiles, only a single compatible bracket, namely the flanged bracket *KLK*, is defined. In contrast, for smaller bulb profiles and the corresponding equivalent flatbar profiles, two different bracket types are applicable, see the first two rows of table 7.1. E.g. for a flatbar profile *FL120x16* and an identical material grade configuration, the brackets *BCB120* and *KL120* are available for selection. Here, two different use cases and hence different fields of application are given, see figure 7.1. Brackets of type *KL* and *KLK* are used for the in-plane connection of a profile with other profiles or walls. In contrast, the bracket types *BCB* and *BCBK* are used to connect a profile to an orthogonally oriented, non-intersecting second profile.

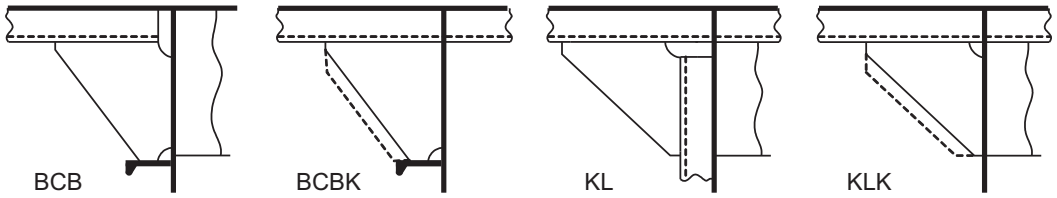


Figure 7.1.: Main Bracket Types of the Design Standard

As a result, for the selected use case, i. e. for a profile-profile bracket connection, this standard defines an unambiguous assignment of profile type and dimension to bracket dimension. I.e. for each profile size, there is at most one applicable bracket size for each bracket type. Multiple instances of differing bracket types may be applicable, though.

In addition to the assignment of bracket dimensions to profile sizes, general information, e.g. about required welding preparations or other manufacturing related aspects, is stated. As the standard is formulated, such that class rules are automatically adhered to for all feasible solutions, therefore, class regulations are automatically fulfilled if a standardized solution is used. Hence, direct calculations are avoided, if a standards compliant solution is applied.

Where required, more information about permissible use cases is given in text form. Hence, the field of applicability is restricted for specific bracket types and certain use cases. As example, flanged brackets are required in case of the free edge of the bracket being more than 50 times greater than the bracket thickness. While this is normally regarded by the bracket standards analyzed, for certain configurations and in case of a low thickness of the flange of the governing profile, unflanged profiles might not be suitable. Hence, flanged solutions might be applicable only, while the unflanged brackets with correct dimensions are not suitable.

7.1.2. The Test Case

The test case used is based on a typical design scenario from the domain of detailed ship structural design of commercial vessels. Derived from a design solution for a double bottom structure located midships, a test model as shown in built position, i.e. upside down, in figure 7.2 is used.

With the test case, the main characteristics regarding bracket design were to be evaluated. For this purpose, all major design criteria as relevant for bracket design are represented within the model. Therefore, the model is not oriented towards a real-life steel design but on the representation of as many different design situations defined in the design standard as possible. It is not a design ready for production.

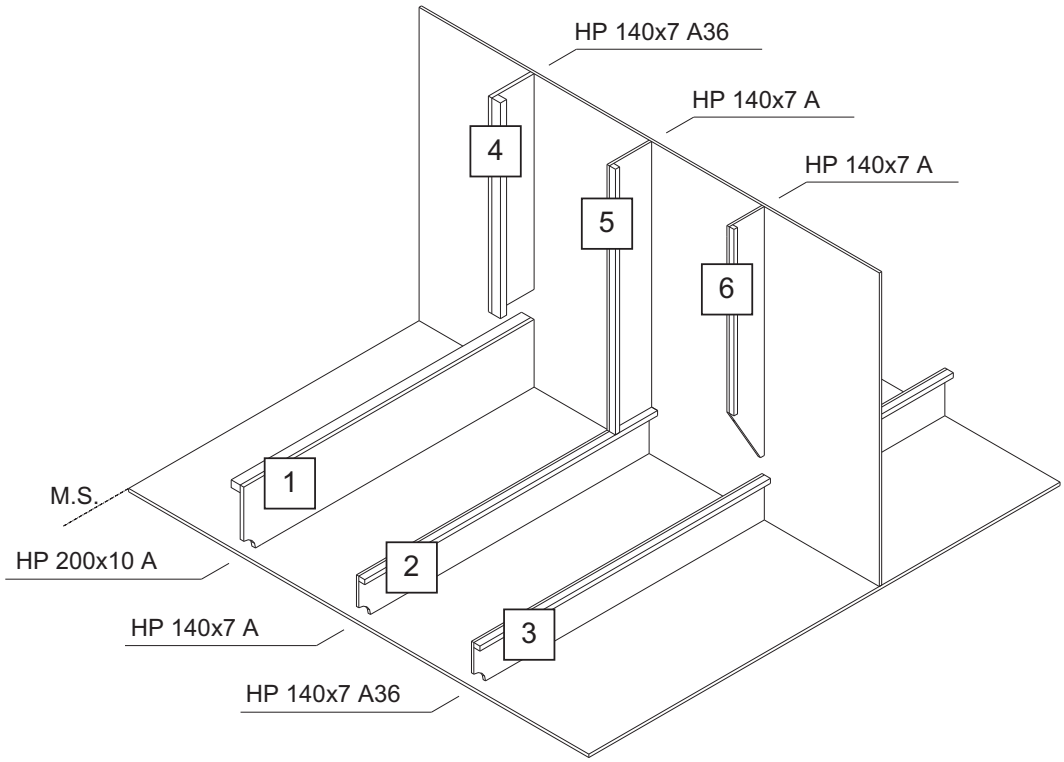


Figure 7.2.: Test Case for Bracket Design

As part of the test model, a single transverse wall segment which is connected to a deck is given. On the deck, three longitudinal bulb stiffeners are given. Compared to the outer two stiffeners, the inner stiffener (1) is of a different size and orientation. Here, an *HP 200x10 A* profile is presented. While this is not a usual design approach, such solutions are applied for improved weld operations or because of severe space restrictions due to piping etc. Due to the installation of equipment and machinery, there is only restricted space available next to the central stiffener (2) of type *HP140x7 A* next to the wall segment. For the outer stiffener (3) of type *HP 140x7* high-tensile steel of grade *A36*.

On the transverse wall, three vertical stiffeners of type *HP 140x7* are given which are aligned with the longitudinal profiles. The inner stiffener (4) is defined with an offset, i.e. there is no direct connection to the transverse bulb bar profile, and has a differing orientation. Also, the stiffener is made out of high-tensile steel of grade A36. Therefore, high strength requirements are given for the definition of the corresponding bracket. For the central stiffener (5), the opposite is the case. The third stiffener (6) acts as buckling stiffener and is hence defined using a snipped profile end-cut at the upper end. In this location, therefore, no bracket connection is needed.

7.1.3. The Data Model

For bracket design, two different yet interacting data models can be identified. On the one hand, the list of bracket types and their corresponding instances with all relevant information pertaining to the application of brackets needs to be represented electronically. On the other hand, for a given design problem, geometric information about the shape and topology etc. of all elements given as part of the design problem is needed.

For the definition of standardized bracket types a data model as shown in figure 7.4 is used. Based on the generic modeling foundations as introduced in section 6.3, the specific bracket types are defined as subclasses of *kpart:GenericObject*. Bracket specific definitions are defined in a common root class *kpart:GenericBracket*. Here, the thickness of the bracket plate, i.e. an attribute common to all given bracket types, is defined. A bracket can be defined either as flanged or as unflanged as represented by the *kpart:hasFlange* property. Where required, a list of allowed corner cutouts is defined with the object property *kpart:hasPossibleNotches*. These corner cutouts can be used to add, e.g., additional weld clearance. Also, the scope of validity can be defined for each bracket instance via the *ktools:validFor* reference. For this purpose, an additional constraint is placed on the property definition as given for the *kpart:GenericObject* class so that only profiles that are a member of the *kpart:StockProfile* class are accepted as partners.

As subclasses of *kpart:GenericBracket*, the governing dimensions of the specific bracket types are defined using datatype properties. For the unflanged symmetric bracket type *BCB* the length of the connected edge is defined. For the alternative unflanged type *KL*, an asymmetric bracket is given. Hence, two dimensioning edges are required as shown in figure 7.3. For the flanged solutions *BCBK* and *KLK*, the flange width is defined, too.

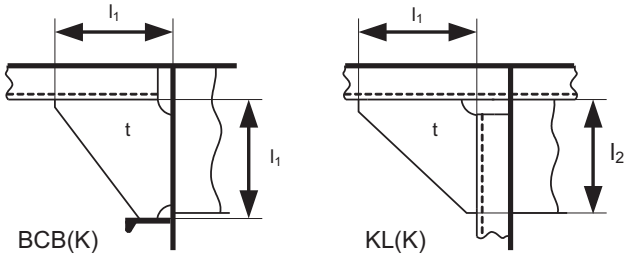


Figure 7.3.: Governing Dimensions for Brackets Types *BCB* and *KL*

For each bracket type, an ordering of bracket instances is required such that the search and retrieval of adequate bracket dimensions within the solution process is eased. For this purpose, for each bracket the property *ktools:bigger* and *ktools:smaller* are introduced. The former references the next larger bracket instance. Defined as inverse property, the opposite is described by the *ktools:smaller* attribute. Using the reasoning capabilities provided by OWL, both properties can be kept in sync automatically [150].

As design activities for bracket design are primarily a selection process of valid design elements as introduced in section 4.5, a so-called *State Attribute* has been introduced. With this attribute, the status of a standardized bracket instance is represented, i.e. its status within a certain selection process is given. For this purpose, the following values are accepted for the state attribute:

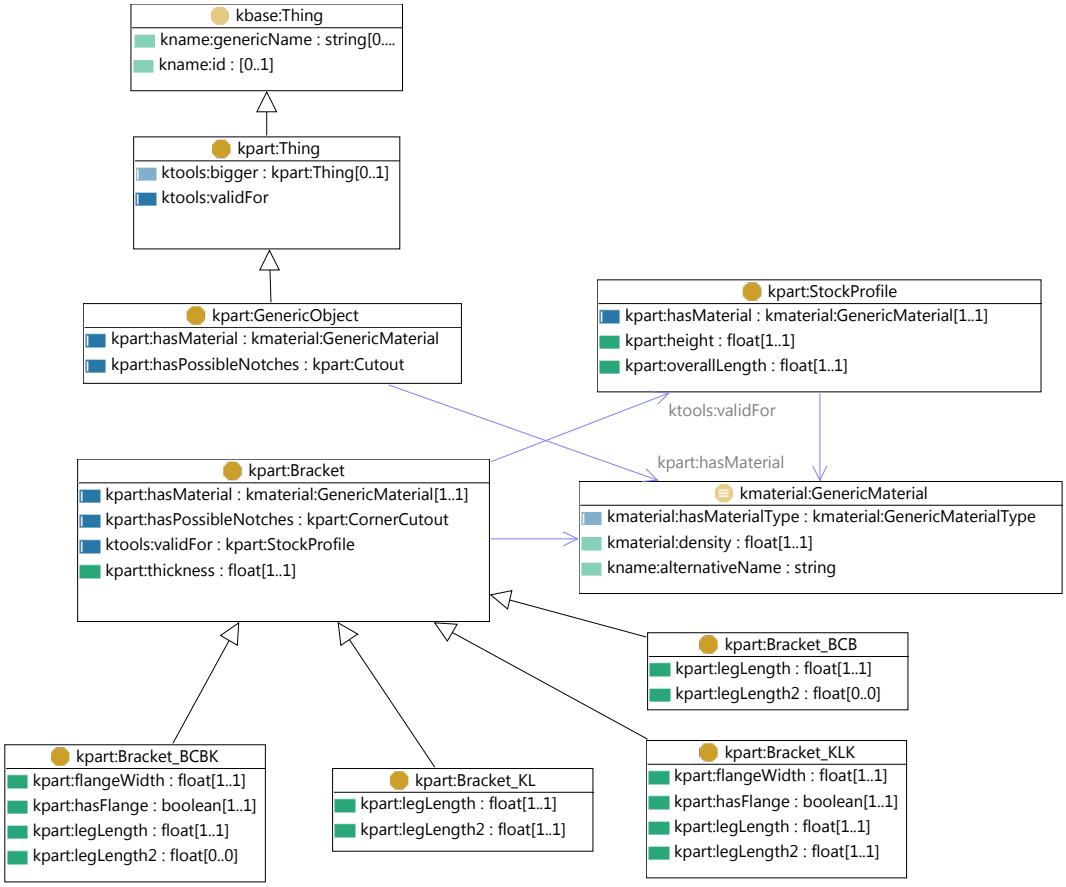


Figure 7.4.: Data Model for Standardized Brackets

- unknown
- plausible
- valid
- invalid

An *Unknown State* characterizes a bracket instance that has not been evaluated. It is not known whether the bracket is a feasible solution for the design problem or not. This is the default state before the selection process has started. A *Plausible State* defines an intermediate state. At an intermediate step of the reasoning process, the instance is not deemed to be invalid. It might be a suitable and valid solution. Or not. A *Valid* solution is given, if the reasoning process is finished and no excluding circumstances are given. In contrast, unsuitable or invalid brackets for the design problem given are marked as *Invalid*.

For the representation of geometry and topology of the ship structure, a simple OWL representation is used which is derived from the Tribon TXHBD standard for the representation of hull steel structures. As a result, a basic data model as shown in figure 7.5 is used. For each design problem, the data representation is imported from the corresponding TXHBD description as explained in section 6.3.2.

Within the data model, an object of type *kmodel:DesignElement* is defined for each element of the steel structure and the type of the element representation is given, see the

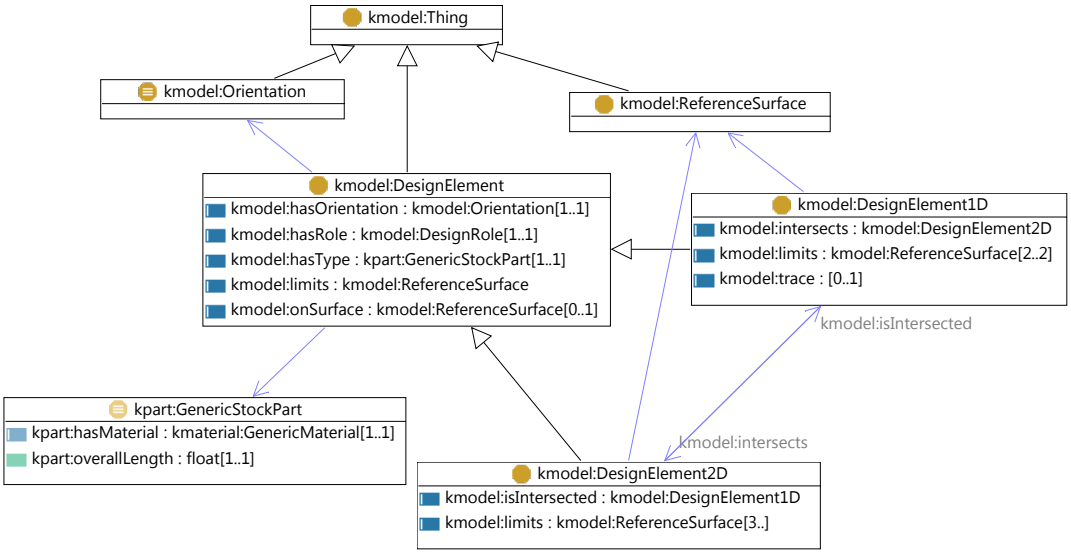


Figure 7.5.: Data Model for the Ship Structure

kmodel:hasType property. As only planar elements located on principal planes are currently supported by the data model, each element can be unambiguously defined by a single surface and the element boundary. Non-planar elements or parts with an orientation different from principal planes cannot be modeled. Therefore, the model developed is applicable to test cases located in the midship section of a vessel. For more complex structures in the aft or bow sections, a more elaborate model would be required.

For the definition of the location and orientation of elements, an approach using principal planes as reference is taken. For this purpose, an element is defined as located on a surface as defined by the *onSurface* attribute. Plates are defined as *DesignElement2D* elements which are limited by a set of planes that are defined by at least three *limits* references. Quasi-one-dimensional elements like profiles are defined using the subclass *DesignElement1D*. Here, the orientation of a profile is given by the *trace* principal plane definition. Start and end point of the profile are defined by the *limits* given. For the *limits* property, cardinality constraints are added as part of the subclassing process. Hence, the correct number of definitions can be validated electronically using reasoning capabilities as provided as part of the OWL standard.

Simple topological relationships are given by the *intersects* and *isIntersected* object properties, see the classes *DesignElement2D* and *DesignElement1D* respectively. As topology is not part of the TXHBD exchange format, topological information is currently added manually before rule execution.

In the structural model, a distinction between the standardized stock profile and the actual profile present in the ship structure is given. Both represent physical parts. Yet, the former defines the main characteristics of the stock material. Hence, a concept of a part is defined where common parameters for the profile of a certain type and size are defined. The latter, i.e. the actual profile, specifies the physical representation of a physical entity in the CAD model. The location and extents of the profile are given via the *DesignElement1D* definition and end-cut contours etc. are defined. The main characteristics of the stock part are included as reference.

As a design activity acts on a specific design problem that is part of a possibly complex

structural model, an option to select a set of elements like plates or profiles which take part in a specific design activity is needed. For this purpose, the concept of a so-called *DesignRole* has been introduced. With this attribute the role an element plays for the current design activity is defined. An *Active Role* denotes a selected element. In contrast, an *Inactive Role* defines an element that does not participate in the current activity.

7.1.4. Rule-Based Solution

For a rule-based solution, a set of rules is defined which acts on the electronic representation of the test case. Hence, the working memory of the rule engine environment is populated before the execution of the problem solving algorithms, i.e. the model representing the ship structure is loaded. The complete list of standardized bracket instances as well as other data like material definitions or profile types and dimensions are imported. Yet, only bracket instances are made available within the working memory for direct access. Referenced elements are not defined as primary members of the working memory and thus can only be accessed programmatically via their references from, e.g., a bracket instance definition.

As part of the design activity of bracket design, for the rule-based design of brackets five consecutive steps are taken, namely:

- add active profiles for bracket design as primary members of the working memory,
- mark invalid brackets based on all relevant criteria,
- construct a valid bracket solution,
- process the result,
- perform cleanup operations.

For the definition of brackets, two profiles are selected and are made available for reasoning within the working memory. A culling procedure is then implemented, so that brackets that are not valid for the profiles selected are removed from further evaluation. Having found one or multiple valid design solutions, the results are processed and, e.g., transferred to the CAD model. Finally, the working memory is reset to an initial state ready for the next run.

Activated Elements For the rule-based bracket definition, two profiles are manually selected by the engineer operating the system. The engineer thus defines the design context the algorithms operate in. For this purpose, the design role of two selected profiles is set to *DesignRole.Active*.

As bracket selection is primarily governed by strength concerns, the section modulus of the cross section of the connected profiles needs to be considered. More precisely, the profile with the smaller section modulus is of importance.

After the initial import, the profile type definitions of the selected profiles like *HP 140x8* are not yet available as primary members of the knowledge model but are only referenced from the selected profiles, see the corresponding knowledge model in figure 7.5. To determine characteristics of the profile with a smaller section modulus, the corresponding profile type definition needs to be given. Using the *MVEL Dialect* for rule definition in Drools this can be formulated as follows:

```
1 rule "Find and add stock profile type with min. section modulus"
```

```
2  agenda–group "Bracket Design"
3  no–loop true
4  when
5    profileA : KModelProfile(designRole == DesignRole.Active,
6                             secModA : profileType.sectionModulus)
7    profileB : KModelProfile(this != profileA,
8                             designRole == DesignRole.Active,
9                             profileType.sectionModulus <= secModA,
10                            profileBType : profileType)
11 then
12   insert(profileBType);
13 end
```

Within an agenda-group, see line 2, for each rule a unique name is defined in line 1. The *no-loop* statement in line 3 defines that the rule should only be executed once. Infinite execution loops due to iterative calls to the same rule or sequence of rules are thus avoided.

The conditional clause, the *when* part, defines the required conditions for the rule to be executed. In line 5 and 6, an arbitrary profile is selected from the geometric model present as part of the working memory. It is assigned as variable *profileA* with rule scope. The profile needs to have an active design role, i.e. it is selected for bracket placement. The section modulus for the profile type is retrieved and stored as rule variable *secModA*.

The second profile for bracket placement is retrieved in the second conditional statement, lines 7 - 10. The profile needs to be different from *profileA*, line 7, and also have an active design role, line 8. As a consequence, the rule engine consecutively tests all possible combinations of profile combinations yet accepts only two different, activated profiles for further processing. All other invalid combinations are ignored. In line 9, the section modulus of both profiles is compared. *ProfileB* needs to have a smaller section modulus than *profileA*. For further use, the profile type with the smaller profile is stored as variable, line 10.

In the action part of the rule, the *then* clause, a single action is performed. The type of the profile with a lower section modulus is added to the working memory. In consequent reasoning steps, the profile type definition can be thus used directly as part of the conditional clause.

Invalid Brackets As part of the approach chosen a culling mechanism is implemented. Bracket instances that are not valid for the given design problem are marked as invalid. They are thus removed from the domain of feasible solutions. If steel of grade A, i.e. with default tensile strength, is used for the profiles, this can be performed as follows:

```
1  rule "Mark invalid brackets (normal strength)"
2  agenda–group "Bracket Design"
3  no–loop true
4  when
5    profileA : KModelProfile(designRole == DesignRole.Active,
6                             material.grade == Material.GradeA)
7    profileB : KModelProfile(this != profileA,
8                             designRole == DesignRole.Active,
9                             material.grade == Material.GradeA)
10   profileType : KStockGenericProfile()
11   bracket: KStdGenericBracket(validFor not contains profileType)
12 then
13   bracket.setDesignState(DesignState.Invalid);
14   update(bracket);
15 end
```

Two non-identical profile instances of grade A steel are found in line 5 to 9. Again, both need to have an active role in the design process and need to be made out of material of

grade A. The stock profile added to the working memory in the previous rule is retrieved from the working memory, see line 10.

Finally, for each bracket instance given, it is tested whether the list of valid profile types and dimensions contains this profile type. If this is not the case, the bracket instance is not applicable to the current design problem. The design state is marked as invalid, see line 13. The *update* call in line 14 informs the rule engine that an object in the working memory has changed. The agenda and solution strategy can hence be updated accordingly.

In case of one of the two selected profiles being made out of high-tensile steel, the design standard requires a bracket of increased strength. For this purpose, the shipyard standard defines that for a given bracket type and for profiles of high-strength steel, the next larger size should be used. E.g., if a bracket *KL120* would be feasible for a bulb profile *HP120*, if high-strength steel is used for the profile the next larger bracket named *KL140* should be used. This is performed by the following rule:

```
1  rule "Mark invalid brackets (high strength)"
2  agenda-group "Bracket Design"
3  no-loop true
4  when
5    KModelProfile(designRole == DesignRole.Active,
6                  material.grade != Material.GradeA)
7    pType : KStockGenericProfile()
8    p : KModelProfile(designRole == DesignRole.Active,
9                      profileType == pType)
10   bracketSM : KStdGenericBracket(validFor contains pType)
11   bracket : KStdGenericBracket(smaller == bracketSM)
12 then
13   bracketSM.setDesignState(DesignState.Invalid);
14   update(bracketSM);
15   bracket.setDesignState(DesignState.Plausible);
16   update(bracket);
17 end
```

In line 5 and 6 it is tested if non-standard steel is used. If this is the case, the dimensioning profile type is retrieved from the working memory, line 7. The shape representation of the dimensioning profile *p*, as defined in the model, is added as variable with rule scope, see line 8. The bracket instances which are valid for the dimensioning profile type and dimension of material grade A are found with the statement given in line 10. The next larger bracket is retrieved, line 11.

If the conditions are fulfilled, the consequence is executed. According to the specified standard, the larger bracket is a feasible design solution. The bracket *bracketSM* is therefore marked as invalid. It is not a standards-compliant design solution. The next larger bracket *bracket* is a candidate. The design state is set to plausible.

If a bracket is designed in an area of restricted space, flanged solutions are preferable. The required installation height can hence be reduced. Unflanged solutions are not taken into consideration. The design state of such brackets is marked as invalid. As rule this can be expressed as follows:

```
1  rule "Restricted space requirement given"
2  agenda-group "Bracket Design"
3  no-loop true
4  when
5    profileA : KModelProfile(designRole == DesignRole.Active)
6    profileB : KModelProfile(this != profileA,
7                             designRole == DesignRole.Active)
8    SpaceManager.isRestricted(profileA, profileB)
9    bracket : KstdGenericBracket(flange == false)
10 then
```

```
11    bracket.setDesignState(DesignState.Invalid);
12    update(bracket);
13  end
```

As presented in the introduction to chapter 7, constraints regarding installation space require access to third-party knowledge not given in the model. Therefore, a manager application called *SpaceManager* is used. These components are applied wherever the access to third-party knowledge available from an external data source needs to be encapsulated and decoupled from the knowledge models that are present as part of the working memory. Hence, the SpaceManager component provides an interface to retrieve information about room designations, space constraints etc. Currently, a hand-crafted knowledge model is used as part of the component. In case of a missing space designation, the user is asked for manual data input as backup solution. In further development steps, this component can be transparently replaced by an automatic solution.

In the conditional part, both selected profiles are retrieved from the working memory, see line 4 and 5. The *SpaceManager* component is queried. If a situation of restricted space is given for the active profiles, line 7 returns *true*. Any instance of an unflanged bracket type can not be used, see line 9.

If no condition of restricted space is given, the opposite approach is used. This is shown in the following rule. Similar approaches could be used to adhere to additional constraints. E.g., if high loads are given, flanged solutions could be preferred. The implementation of such rules is not shown.

```
1  rule "Restricted space requirement not given"
2  agenda-group "Bracket Design"
3  no-loop true
4  when
5    profileA : KModelProfile(designRole == DesignRole.Active)
6    profileB : KModelProfile(this != profileA,
7                             designRole == DesignRole.Active)
8    not SpaceManager.isRestricted(profileA, profileB)
9    bracket : KstdGenericBracket(flange == true)
10 then
11   bracket.setDesignState(DesignState.Invalid);
12   update(bracket);
13 end
```

Construct Design Solution For bracket design, only hard constraints are given. I. e. for each condition a clear result is obtained. Fuzzy reasoning procedures are not to be expected. The result of the previous rules therefore determines a set of valid and feasible brackets, i.e. any bracket that has a design state different from *DesignRole.Invalid* is a valid solution. Furthermore, the standard is formulated explicitly. For each given problem and considering the constraints explained above, only one valid and feasible bracket instance is given. A solution has hence been found.

```
1  rule "Construct bracket solution"
2  agenda-group "Bracket Design"
3  salience -50
4  when
5    not KDesignBracketSolution()
6    bracket : KstdGenericBracket(designState != DesignState.Invalid)
7    profileType : KStockGenericProfile()
8    profileA : KModelProfile(designRole == DesignRole.Active,
9                             profileType == pType)
10   profileB : KModelProfile(this != profileA,
11                            designRole == DesignRole.Active)
```

```
12
13 then
14    bracketSolution = new KDesignBracketSolution();
15    bracketSolution.setConnectedA(profileA);
16    bracketSolution.setConnectedB(profileB);
17    bracketSolution.setBracketType(bracket);
18    insert(bracketSolution)
19 end
```

A computational representation of the solution is constructed. A rule of a lower salience, line 3, is defined. The reduced importance guarantees that this rule is executed only if all selection rules introduced above are processed. The condition in line 5 aborts rule execution, if a bracket solution is already present in the working memory. It is ensured that only a single valid solution is created. Alternatively, the *no-loop* statement presented in previous rules could be applied.

Using the active profiles, line 8 to 11, and the single bracket not marked as invalid, a bracket solution is defined. A so-called *KDesignBracketSolution Object* is created, see line 14. Using a port-based approach towards the representation of connectivity as developed by Liang [151], topological relationships are defined, see lines 15 and 16. Here, an implicit assumption regarding connectivity is made in so far as bracket connectivity is defined, see figure 7.6. A bracket is assumed to be connected to the larger selected profile via the *setConnectedA* statement and vice versa. Finally, the solution is inserted into the working memory and hence made available for further processing.

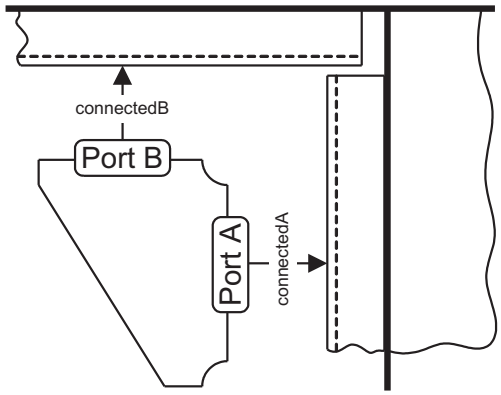


Figure 7.6.: A Simple Ports Model for the Topology of Bracket Placement

Depending on the height of the larger profile, a certain offset is used to install the bracket, see figure 7.7. The offset defines the distance of the outer plane of the bracket in relation to the flat side of the profile.

For profiles with a height larger than 160 mm, an offset of 10 mm is defined in the design standard. As rule, this is formulated as follows:

```
1 rule "Define installation offset"
2 agenda-group "Bracket Design"
3 when
4    bs : KDesignBracketSolution(profileA : connectedA)
5    profileA ( height > 160)
6 then
7    bs.setOffset(10);
8    update(bs);
9 end
```

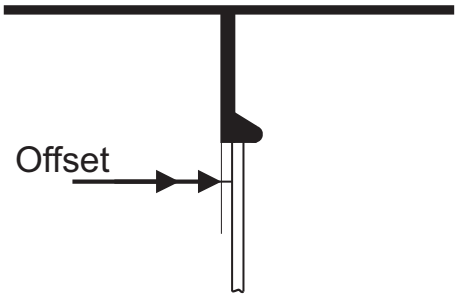


Figure 7.7.: Offset Definition for Bracket Placement

The largest profile, see the *connectedA* property, is retrieved from the design solution in line 4 and assigned to the local variable *profileA*. The comparison regarding profile height is performed in line 5. For profiles of a height larger than 160 mm, the offset is set to 10 mm in the action part of the rule. Similarly, the use of an offset of 8 mm for smaller profiles can be defined using a second rule formulation.

Process Results With the solution prepared the result needs to be processed. A lower salience guarantees that the rule is executed after any additional updates to the bracket solution object have been performed. Iterative calls to this rule are prevented by the *no-loop* statement.

```
1 rule "Process results"
2 agenda-group "Bracket Design"
3 salience -75
4 no-loop true
5 when
6   bs: KDesignBracketSolution()
7 then
8   ... (process results)
9 end
```

As presented in this thesis, the reuse of results within a CAD or PDM model is not considered, see section 6.1.1. The results are only shown to the user; an interaction of CAD model and rule system does not exist.

Cleanup If the working memory is to be reused for subsequent reasonings, the initial state needs to be restored. Temporary variables and the design solution objects need to be removed from the working memory. For all objects, the design state and design role need to be set to a default state. As examples, the rule formulation for the removal of the temporary object denoting the profile type is shown below. The design state of bracket instances is reset as presented in the second rule.

```
1 rule "Cleanup profiles"
2 agenda-group "Bracket Design"
3 salience -100
4 no-loop true
5 when
6   profile : KStockGenericProfile()
7 then
8   retract(profile)
9 end
```

```
1  rule "Remove design state"
2  agenda-group "Bracket Design"
3  salience -100
4  when
5    part : KStdGenericPart(designState != DesignState.Unknown)
6  then
7    part.setDesignState(DesignState.Unknown);
8    update(part);
9  end
```

7.1.5. Design Results

For the given test case, see figure 7.2, different combinations of activated elements are feasible. Some produce valid results while for other configurations the design activity of bracket design aborts with an error. As a result, an engineer operating the design algorithm and selecting each feasible combination of profiles successively, followed by the execution of the rule-based design activity, can reach a valid and complete result for bracket design. For the test case, therefore, this leads to a result as shown in figure 7.8.

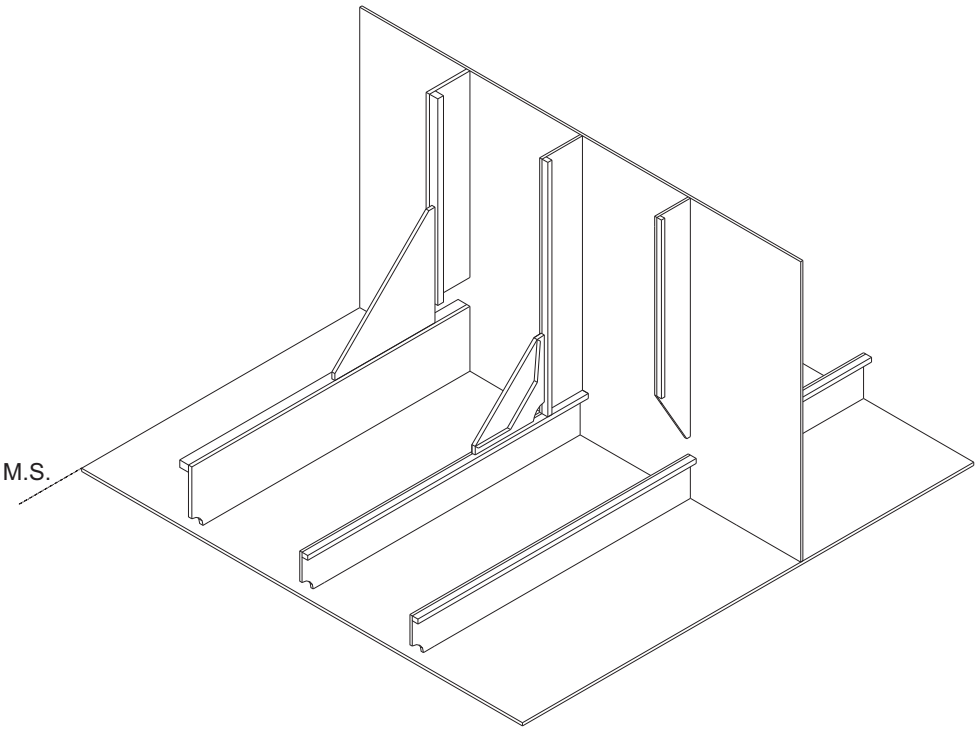


Figure 7.8.: Design Results for the Bracket Design Activity

In total, two brackets are defined for the inner two pairs of stiffeners. For the outer pair, the vertical stiffener acts as buckling stiffener only. Hence, no bracket is required; the algorithm aborts. For the pair of stiffeners located towards midship, an unflanged bracket of type *KL 160 A* is chosen, i.e. the high tensile steel used for this vertical stiffener and the hence required higher dimension of the bracket are taken into account. As the two connected profiles have a different orientation, the installation offset is defined such that welding as well as load transfer is given.

For the middle pair of stiffeners, each of type *HP 140x7 A*, a flanged solution has been chosen to accommodate the restricted space constraint given, see section 7.1.2. A bracket of type *KLK 140 A* is given. The flange is oriented outwards as stated within the design standard. The offset for the installation of the bracket is set to 8 mm.

7.2. Notch Selection and Definition

As presented in section 5.5.2, notches are placed on edges and in corners of plate-like structures, girders, profiles and sometimes of brackets. Often, these notches are defined to improve weldability, to avoid high corner stresses or as water culvert. Based on the given design standard, the knowledge-based automatic selection, design and placement of corner cutouts is presented in the following. Here notches that are located within an edge of a plate-like structure are not considered.

7.2.1. The Design Standard

For notches, a concise set of standards is given within the detailed design standard. The notch types and their instances, i.e. the possible dimensions for a certain notch type, are clearly defined. Different use cases are identified in text form where relevant aspects for the selection of applicable notch types and shapes are described for each use case.

For notches placed on corners, four different notch types labeled A to D are given as shown in figure 7.9. In total, governing dimensions are defined for 122 different notch configurations. For notches of type A, only a single instance is defined, i.e. the size and contour is fully defined if the required type is known. For notch types B to D, the radius of the notch is dependent on the thickness of the plate the notch is placed in. Also, for type D the width of the notch is defined as function of the distance between weld and adjacent vertical wall.

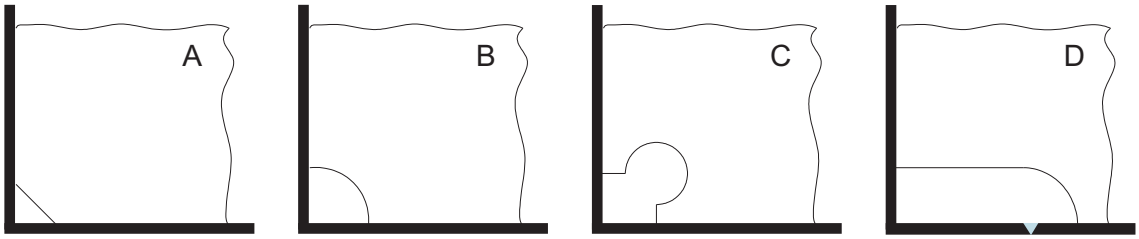


Figure 7.9.: Notch Types for Corner Cutouts

From a functional perspective, four different use cases for the application of notches can be identified, namely:

- watertight corner notch,
- notch as culvert,
- notch as stop-weld,
- space for continuous welding.

A watertight solution as represented by notch type *A* is used, if the plate the notch is placed on, needs to provide a fluid-tight barrier. Here, the contour of the notch is designed such that sufficient clearance for the fillet weld between the adjacent plates is provided.

Notch type *B* is used for non-watertight plate structures. For situations of high dynamic loads, type *C* should be used instead. In both cases, a water culvert is provided. In addition, this notch type is applied for the following scenarios:

- stop-weld
- weld clearance

If one of the plates adjacent to the plate with the notch represents a tank wall, a stop-weld is required, so that fluid leakage between the fillet welds is prevented. If the weld between bottom and side plate is applied at a later stage in the assembly process, sufficient clearance for a continuous welding operation needs to be provided. This can be done using notch types *B* and *C*. In case of an existing weld in the bottom plate nearby, type *D* is used as shown in figure 7.9 right.

If a watertight design is required and a weld on the bottom plate is given, the standard solutions are not applicable for the design scenario. No feasible and valid design can be found. Either a design error introduced by a previous design activity is given or - according to the design standard used as reference - a non-standard solution is required. In both cases, no automatic solution can be obtained. The algorithm hence is designed to abort and the user is informed about the problem encountered.

7.2.2. An Extended Test Case

For the verification of the rule-based solution, the test case presented in section 7.1.2 is extended, see figure 7.10. In addition to the transverse wall and deck structure with longitudinal and vertical stiffeners, a transverse girder as well as two longitudinal walls are defined. Here, for the design of corner cutouts both transverse structural members, i.e. the transverse wall and the girder, are regarded.

The midship part of the transverse wall (1) shown on the left is defined as tank wall where the tank is located. Any notch located in this plate therefore needs to be fluid-tight. The transverse girder (2) is located outside the tank. Here, no requirements regarding fluid-tightness are given. As this girder is designed for minimum height, it is assumed that a situation of high dynamic stresses is present towards the shell, i.e. on the right end of the girder. These need to be considered during the selection of valid notch designs. In the horizontal deck structure next to the inner wall, a longitudinal weld seam (3) is defined. The weld is located at a distance of approximately 100 mm from the inner longitudinal wall (4) and runs over the aft part of the considered deck structure. On the outer side of the right longitudinal wall (5), a tank is present.

7.2.3. The Data Model

For the definition of standards, a data model similar to the one used for bracket design shown in figure 7.4 is used. For the sake of brevity, this model is not shown in detail. As part of the model, the relevant notch types and their respective instances with governing dimensions are

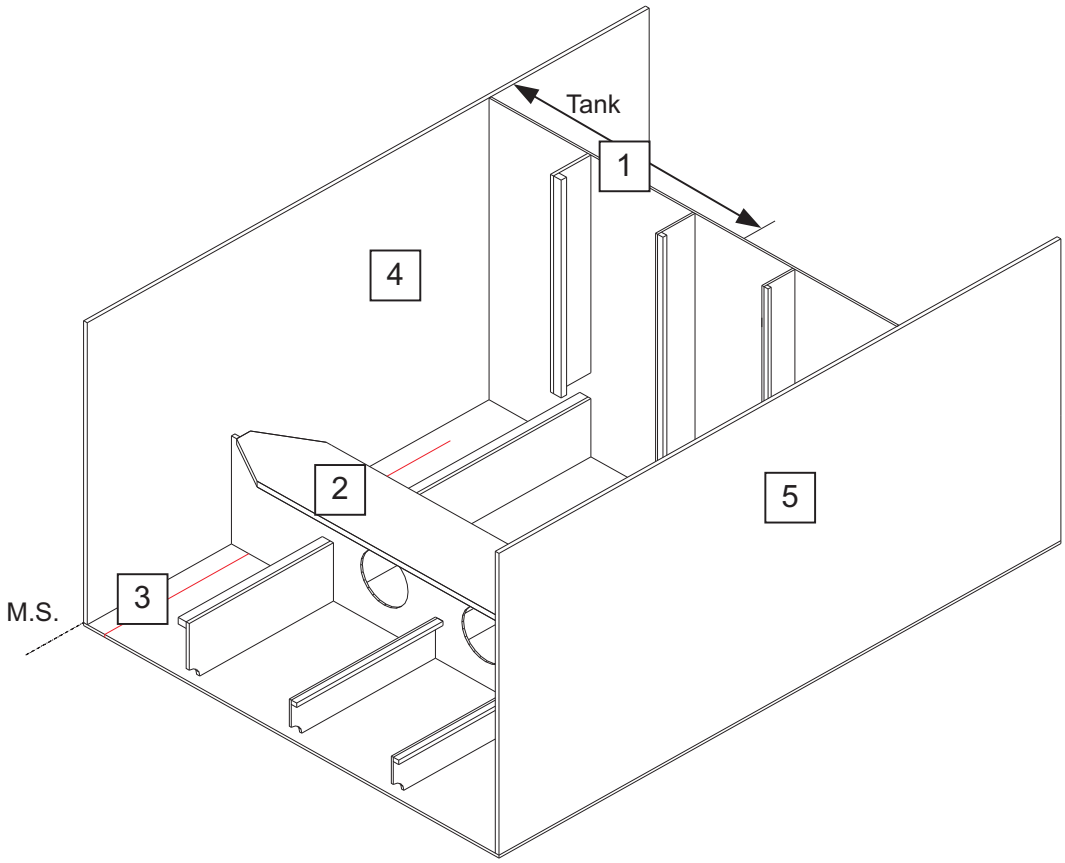


Figure 7.10.: Test Case for Notch Design

defined. For each notch representation, the corresponding field of applicability is given. For this purpose, a *validFor* object property is used to reference the characteristics of suitable stock plates the notch can be used for. Based on, e.g., the thickness of the plate, the selection of applicable notch types or notch dimensions is thus made feasible as shown in the following rule formulations.

For each notch type and the corresponding instances, so-called *Functional Attributes* are given where these attributes are used to define functional roles a notch can fulfill. If a certain function is required in a design problem, yet the relevant property is not set for a notch instance, this notch is hence not a solution candidate for the design problem.

For the definition of functional roles, a hierarchy of sets of keywords has been defined in an ontology named *kfunction*. A hierarchical classification is used, i.e. a taxonomy is given. E..g, strength-related information regarding functionality is defined as sub-elements of the main functional category *kfunction:GenericStrength*. Aspects regarding manufacturing are given as part of the group *kfunction:GenericManufacturing*.

The ship structure is imported from a TXHBD description. The information provided is converted to an OWL representation. As the TXHBD description does not provide suitable constructs, topological information is entered manually. Functional aspects pertaining to the ship structure are currently not stored as part of the PDM imported from the CAD system. The data is added before the KBE algorithms are executed. For reuse as part of the knowledge representation, this information is stored externally as meta-information as

shown in section 6.3.1.

To locate welds and to calculate distances, e.g., from the location of a corner notch, geometric dimensions are of importance. Geometric algorithms are therefore required which can operate on the topology and shape of the ship steel structure. For this purpose, a geometric kernel could be applied. Such a kernel is a fundamental and complex part of any CAD system. Also, an interface to access the geometric functionality of the system could provide the needed functionality, see section 6.3.2. Due to the complexity involved, a so-called *GeometryManager* component has been introduced for encapsulation, abstraction and simplification of geometric operations. This component offers solutions for the scenarios defined by the test case only. More advanced geometric operations are currently not supported.

7.2.4. Rule-Based Solution

In the previous example, the working memory was populated by all standardized bracket instances as defined by the design standards. During execution, rules were used to remove invalid brackets, i.e. brackets that do not fulfill the requirements and constraints. The approach can hence be characterized by a subsequent culling process.

For the design activity of notch design, an alternative approach is applied where an intermediate persistent, object-oriented database as introduced in section 6.3.1 is used. With this approach, more elegant and powerful formulations of algorithms are feasible. Yet, due to the intermediate database, additional requirements regarding the applied software components are given. Therefore, these formulations are not usable in certain cases where such intermediate knowledge models cannot be applied.

As multiple reasoning steps are performed within the KBE process applied towards notch design, rule partitioning, i.e. the logical separation of rule sets with different scope, is introduced. For this purpose, multiple agenda groups are defined to realize a clean separation of the different solution phases.

The object-oriented database *Db4o* [135] which is used in this test case, provides multiple mechanisms for information retrieval. In addition to a direct retrieval by name or identifier, various query methods are supported. Here, the so-called *Query by Example* mechanism, i.e. a pattern-matching approach, is applied. Based on a so-called prototype object labeled *PO*, all objects in the database are retrieved that are of the correct class or subclass. If attributes are defined for the prototype object, the query only returns objects that have a corresponding signature. Matching instances in the database have values identical to the values of the attributes that are set as part of the PO. E.g., a prototype object defined as:

```
1 po : KStdGenericNotch
2          hasFunction contains Function.Watertight;
3          highStress = false;
```

returns all notch instances, where *hasFunction* lists *Function.Watertight* as functional property and that is not suitable for application in high stress situations.

The main workflow as implemented for notch design is shown in figure 7.11. Here, four different stages as used for an automatic design of notches are shown. Shown in white, direct interactions with the database model as provided by a *Db4o Instance* occur. E.g., in the first stage, initially, the ship structural model is loaded into the working memory. In stages shown in gray, rule execution is performed by the *Drools Rule Engine*.

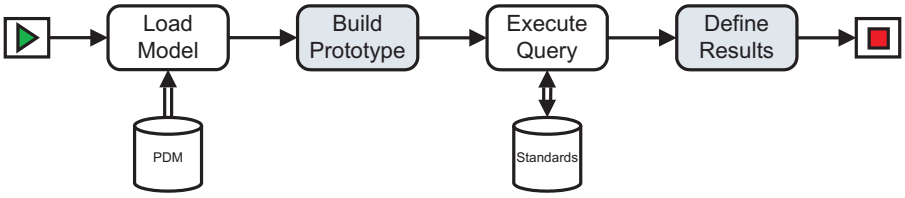


Figure 7.11.: Design Workflow for Notch Design

During the first stage of rule execution, mandatory constraints as formulated by the design standard are evaluated. A prototype object is defined. The characteristics defined by the requirements are added to the prototype object. A query is performed using *Query by Example* [152], see the third solution stage. Plausible solutions are loaded into the working memory. In the last step, further evaluations of all plausible candidate solutions obtained are performed. A design result is configured.

7.2.4.1. Model Import

Within the first step of the design activity, the KBE system needs to be configured. In contrast to the test case of bracket design, here, a twofold approach is taken. On the one hand, the design problem, i.e. all relevant information regarding the steel structure etc., is read from a TXHBD steel description and is hence made available for direct evaluation by all rules. On the other hand, all relevant design standards are stored within an intermediate, Db40-based database as presented in section 6.3.1. These are not directly imported into the rule engine before rule execution.

In this way, the design problem can be evaluated using a rule-based approach. Yet, the working memory is not cluttered with additional information not needed during this phase. In a later step, see the explanations in the following section, only possibly suitable standards are then loaded into the system for further processing.

7.2.4.2. Query Preparation and Population of Working Memory

After model import, no information regarding suitable notch types and instances is given as part of the working memory of the rule engine. Therefore, for further processing, suitable solution candidates need to be loaded into the rule engine and are hence made available for further evaluation. To load these objects, a *Query by Example* method is used.

For an example-based query to be executed, a prototype object is needed. As part of the design scenario presented, this is created with the following rule.

```
1  rule "Create prototype object"
2  agenda-group "Corner Notch - Prototype Preparation"
3  when
4    not KStdGenericNotch(designRole == DesignRole.QueryPrototype)
5  then
6    KStdGenericNotch notch = new KStdGenericNotch();
7    notch.setDesignRole(DesignRole.QueryPrototype);
8    notch.addAllowedLocation(Location.OnCorner);
9    insert(notch);
10 end
```

As extension to the design roles introduced in section 7.1, a design role designated *QueryPrototype* is introduced. Any object marked as prototypical is used only for queries. The object is therefore excluded from the actual design process.

This rule fires only if no *KStdGenericNotch* object with a prototype design role is present in the working memory. Multiple execution of the rule is therefore prevented by the statement given on line 4. In the action part of the rule, a new notch object is created, line 6, and inserted into the working memory, see line 9. The design role is set to *DesignRole.QueryPrototype*.

The test case is focused on the definition of notches placed on corners of plate structures. Other use cases are not considered. A constraint regarding the placement of the notch is added. Modeled as a list, the property *allowedLocation* can define one or multiple feasible locations the notch can be placed in. With the statement on line 8, only cutouts that list *Location.OnCorner* as one of possibly multiple allowed locations are regarded.

For the plate the corner notch is placed on two distinct functional constraints can be given that require special consideration during notch design. On the one hand, the plate may provide a watertight compartmentation. A rule formulation as shown in the following can be used.

```
1  rule "Located on watertight plate"
2  agenda-group "Corner Notch - Prototype Preparation"
3  no-loop true
4  when
5    notch : KStdGenericNotch(designRole == DesignRole.QueryPrototype)
6    plate : KModelGenericPlate(designState == DesignState.Active,
7                               functions contains Function.Watertight)
8  then
9    notch.addFunction(Function.Watertight);
10   update(notch);
11 end
```

The prototype notch object is retrieved in line 5. Therefore, this rule is only activated if such an object exists, i.e. after the previous rule has been called. A *no-loop* statement is given to prevent multiple calls to the same rule. The plate the corner cutout is placed on, is retrieved in line 6. It is ensured that it has an active design role, i.e. the plate needs to be selected. The plate needs to be watertight, see line 7.

If the constraints are met, a requirement for a watertight solution is added to the prototype definition of the notch. As functional constraint, any plausible solution needs to have a function of *Function.Watertight*. Also, high dynamic loads may be present. The notch may act as water culvert etc. Similar formulations can be applied for this purpose.

On the other hand, a stop-weld may be needed. This is the case if a selected edge of the plate on which the cutout is defined connects to a tank wall.

```
1  rule "Connected to watertight plate"
2  agenda-group "Corner Notch - Prototype Preparation"
3  when
4    notch : KStdGenericNotch(designRole == DesignRole.QueryPrototype)
5    plate : KModelGenericPlate(designState == DesignState.Active,
6                               edge from edges)
7    edge(designState == DesignState.Active,
8         plateConnected : neighbors != plate)
9    plateConnected(functions contains Function.Watertight)
10 then
11   notch.addFunction(Function.StopWeld);
12   update(notch);
13 end
```

In line 5 and 6, the selected plate is retrieved from the working memory and assigned to the local variable *plate* with rule scope. Using the topological information given in the model, one edge is retrieved from the list of edges of the plate. The edge is stored as local variable *edge*. The statement in line 7 f. requires that the retrieved edge needs to have an active design state. The edge is selected. The neighboring plate which is not the active and selected plate is chosen. This is stored as *plateConnected*. It is tested whether the neighboring plate *plateConnected* has a watertight requirement, line 10.

Multiple edges are defined for a plate. During execution, the rule engine iterates over all combinations in line 6 until the requirement regarding the edge as defined in line 7 is fulfilled. This approach allows to successively restrict the feasible domain. If all conditions are met, the neighboring plate is watertight. A stop-weld definition is hence required and the corresponding functional requirement is set on the prototype object. The working memory is updated accordingly.

Nearby welds need to be considered. Plates connected to the activated edges are of importance. The distance from the welds on one of these plates to the corner needs to be determined. A distance of less than 150 mm leads to a different design solution, see the discussion of the design standard in section 7.2.1. In this case, a notch of notch type D should be used.

```
1  rule "With weld nearby"
2  agenda-group "Corner Notch - Prototype Preparation"
3  when
4    notch : KStdGenericNotch(designRole == DesignRole.QueryPrototype)
5    plate : KModelGenericPlate(designState == DesignState.Active,
6                               edge1 from edges,
7                               edge2 from edges)
8    edge1(designState == DesignState.Active)
9    edge2(designState == DesignState.Active,
10         this != edge1)
11   GeometryMgr.weldCrossing(plate, edge1, edge2, 150)
12 then
13   notch.addFunction(Function.WeldCrossing);
14   update(notch);
15 end
```

The plate as well as two edges are retrieved from the working memory, see line 5 ff. Both edges need to be selected. Different edges are required. As the geometric interpretation of the topological model and the geometric data is not solved within the scope of this thesis, a manager component is introduced. For commercial deployment a more robust solution could query the PDM model of the CAD system.

The function call *weldCrossing* returns true, if there is a weld on one of the plates connected to the edges within a distance lower than the limit given. If this is the case, the corresponding requirement is set for the prototype corner cutout object.

As a result, a fully configured prototype object for the retrieval of plausible notch designs from the standards database is given. With this prototype, all hard constraints regarding notch design and under consideration of the design context of the current design problem are fulfilled as defined by the configuration rules.

7.2.4.3. Query Execution

The actions shown in the previous rules configured the prototype object. Once all hard constraints are taken into account, the prototype object is used to query the existing standards

database for suitable objects. A list of plausible results is returned by the *Query By Example* mechanism provided by the database *Db4o* and these results are added to the working memory for further evaluation.

```
1  rule "Retrieve candidates"
2  agenda-group "Corner Notch - Prototype Preparation"
3  salience -50
4  when
5    notch : KStdGenericNotch(designRole == DesignRole.QueryPrototype)
6  then
7    queryAndImport(notch)
8  end
```

For this purpose a rule of a lower level of importance is defined. The configuration rules for the protoype object thus have precedence over this rule. In line 5, the prototype object is retrieved. The action part consists of a call to a global function defined as part of the rule base. This eases maintenance and development as the same functionality can be used for different design problems and in multiple rules. The prototype object is given as argument.

```
1  function queryAndImport(Object obj) {
2    List <obj.class> results = db.query(obj);
3    for (result : results) {
4      result.setDesignState(DesignState.Plausible);
5      insert(result);
6  }
```

The global function queries the given Db4o database instance. For this purpose, the prototype object *obj* is passed as argument. The results are stored in a list. The list may contain $0 \ldots n$ elements. For type safety, *Java Generics* are used [153]. As all hard constraints are explicitly fulfilled by the prototype object, for each element the design state of the retrieved notch instance is set to *DesignState.Plausible*. One or multiple candidate solutions for the design problem have been found that are now inserted into the working memory.

If no results are found, the design scenario is not standards-compliant. The rule engine aborts. A warning is shown. If any results are found, the first reasoning step is complete. Further evaluation of such custom configurations needs to be performed.

Within this test case, for ease of maintenance, the so-called rule partitioning as introduced in section 6.2 is used and separate agenda groups are given hence separating the different solution steps.

```
1  rule "Agenda group transition"
2  agenda-group "Corner Notch - Prototype Preparation"
3  when
4    KStdGenericNotch(designRole != DesignRole.QueryPrototype)
5    notch : KStdGenericNotch(designRole == DesignRole.QueryPrototype)
6  then
7    retract(notch);
8    drools.setFocus("Corner Notch - Results");
9  end
```

Line 4 tests for the existence of results. The prototype object is excluded from the examination. In line 5, the prototype is retrieved and assigned as local variable *notch*. For the handling of additional configuration aspects it is no longer needed and is retracted from the working memory, see line 7. Finally, on line 8, the agenda group which defines the second solution step is activated. A transition to the next reasoning step defined as separate agenda group is performed. With the second agenda group selected, all rules defined as part of the first agenda group *Prototype Generation* are made irrelevant.

7.2.4.4. Further Criteria and Result Configuration

With the results of the query imported into the working memory, a list of plausible design solutions is given. Optional requirements could be of importance. As such requirements are not stated explicitly in the analyzed design standard, they are not regarded as part of this test case. Yet, an optional approach is presented as part of the use case for collar plates and cutouts, see section 7.3.

Depending on the configuration of the prototype object used within the previous solution stages, further criteria may exist for the final design solution. As an example, here, the configuration of the notch radius for notches of notch type C is shown. Following the design standard and using the thickness of the plate the notch is placed on, as input, the notch radius is calculated as follows:

$$R = 2 \cdot t_{Plate} \quad with \quad R \geq 35mm \tag{7.1}$$

Two different cases need to be distinguished. For a plate thickness of less than 17.5 mm, the corner radius is set to 35 mm. For thicker plates, the radius is calculated as function of the plate thickness. Using a rule-based formulation, this second case can be expressed as follows:

```
1  rule "Radius for high stress notches"
2  agenda-group "Corner Notch - Results"
3  no-loop true
4  when
5    notch : KStdHighStressNotch(designState == DesignState.Plausible)
6    plate : KModelGenericPlate(designState == DesignState.Active,
7                               t : plateType.thickness > 17.5)
8  then
9    notch.setRadius(2*t);
10   update(notch);
11 end
```

In line 5, all candidate solutions of type *KStdHighStressNotch*, i.e. of type C, are selected. Instances of other notch types are ignored. Inheritance is exploited. In line 6 f., the plate thickness of the selected plate is tested. A minimum plate thickness of 17.5 mm is required. In the action part, the actual notch radius is calculated. The design element is specifically configured for the design problem at hand.

With additional criteria processed, results can be constructed as shown for bracket design, see section 7.1. In most cases, a single candidate solution is obtained for the given design standard. If not, plausible solutions are presented to the user. This is currently realized in list form shown within a separate log window. It is then the user's responsibility to select and implement an optimal solution. Also, finally, cleanup operations are performed.

7.2.5. Design Results

For notch design, the test case presents 5 different design problems that need to be handled correctly by the design algorithm. Using manual selection and activation, the complete set of rules lead to the design results as shown in figure 7.12.

In position (1), the weld seam located approx. 100 mm next to the longitudinal wall midships to be considered. Here, a notch of type D is defined. Due to high dynamic stress levels, a notch of type C is applied on the other end of the transverse girder, see position (2). For the

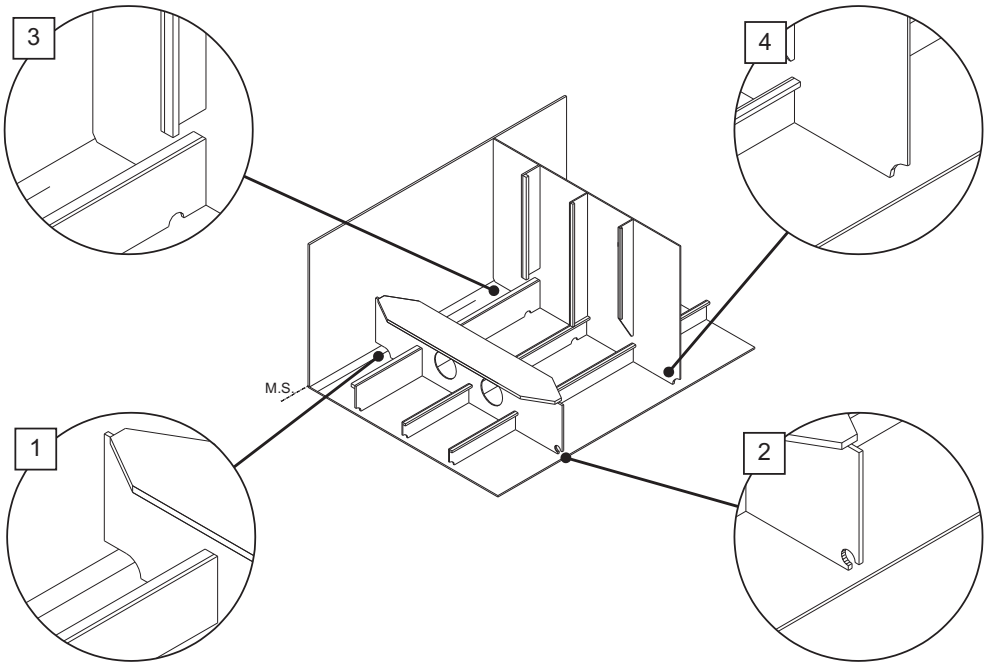


Figure 7.12.: Design Results for the Notch Design Activity

inner edge of the tank wall, a watertight solution is required. Therefore, a notch of type *A* is applied. In addition, for the two longitudinal stiffeners that penetrate the tank wall, weld stop notches are also required. Finally, a notch of type *B* is defined for the outer edge of the transverse wall. Here, the default case, i.e. non-watertight and no special requirements with respect to dynamic loads, is given.

7.3. Collar Plates and Cutouts as Complex Parts

In detailed ship structural design, cutouts are used to define an opening in a plate or girder to provide space for an intersecting profile. Hence, cutouts are feature definitions that operate on an existing plate part. Collar plates are used to physically connect the intersected plate or girder with the profile. These are defined as physical parts. Together, the design of cutouts and collar plates can be classified as a compound design activity of a complex part. In this section, the rule-based problem and solution formulation for this complex design problem is presented.

7.3.1. The Design Standard

As part of the evaluated design standard, detailed information about the standardized design solutions for cutouts and collar plates is given. A set of types of cutouts and collar plates is given. For each type of cutout and collar plate, a set of instances with dimensions is defined. With respect to the dimensions of the profile, the field of applicability is given for each instance. As certain types of collar plates can only be used for specific cutout types, a correlation of collar plates and cutouts is defined.

Depending on the design context, different requirements regarding the function and contour of the opening as defined by a cutout definition are present, see the following paragraphs. Also, collar plates may be required to fulfill different functional requirements. Within the evaluated shipyard design standard an explicit definition of such functional aspects is not given, i.e. this is considered as basic knowledge an engineer is required to possess. As automatic or semi-automatic KBE algorithms for the design of collar plates and cutouts require a structured representation of all relevant requirements and functional relations, such a categorization has been developed in close cooperation with the engineers at the shipyard as part of the work presented. The main concepts derived are introduced in the following.

Cutouts Within the design standard analyzed, a total of 14 different types of cutouts is defined. For each cutout type, up to 30 instances are given where each cutout is defined as to be compatible with a single or multiple intersecting profiles. Focusing on cutout types applicable for the intersection of bulb profiles with plate-like structures only, 11 out of these 14 types can be used. As an efficient application of knowledge-based engineering algorithms is only possible if high numbers are present for a given problem, only common solutions are regarded in the following. Using the results from the quantitative analysis presented in section 5.4, for generic profile-plate intersections, four main cutout types can be identified. These are shown in figure 7.13.

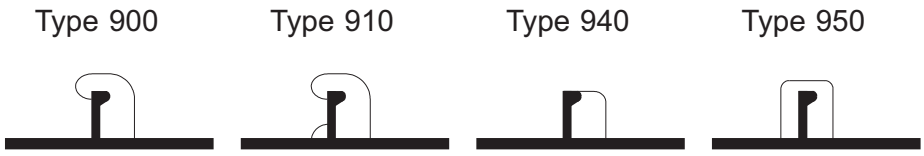


Figure 7.13.: Main Cutout Types

In addition to the clearance which a cutout provides for the installation of a profile, the design of the cutout contour is governed by the following factors:

- load transfer,
- order of assembly,
- weld clearance.

With the ship structure being assembled out of structural elements like girders, profiles and plates, local and global loads need to be distributed within the ship structure. In case of an intersection of profiles and plate-like structures, some kind of direct connection between plate and intersecting profile is therefore required. As governing dimensions here, the total connection length between profile and plate is of importance. As the total connection length determines the maximum load the solution can tolerate, therefore, requirements for this measure are stated within class rules.

If load transfer is required, cutouts of type *900*, *910* and *940* can be applied. Depending on the size and type of the load, the required level of connectivity may vary. For higher requirements more complex and costly solutions need to be chosen which provide a longer connection length. If load transfer is not mandatory, a connection between plate and profile can be seen as optional. Therefore, a cutout of type *950* can be used which provides a basic and hence less cost-intensive solution.

For assembly, two different manufacturing techniques can be applied as shown in figure 7.14. These are optimized to the manufacturing process of the shipyard, the block structure of the vessel as well as the general order of assembly. On the one hand, a plate structure with installed profiles is used. A second plate is lowered orthogonally onto this panel. In this case, cutouts need to be selected such that sufficient clearance is given to lower the second plate from above. Here, for bulb profiles, the bulb needs to be taken into account. In a not that common approach towards manufacturing, on the other hand, both plates are joined first. In a second step, the profiles on the plate are installed, i.e. they are slid into position. For such a slotted assembly, the profiles are mounted and welded onto the plate after the installation of the second plate.

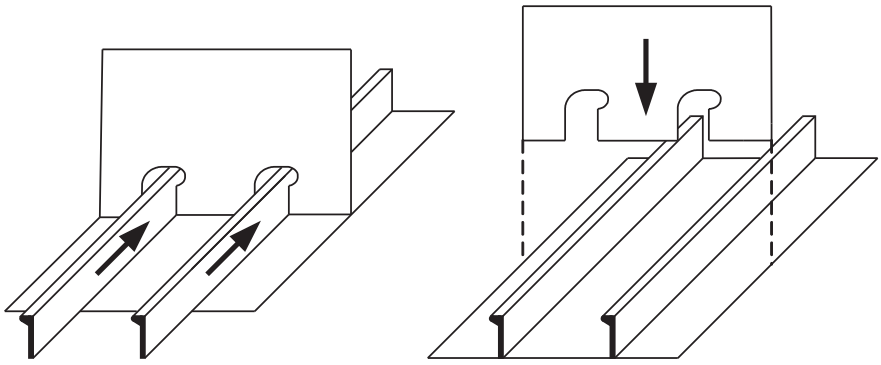


Figure 7.14.: Order of Assembly as Relevant for Cutout Design

Finally, in case of weld operations or to act as water culvert, a clearance at the connection of profile and plane might be required as provided by cutouts of type *910*.

Collar Plates In ship structural design, collar plates serve as additional connection between plate-like structure and intersecting profile. An increased connection length is achieved. Within the analyzed design standard, 7 different types of collar plates are defined. Taking into account only collar plates that are compatible with the selected major cutout types as shown in the previous section, only five standardized solutions named *010*, *020*, *070*, *140* and *170* need to be considered, see figure 7.15. For each type, up to 16 instances with dimensions that are compatible with the corresponding cutouts are defined, i.e. a compatibility matrix of collar plates, cutouts and profiles exists.

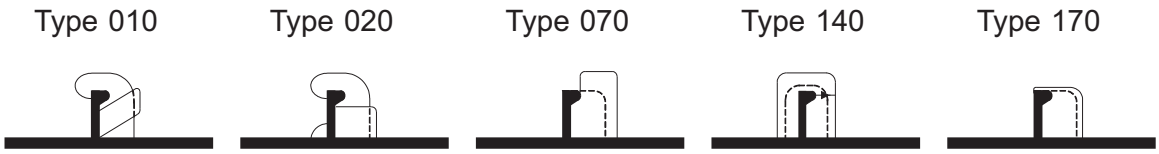


Figure 7.15.: Main Collar Plate Types

For the installation of collar plates, two different functional requirements can be identified, namely:

- load transfer,

- watertight compartmentation.

Depending on the cutout type used, load transfer can be realized by a direct connection of profile and plate-like structure via a cutout of type 900. In contrast, if there is no direct connection, as is the case, e.g., if a cutout of type 950 is used, collar plates are applied. In case of high load levels, the connection length of a single-sided connection might be insufficient. Here, an additional collar plate can be installed hence satisfying the requirements as mandated by class regulations. To facilitate load transfer, the design standard analyzed defined three types of collar plates, namely type *010*, *020* and *070* as shown in figure 7.15.

If the intersected plate needs to provide a watertight compartmentation, collar plates acting as sealing plates are required. For this purpose, the collar plate types *140* and *170* can be applied. For cutouts providing a direct profile-plate connection on a single side, the types *070* and *170* can be used. If the cutout provides clearance on both sides of the profile, i.e. for cutouts of type *950*, a sealing plate consisting of two elements is used, see the cutout type *140*.

For the selection of the optimal combination of cutout and matching collar plate type, the given functional requirements need to be met. This is mandatory. Due to the costs associated with welding operations, the minimization of the weld length required is of importance. Therefore, valid solutions with a reduced connection length are preferred, i. e. solutions with a cutout of type *900* to *940* and a single additional collar plate of type *010* or *20*, if required, are preferred over design solutions with collar plates of type *140*.

7.3.2. The Test Case

For the validation of the KBE algorithms developed, the test case presented in the previous examples is used and extended. As part of this test case, requirements regarding strength and watertightness are already given, see figures 7.2 and 7.9.

In addition, for the test case an order of assembly as shown in figure 7.16 is used for profiles, walls and transverse girders. As it is assumed that the section is built upside down, the transverse wall is installed on the deck followed by the longitudinal profiles on the deck. The profiles are hence installed slotted. In a third step, the transverse girder is installed by lowering it onto the deck that is equipped with profiles.

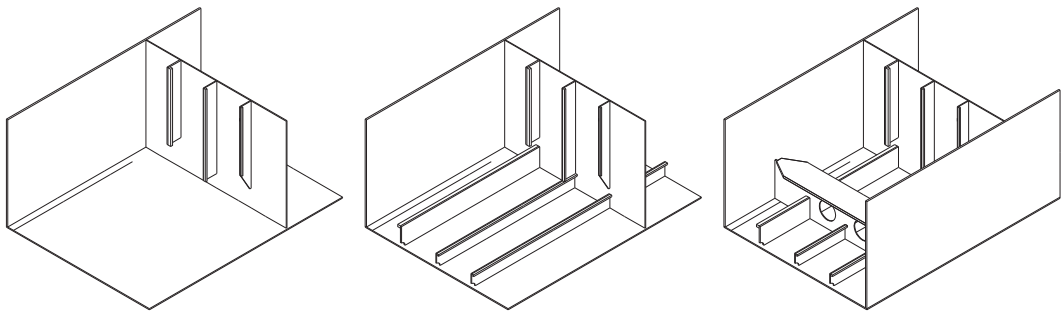


Figure 7.16.: Assembly of the Test Case for Cutout and Collar Plate Design

The aft section of the innermost profile installed on deck has a different orientation. Here, the profile is oriented with the bulb facing the center line whereas the outer two profiles have

a bulb facing outwards. While not being a common design scenario, this border case places special requirements on the design algorithms regarding the selection of applicable cutout contours. Hence it has been chosen to provide a means for verification and is used for the validation of the design algorithms developed only.

7.3.3. The Data Model

For standardized cutout and collar plate types the modeling approach chosen is similar to the previous two examples. An OWL ontology is used to define relevant concepts and corresponding instance data as given by the design standard. The major new aspects of the data model are shown in figure 7.17. As part of the data model, functional aspects are regarded. Also, the field of applicability needs to be given for cutout and collar plate instances. For this purpose, *kpart:validFor* object properties are used to hence link potentially feasible combinations of collar plates, cutouts and adequate profiles. As a result, such properties are used to define a list of applicable stock profile types a cutout is applicable for. Also, for collar plates the range of applicability with respect to cutouts can be specified.

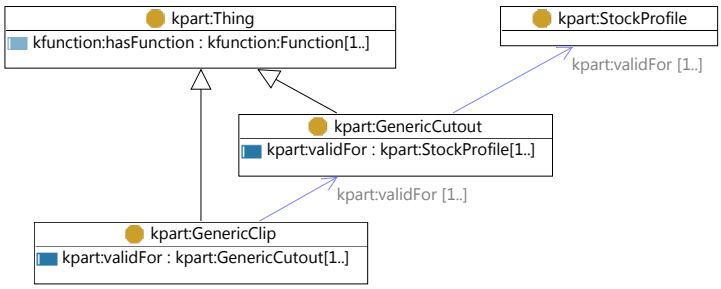


Figure 7.17.: Data Model for Cutouts and Collar Plates

To represent the ship structure, a direct import into the rule engine as introduced in section 6.3.2 is used. As input, an XML representation is used. As topology and connectivity of ship structural elements play an important role in the design of cutouts and clips, the expressibility of the TXHBD exchange file format as defined by Aveva has been extended. Using the topological model presented in section 7.1.3 as reference, see figure 7.5, the XSD schema definition for TXHBD has been extended with the corresponding XML constructs. Hence, elements for the definition of geometric primitives like planes or normals and also for the representation of topological relationships between parts of the steel structure are available. In addition, the file format is extended as to also allow the representation of functional aspects. Hence, functional attributes can be assigned for the parts defined within the data model. As a result, design decisions based on functional properties of given elements of the ship structure are made feasible.

In general, two different data sources are used for data manipulation and storage. On the one hand, all standardized design standards for this test case are developed as part of an ontology which is loaded into the working memory upon rule-engine execution. On the other hand, the ship structural model is exported from Tribon to the extended TXHBD XML-based data format, where topology etc. is added manually. In a second step, for each individual design problem, the relevant data is then directly and automatically loaded from this intermediate

database into the KBE environment. Where required, transformation operations might be performed.

For the import of the relevant structural data, the so-called *Smooks Data Loader* [146] is used. Data defined in the TXHBD compliant XML file which represents the steel structure of the selected design problem is hereby mapped into Java objects. A Java class hierarchy is created and populated with the imported instances.

During the transformation process, the XML model is read and mapping operations are performed, if defined. For this purpose, the data loader uses a pattern-matching approach. Similar to the rule-formulations used throughout this thesis, hence, transformation rules can be defined. If a certain input configuration as defined in a transformation configuration is found, the corresponding action is executed [154].

If, e.g., a definition of a plate instance given as part of the TXHBD data model may be transformed into a corresponding Java instance of class *KModelPlate*, this transformation may be expressed as follows:

```
1 <resource-config selector="plate-entity">
2   <resource>org.milyn.BeanPopulator</resource>
3   <param name="beanClass">org.uro.konsens.kmodel.KModelPlate</param>
4   <param name="bindings">
5     <binding property="id" type="String" selector="plate-entity/ID"/>
6     ...
7   </param>
8 <resource-config>
```

Within a single transformation configuration, the pattern to be matched is given as part of the transformation container, see the *selector* attribute in line 1. For each XML element of type *plate-entity*, this pattern matches and the parameter mappings, i.e. the so called *Bindings* as shown in line 4 ff., are executed. In this example, a single binding definition is given. Each *ID* assignment given as part of the XML data is defined as attribute *id* of the generated instance of type *KModelPlate*. Relevant attributes of the plate definition are hence mapped to the class instance. In addition, more elaborate transformation configurations are used to preserve links between objects.

For the design scenario considered, it is assumed that the design context is fully known. Third-party information not present in the data model is abstracted into manager components as introduced in the previous test cases. The profile and plate, where cutout and possibly collar plates should be defined, are manually selected and hence designated via a design role statement. The design role is set to *DesignRole.Active*. For the selection of a suitable cutout type, the orientation of neighboring profiles is of importance. Therefore, these neighboring profiles are involved in the design process. To represent this situation, an additional design role named *DesignRole.Involved* is introduced.

In the previous examples, an unambiguous design state is given, i.e. at any point in the reasoning process, the design state is fully defined. A standard solution can therefore be either plausible, valid or invalid. Intermediate states are not allowed. For standards resulting in a single valid solution for every possible and allowed design problem, this is a viable approach.

If multiple valid design solutions are possible for a certain design problem, yet an automatic selection of the best solution should be achieved, a different approach is required. To find a valid and optimal solution, a qualitative assessment of the solutions needs to be performed. Therefore, to represent the design state, a more advanced modeling approach has been chosen.

The concept of *Total Quality* is used to denote the level of fitness of a solution. As multiple criteria can influence the determination of the total quality, the concept of design state is extended as shown in figure 7.18.

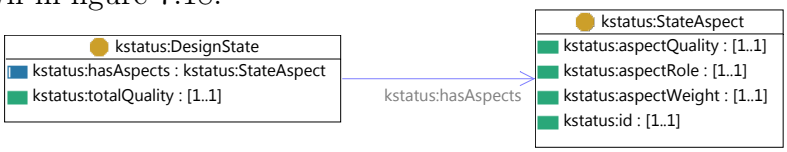


Figure 7.18.: Extended Design State

Here, the handling of quality aspects for standardized parts is performed within the class *DesignState* where this class acts as container of all qualitative considerations. Wtih this class providing the interface methods for a convenient handling of qualitative aspects, object properties of type *hasQuality* are used to reference the design state object from an instance of a standardized type, i.e. to attach such a design state representation to an arbitrary object like a cutout type etc.

As a result, the total quality of a design solution can be calculated. Quality measurements of multiple parts or features can be merged. Following the approach presented by [155], a weighted and normalized design quality can be calculated. The design quality of a solution, i.e. the suitability of a given standardized design element for a given design problem, can be judged according to various *Design Aspects*. Hence, within the data model these different aspects are defined by so-called *StateAspects* and are added to the corresponding *DesignState* representation. Each *StateAspect* instance is responsible to represent a qualitative measurement with respect to a specific *aspectRole* like cost or weight of the solution etc. For the classification of aspect roles, a descriptive name is currently used.

For the design state of standardized parts, two different types of aspects need to be considered, namely:

- soft design aspects,
- hard design constraints.

Soft Design Aspects represent the quality of a design solution with respect to given design aspects. As aspects, e. g., functional properties can be referenced via the *aspectRole* definition. Hence, cost effects or manufacturability can be considered.

With the objective of calculating the overall quality of a design element, standards instance or design state object, the quality q_i of a single design state aspect i is defined by the *aspectQuality* property. For the definition of the aspect quality, normalization is used, i.e. it is: $q_i = 0 \ldots 1$. In addition, the contribution of a single design state aspect to the overall quality can be influenced by a so-called weight w_i, see the *aspectWeight* property. Then, the total quality Q_T of the design element the design state object is attached to can be calculated as averaged sum over all qualitative aspects considered.

$$Q_T = \sum_{i=0}^{n} \frac{w_i \cdot q_i}{n} \qquad \forall q_i \neq -1 \tag{7.2}$$

As the design solution process applied is based on a subsequent culling process, invalid designs violate so-called *Hard Constraints* and should be excluded from further processing

by the KBE algorithms. Hence, any standard solution which does not satisfy a certain hard constraint, should be removed from the set of plausible results, i.e. the constraint should act as criterion for exclusion. For this purpose, an aspect quality of $q_i = -1$ can be set for the design aspect violated. The total quality calculates to $Q_T = -1$.

$$Q_T = -1 \qquad \exists \quad q_i = -1 : \quad i = 0 \ldots n \tag{7.3}$$

As result, a quality in the range $0 \leq Q_T \leq 1$ is derived for feasible design proposals. A quality of $Q_T = -1$ denotes an invalid design.

7.3.4. Rule-based Solution

Similar to the test case of bracket design, see section 7.1, for the design of cutouts and clips the rule engine is acting on a populated working memory, i.e. all relevant design standards as well as the structural model that represents the design problem are loaded. As part of the design activity given, two consecutive design stages can be identified, namely the selection of

- cutouts,
- collar plates.

In the first stage, standardized cutouts are analyzed based on the known design context. Invalid cutouts are marked as invalid. For valid cutout instances, the design quality is calculated and candidate design solutions are created as a result. In a second step, if required, collar plates are selected for the intermediate results obtained. Again, quality aspects are regarded. For eased maintenance, rule partitioning is used to separate these two design stages. For this purpose, agenda groups are defined accordingly. In the following, selected rules are presented so that major aspects of the approach taken can be shown. As this thesis concentrates on the development of approaches for rule-based problem solving methods the solution outline is presented only. Not all rules relevant for a complete solution are shown.

7.3.4.1. Cutouts

As shown in section 7.3.1, the order of assembly plays an important role for the selection of feasible cutout types and instances. For this case, the cutout types that can only be used for the assembly sequence of plate first, profiles later, do not provide sufficient clearance. Therefore, a hard constraint can be formulated in so far as cutouts need to provide sufficient clearance, if the intersected plate is installed last. Expressed as rule, this is defined as follows:

```
1  rule "Plate installed after profiles - Clearance"
2  agenda-group "Cutouts"
3  when
4    plate : KModelGenericPlate(designRole == DesignRole.Active,
5                               assemblyOrder = AssemblyOrder.ProfilesFirst)
6    cutout : KStdGenericCutout(assemblyType not contains AssemblyType.Lowered)
7  then
8    cutout.setQuality(Quality.Invalid);
9    update(cutout);
10 end
```

In line 4 and 5, the order of assembly of the selected plate is analyzed. It is required that the plate is installed after the profiles as defined by the *AssemblyOrder.ProfilesFirst*

statement. Cutouts which are not usable for this order of assembly are retrieved in line 6. As a consequence, matching cutouts are marked as invalid, see the first line of the action part of the rule. Therefore, the convenience method *setQuality* is used to set the corresponding quality factors. Finally, the working memory is updated to accommodate to the changed information base and to trigger a reassessment of the agenda.

Cutout types with a direct connection of the plate to the profile can per se be used for the plate-after-profile assembly procedure. Yet, if multiple asymmetric profiles like bulb profiles with different orientations are used on a single edge of a plate, the clearance provided by such types is insufficient.

```
1  rule "Plate installed after profiles - different profile orientations"
2  agenda-group "Cutouts"
3  no-loop true
4  when
5    plate : KModelGenericPlate(designRole == DesignRole.Active,
6                               assemblyOrder = AssemblyOrder.ProfilesFirst)
7    KModelGenericProfile(designRole == DesignRole.Active,
8                         orientationA: orientation)
9    KModelGenericProfile(designRole == DesignRole.Involved,
10                        orientation != orientationA)
11   cutout : KStdGenericCutout(functions contains LoadTransfer.LoadTransferDirect)
12 then
13   cutout.setQuality(Quality.Invalid);
14   update(cutout);
15 end
```

The assembly order of the plate is evaluated in line 5 and 6. The plate is installed after the profiles as given by the *AssemblyOrder.ProfilesFirst* designation. The active profile is retrieved in line 7. For this profile, the orientation is stored as local variable *orientationA*, see line 8. In the following two lines, a second profile with a different orientation is searched for. As final conditional clause, a cutout that lists a function of *LoadTransferDirect* is queried for. The cutout is assigned to the variable *cutout* with rule scope.

If all conditions are met, an infeasible solution has been found for the design problem given. On a common edge, profiles of different orientation are present. A cutout with a direct connection of profile and plate is retrieved. Thus, in the action part, the cutout is marked as invalid.

```
1  rule "Ignore cutouts with LoadTransferDirect if no load transfer is required"
2  agenda-group "Cutouts"
3  when
4    plate : KModelGenericPlate(designRole == DesignRole.Active)
5    profile : KModelGenericProfile(designRole = DesignRole.Active)
6    not StrengthManager.requiresConnection(plate, profile)
7    cutout : KStdGenericCutout(functions contains LoadTransfer.LoadTransferDirect)
8  then
9    cutout.setQuality(Quality.Invalid);
10   update(cutout);
11 end
```

If load transfer is not defined as a requirement, a connection of profile and plate is not mandatory. In the rule formulation above, this is tested via an external manager component *StrengthManager* for each design problem as stated on line 6. If no connection is required, all cutouts that provide such a direct connection of plate and profile are retrieved, see line 7, and are removed from further reasoning. They are marked as invalid. Based on an interpretation of the design standard, in this test case, therefore, the requirement for *No Connection* is interpreted as a hard constraint, i.e. no cutout definition with a connection of plate and profile is assumed to be a valid candidate solution if such a connection is not required.

In contrast, if load transfer is required, either a single- or a two-sided connection can be used. For a single-sided connection, a direct connection or a connection via a collar plate or clips can be chosen.

```
1  rule "Direct connection has higher quality"
2  agenda-group "Cutouts"
3  when
4    plate : KModelGenericPlate(designRole == DesignRole.Active)
5    profile : KModelGenericProfile(designRole = DesignRole.Active)
6    StrengthManager.requiresConnection(plate, profile)
7    cutout : KStdGenericCutout(loadTransfer contains LoadTransfer.LoadTransferDirect)
8  then
9    cutout.updateQuality("cost", 1);
10   update(cutout);
11 end
```

Mainly due to concerns regarding cost, a direct connection of profile and plate is preferable as the number of parts required and manufacturing time needed can hence be reduced. Therefore, cutouts applicable for direct load transfer are preferred, see line 9. For such cutouts, a high qualitative rating regarding cost is assigned. In a similar way, the quality of cutouts which mandatorily require collar plates can be degraded as follows:

```
1  rule "Indirect connection via collar plate has lower quality"
2  agenda-group "Cutouts"
3  when
4    ...
5  then
6    cutout.updateQuality("cost", 0.2);
7    update(cutout);
8  end
```

If profiles are installed after the plate, an improved weldability is given in case of clearance provided on both sides of the profile. As a result, a higher built quality is assigned to cutouts of type *910*. For the functional aspects *Manufacturability* and *Cost* a positive quality ratio is given. Similarly, missing weld clearances may lead to a decrease of said aspects.

With the configuration and evaluation of cutout instances complete by a set of rules as shown in excerpt above, all cutout instances that are not applicable for the current design problem and within the given design context are marked as such. As a result, for the remainder a series of so-called intermediate result objects can be created. These result objects serve as prototype for the final solution where each result contains a preliminary definition of a design solution. The profile and plate concerned are referenced. A feasible cutout is given. Quality aspects are defined for the cutout.

```
1  rule "Create intermediate result prototypes"
2  agenda-group "Cutouts"
3  salience -50
4  no-loop true
5  when
6    plate : KModelGenericPlate(designRole == DesignRole.Active)
7    profile : KModelGenericProfile(designRole = DesignRole.Active)
8    cutout : KStdGenericCutout(quality not Quality.Invalid)
9  then
10   ccSolution = new KDesignCutoutCollarPlateSolution();
11   ccSolution.setProfile(profile);
12   ccSolution.setPlate(plate);
13   ccSolution.setCutout(cutout);
14   ccSolution.setDesignState(DesignState.IntermediatePrototype);
15   insert(ccSolution);
16 end
```

In the conditional clause, the selected plate and profile are retrieved, see line 6 and 7. Any cutout not marked as invalid is used, line 8. For each combination of plate, profile and cutout

an intermediate solution prototype is created in the action part of the rule. References to the design objects selected are created. The object is inserted into the working memory.

The design solution of the object created is marked as *DesignState.IntermediatePrototype*. The result therefore is not a final design. It is a prototype, i. e further configuration steps may be necessary to reach a final solution. Depending on feasible configurations for the collar plates, more than one candidate solution may be created using a single intermediate result prototype.

Finally, as shown in the example regarding notch definition in section 7.2, the next design step is activated. The focus of the rule engine is set on to the following agenda group named *CollarPlates*.

7.3.4.2. Collar Plates

For a given design problem, different types of collar plates can be used. For each type, multiple instances with differing dimensions are given. For each of these instances, the field of applicability is restricted to specific cutout dimensions as defined in the standard. Therefore, in a first step, any cutout that is not compatible with the remaining plausible cutout definition is removed from further processing.

```
1  rule "Retract invalid collar plate instances"
2  agenda-group "CollarPlates"
3  when
4    cPlate : KStdGenericCPlate()
5    not exists KStdGenericCutout(designState not DesignState.Invalid,
6                                 validFor contains cPlate)
7  then
8    retract(cPlate);
9  end
```

Collar plate instances are retrieved from the working memory in line 4. For each collar plate instance, the rule is executed. The existential quantifier *exists* is used in the following statement. The quantifier returns true if at least a single object fulfills the criteria given. In this case, in line 5, the negated version of this quantifier is used. It is tested if there is no single cutout object of a feasible design state which is valid for the collar plate instance assigned in the previous statement.

If this is true, the collar plate instance is not applicable to the design problem at hand. With the objective to show different approaches towards rule formulation, a different modeling approach is used in contrast to the previous examples. Here, invalid collar plate objects are removed from the working memory. They are retracted, see line 8, and are thus not available for further reasoning steps.

Similarly, any non-watertight collar plate provides an infeasible solution for the design problem if a watertight intersection of plate and profile is required. Hence, they are retracted from the working memory by a corresponding rule, too. Vice versa, if a non-watertight solution is explicitly required, any collar plate instance that leads to a watertight result is also removed.

Incomplete Information In reality, the information base the design process acts on is incomplete. Not every relevant aspect of the design context is known and defined in the product

data model at the time when a design activity takes place. Therefore, for any piece of information present as part of the design, in general, three states can be distinguished, namely *true*, *false* and *unknown*. For the aspect of loads as given in a planned cutout location, this translates to:

- high loads
- normal or no loads
- unknown

Closed-world reasoning, as applied exclusively so far, assumes that any statement which is not known to be true is regarded as false. An unknown fact is also set to false. In case of incomplete information, this is obviously not always valid. E.g. under closed-world reasoning and in case of missing information regarding the load level present in the location of a cutout connection, any test for the default situation *defaultLoads* returns false, hence a high strength solution is incorrectly suggested by the design algorithm.

In contrast to such closed-world reasoning, under the so-called *Open World Assumption* (OWA) the truth value of a statement is independent of whether the fact is known or not. It is assumed that an unknown fact can be true or false. Under the OWA, therefore, each fact or statement is therefore modeled as tristate [35] and hence fulfills these three different states. Any comparison with such an attribute returns a truth value as given in table 7.2. As an unknown value of *b* can take arbitrary values, therefore, a return value of true is generated in case of an unknown parameter.

Condition	**Closed World**	**Open World**
a == b	true	true
a != b	false	false
b unknown	false	true

Table 7.2.: Comparison of Closed-World and Open-World Scenarios

In case of optional parameters which need to be considered as part of a design algorithm, closed-world-reasoning requires at least two individual rule formulations as follows to handle the default case correctly:

```
1  rule "Optional Parameter missing"
2  when
3    KModelGenericPlate(hasLoad == null)
4  then
5    ... (default action)
6  end
7
8  rule "Optional Parameter set to Default Value"
9  when
10   KModelGenericPlate(hasLoad = Loads.defaultLoads)
11 then
12   ... (default action)
13 end
```

For the example of loads acting at the location of a cutout design problem, here, the first rule is defined which is executed in case of the optional parameter *hasLoad* missing. In this case, the default design decision is taken. For the second rule, the conditional part matches

if this optional parameter is given and if it is set to the default value. Again, a design default is chosen.

Using the open-world assumption, these two rules can be merged. Hence supplementary rules for the case of unknown data are not required if open-world reasoning can be applied. For this purpose, a custom operator has been added to the rule engine, see section B in the appendix for implementation details. A comparison according to open-world rules has been implemented. The *Equal Or Null* (EON) operator compares two objects as shown in table 7.2. If the right-hand side object is null, the comparison returns true independent of the value of the left-hand side. As a result, both rules shown above can be merged into a single expression.

Applied to the problem of cutout design, a not highly stressed solution can be considered as the common or default solution. If no requirement regarding loads is given, thus, no special requirements regarding connection length are given. Using the custom operator *EON*, this can be expressed as follows:

```
1  rule "Default Loads as Default (EON)"
2  agenda-group "CollarPlates"
3  when
4    KModelGenericPlate(designState == DesignState.Active,
5                       pLoads: hasLoads)
6    Function.NormalLoads eon pLoads
7    cutout : KStdGenericCutout(functions not contains Function.HighLoads)
8  then
9    cutout.setQuality(Quality.Invalid);
10   update(cutout);
11 end
```

The functional requirement of the plate regarding loads is assigned to a local variable *pLoads* in line 4 and 5. A convenience function is used to extract the relevant functional aspect from the list of functions. In line 6, the *eon* operator is used to evaluate whether a standard configuration *Function.NormalLoads* is required. Due to the definition of the operator, the functional requirement of the plate has to be given on the right-hand side of the comparison. If the first two statements of the conditional part of the rule are satisfied, a standard solution is aimed for. Any cutout plate is marked as invalid and is hence excluded from further processing, see line 7 and 9 that is primarily targeted towards situations where high loads are given.

Using this approach, additional optional parameters as required for the selection of adequate standardized collar plates are evaluated. In addition, the remaining solutions can be further evaluated with respect to additional criteria. For this purpose, rule formulations under a closed-world-assumption are applied.

7.3.4.3. Results Generation

In the previous two sections, concepts for the evaluation of collar plate and cutout instances have been presented. In addition, the concept and configuration of so-called prototype design solutions, which are in this case based on valid cutout instances, have been introduced.

As a result, preliminary results are defined for valid cutouts. Yet, valid collar plates and the respective feasible combinations of valid cutouts and collar plates are so far not considered. For the design activity to be finished, therefore, the information presented as part of the solution prototypes needs to be augmented. As shown in figure 7.19, further information

regarding the installation of collar plates, if required, needs to be added. As a result, a set of feasible design candidates is to be created. Finally, these candidates are evaluated; an optimal result needs to be found.

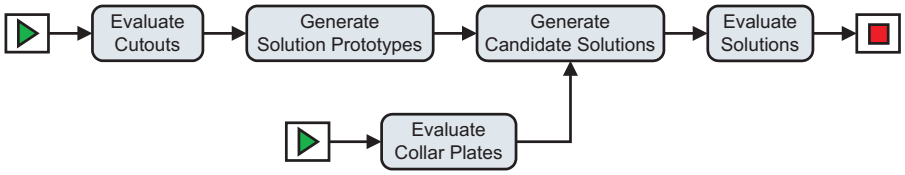


Figure 7.19.: From Solution Prototype to Design Candidate

For each possible configuration of prototype result and collar plate available, a so called *Candidate Solution* is created. As an example, candidate solutions are created as shown below. It is assumed that collar plates are installed single-sided, only.

```
1  rule "Construct candidate solution - collar plates single sided"
2  agenda-group "Cutout CPlates Results"
3  when
4    resultProto : KDesignCutoutCPLateSolution(
5                    designState == DesignState.IntermediatePrototype,
6                    cutout : hasCutout)
7    cPlate : KStdGenericCPlate(isValidFor contains cutout)
8  then
9    result = resultProto.clone();
10   result.setDesignState(DesignState.Candidate);
11   result.setCPlate(cPlate, Installation.SingleSided);
12   insert(result);
13 end
```

As part of the conditional part of the rule, the prototype solution is retrieved in line 4. Also, any collar plate present in the working memory, i.e. that was not retracted during the execution of the previous agenda-group, is found, see line 7. Compatibility is considered in so far as the collar plate retrieved needs to be applicable to the cutout instance defined as part of the prototype result *resultProto*.

Once a result protoype has been found, a so-called design candidate solution can be created. For this purpose, the prototype object is cloned in the first statement of the action part, see line 9, i.e. a copy of the prototype object is created which contains all characteristics already defined in the prototype. The design state is set to *DesignState.Candidate*. The selected collar plate instance is assigned to the result and the installation type is given as additional argument.

In addition to the rule formulation above, no valid collar plate is found if no direct connection of plate and profile is required. Hence, further rules are required to define candidate design solutions for double-sided collar plates. For this purpose, it needs to be tested whether the referenced cutout in the prototype supports a single- or double-sided installation of collar plates and corresponding actions may need to be defined.

For all design candidates generated, additional constraints may be given or further configuration steps may be needed. E.g.,with regard to load transfer, two different requirements and constraints can be identified, namely:

- the length of the connection needs to be sufficient,
- a single-sided connection is preferable.

Depending on the load level, the class rules define a minimal required connection length between profile and plate. For high loads, a two-sided connection may be required. The relevant regulations thus define a hard requirement. From the point of view of manufacturability and cost, a one-sided connection is preferable. As part of the rule formulations used, therefore, a soft constraint should be used to assign a higher level of design fitness for single-sided connections.

To evaluate these requirements, the complete design proposal needs to be evaluated. Two rules are required that act on these proposals. Both aspects mentioned above can be expressed using the rule formulation concepts introduced. If a design candidate violates the hard constraint regarding connection length, the design state of the candidate is either marked as *DesignState.Invalid* or the object can be retracted from the working memory.

With the configuration and evaluation complete, a set of feasible results is given. These results fulfill the hard requirements given by the design context and the information defined in the standards database. Quality criteria are given and the global fitness of the solution candidates can hence be calculated. If an optimal solution needs to be found this can be expressed as follows:

```
1  rule "Find optimal solution"
2  agenda-group "Cutout CPlate Results"
3  when
4    solution : KDesignCutoutCPlatesSolution(designState == DesignState.Candidate,
5                                            sq : quality)
6    not exist KDesignCutoutCPlatesSolution(designState == DesignState.Candidate,
7                                           quality > sq)
8  then
9    solution.setDesignState(DesignState.Optimal);
10   update(solution);
11 end
```

A solution is retrieved and assigned to the local variable *solution*, line 4 and 5. The overall design quality of the candidate result is assigned to the local variable *sq*. The second conditional statement in line 6 f. tests for any solution of a higher quality compared to *solution*. If this is not the case, the solution found via the first statement is an optimal solution. The design state of the solution is changed to *DesignState.Optimal*.

```
1  rule "Process optimal solution"
2  agenda-group "Cutout CPlate Results"
3  when
4    solution : KDesignCutoutCPlateSolution(designState == DesignState.Optimal)
5  then
6    ...
7  end
```

With an optimal solution found, the generated result can be processed. The user is informed about the result achieved. The product data model may need to be updated accordingly. If realized, the changes can be incorporated into the CAD system. Here, collar plates and cutouts with additional meta-data can be defined according to the optimal result. Topology information can be updated.

7.3.5. Design Results

Manual execution of the design activity presented leads to a single design result. Operated iteratively, all required steps towards a feature complete design with respect to cutouts and clips can be performed. For the given test case and the given order of assembly, see figure

7.16, a result as shown in figure 7.20 can hence be obtained. Here, the location of collar plates is drawn as identical with the location of brackets which may lead to interferences of these two different types of design elements. As part of the model, the opposite is true. Collar plates are always located facing forward for the forward part of the vessel. Here, this arrangement has been chosen purely for an improved visualization.

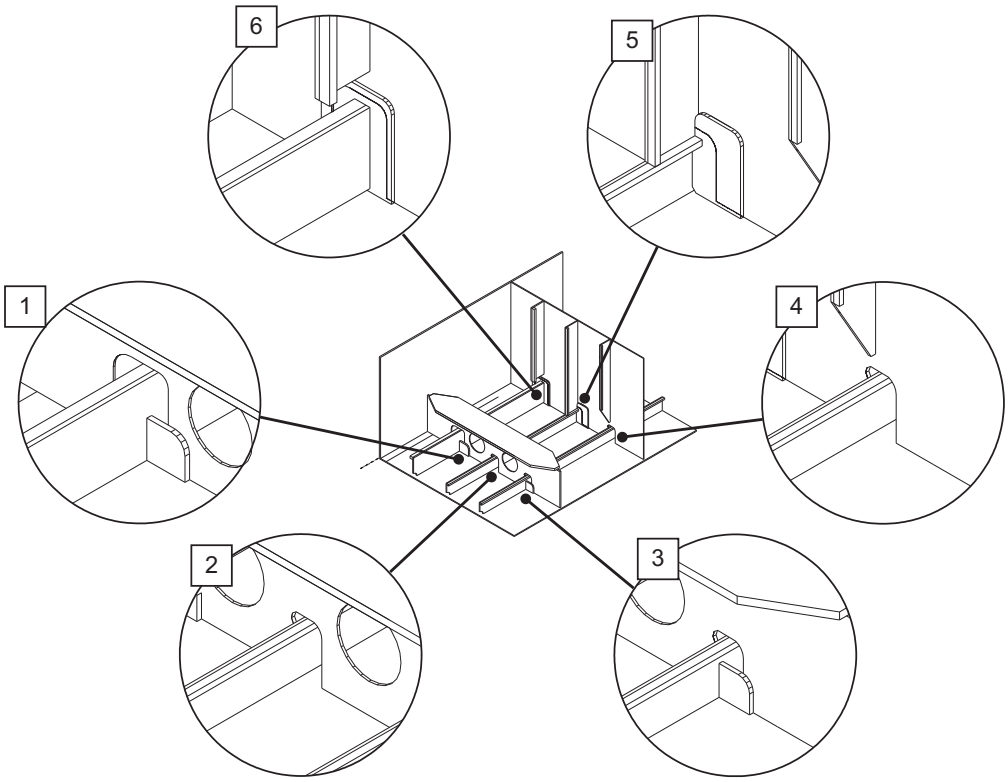


Figure 7.20.: Design Results for the Cutout and Clips Design Activity

As part of the design activity for cutouts and collar plates, in total, 6 locations need to be considered. As the innermost deck stiffener has a differing orientation compared with the outer two deck stiffeners, in location (1) and (6), therefore, cutouts of type *950* are installed. For the girder, the connection of girder and profile is realized by a clip of type 020. In contrast, the tank wall requires a fluid-tight solution. Hence a collar plate of type *140* is required. In position (2) the default case is given, i.e. a single-sided connection of plate and profile is provided by the cutout of type 900. No collar plate is used. For the adjacent HP profile of type *HP 140x7 A36*, see position (3), high stress levels require a double-sided connection. Hence, a cutout from the *900*s series is used in conjunction with a collar plate of type 020. For location (4), a situation identical with (2) is given. Finally, the connection of center deck profile and tank wall, see location (5) needs to be fluid-tight. A default cutout of type *900* is hence sealed with a collar plate of type *070*.

8. Complex Design Patterns

In detailed ship structural design, the definition of the details of the ship structure is of prime importance. As part of this design stage, a large number of individual design solutions needs to be determined. For the definition of each detail, as presented in the previous chapter, a design problem needs to be solved via an atomic design activity. As interdependencies between the designed details may exist, existing solutions may have an effect on the design process for further design problems.

For each single design activity, three major consecutive decisions have to be taken before the actual KBE-based design algorithm can be executed. These can be identified as follows:

- Location: Where should the design activity act on?
- Task: What should be designed?
- Mode of Operation: How is the design activity executed?

For the execution of an identified design activity, the *Location(s)* to operate on needs to be given, i.e. active or involved elements are needed as input knowledge so that the actual design activity can be carried out with a known set of input data. For a plate named *A4500*, e.g., a single design activity may be performed to define a bracket which connects, e.g., a profile with the ID *A4500-P1238* of type *HP140x7* with a second profile named *A4500-P3369* of type *HP120x6*. For each design activity, the selection of active or involved design elements can be within the responsibility of the engineer, see the previous chapter, or can be performed automatically by some knowledge-based algorithms.

Also, any design process requires that the *Task* to be performed, is known, i.e. the design activity to be executed needs to be given. It should be clear what kind of action is to be taken. As an example, for the given panel of the steel structure named *A4500*, brackets are required. The design task to be executed hence is *Bracket Design*.

Finally, some information regarding the *Mode of Operation* is required. Design activities may operate interactively or automatically. In the previous chapter, the implementation of design activities as rule-based algorithms is shown. The focus is placed on the actual design activity. Steps needed to infer a valid solution from a set of input requirements and constraints are explained.

In this chapter, the consecutive execution of multiple design activities is examined. Means for a fully automatic, consecutive mode of operation are developed. In section 8.1, the identification and activation of design elements for a known design task are evaluated, i. e. the *location* this design activity acts on is determined. E.g., to design a profile-plate intersection the plates and profiles to operate on need to be found. Extending this approach, the independent and automatic execution of multiple design activities with a *given task* or objective like, e.g., the design of multiple brackets in a single session, is presented in section 8.2. Finally, the concept of design workflows is introduced in section 8.3. The execution of a *set of design tasks* in a controlled, rule-determined order is presented. It it shown how to

control the design process and which action should be taken for what kind of problem.

8.1. Activation of Design Problems

For any design activity to take place, the design context needs to be known. Or to put it more simply: "If you don't know, where to work, nothing gets done." For this purpose, the concept of activation has been introduced in the previous chapter where an activated design element denotes an element of the product data model that takes part in the design activity. For the test cases of bracket design in section 7.1, e.g., two connecting profiles are selected and hence activated by the engineer in a manual mode of operation.

Activation is therefore a mandatory step before the execution of any problem solving method. Without activated or selected parts and features, the data provided as input to the problem solving method is incomplete and hence insufficient. The design context is not defined in a sufficient level of detail.

For an automated design of details in ship structural design, automatic methods for the determination of the design context are therefore required. With a focus placed on the automatic design of standardized solutions, the process of element selection and activation needs to be performed without human intervention. For this purpose, the product data model representing the ship structure needs to be analyzed. A rule-based approach towards activation as shown in figure 8.1 has been developed. This workflow is executed each time the design context, i.e. the activated elements of a design activity, needs to be determined.

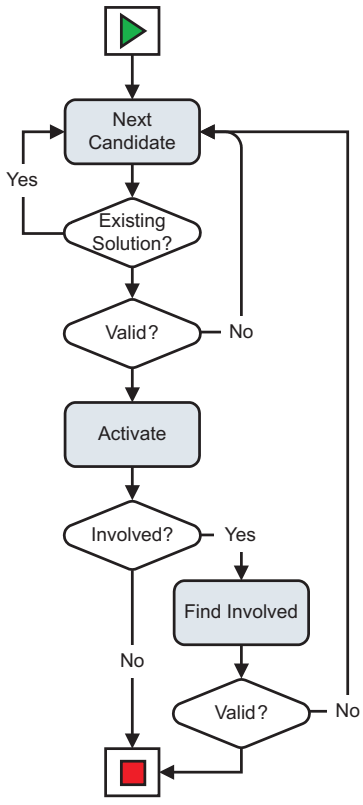


Figure 8.1.: Automatic Selection and Activation of Design Candidates

In a first step, similar to the approach presented for the profile-plate intersection, see section 7.3, the rule engine is used to generate sets of candidates for activation and further evaluation. This is an iterative process, i.e. all feasible combinations are tested consecutively. E.g. for the test case of bracket design as shown in figure 8.2, the profiles *No. 1* and *No. 6* might be selected as candidates by the rule-engine in a single step. This represents an invalid activation.

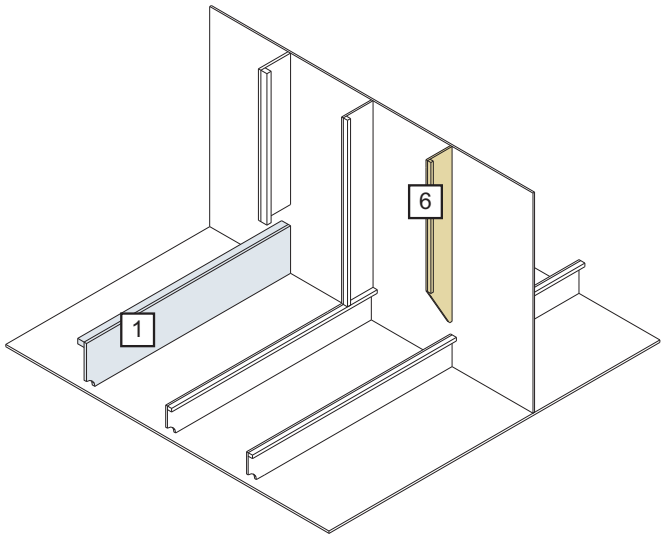


Figure 8.2.: An Example of Candidate Selection for Activation

The validity of such a set of selected elements is then analyzed. The set is discarded, if there already exists a solution for the design activity of choice. A pre-existing solution is hence a criterion for exclusion from further processing. For bracket design, this would be the case if the profiles *No. 1* and *No. 3* are already connected by a bracket. In a second step of verification, the design context is evaluated. The activation generated needs to be valid and standards-compliant. For bracket design, e.g., the type and dimensions of the selected profiles need to be compatible with the shipyard standard. Also, the orientation of the profiles needs to be as defined by the detailed design standard. In case of an unknown profile type or an incorrect dimension, the selection is rejected. A new iteration is started and a new candidate activation is generated.

With a valid activation found, additional design elements that play an integral part in the design activity are determined if required. These are marked as involved. After the process of activation has terminated, finally, the actual design activity is executed as presented in chapter 7.

Candidate Selection and Evaluation Candidates for design activities denote potential constellations of design elements like profiles, plates etc. where a design activity can act on. To explain the process of candidate selection, the test case as presented in section 7.3 is used in the following. Hence, as design objective, the automatic design of cutouts and collar plates is given. For this purpose, two connected plates and an intersecting profile need to be found. This selection defines the primary design context valid for the consequent design activity of cutout design. Feasible candidates can be determined for this case as follows:

```
1  rule "Find and activate candidates"
2  ruleflow-group "Cutout CPlate Activation"
3  when
4    plate : KModelGenericPlate(designState != DesignState.Active)
5    profile : KModelGenericProfile(onPlate : onPlate)
6    GeometryManager.hasIntersection(plate, profile)
7    GeometryManager.hasConnection(plate, onPlate)
8    not exists KModelCutout(connectedTo contains (plate and profile))
9  then
10   plate.setDesignState(DesignState.Active);
11   update(plate);
12   profile.setDesignState(DesignState.Active);
13   update(profile);
14 end
```

For the rule definition, a *ruleflow-group* statement is used. Similar to the concept of *agenda-groups*, a ruleflow group can be used for rule partitioning. Agenda groups are controlled by consequences of rules, i.e. within the rule definition. Ruleflow allows to define the order in which rule sets are evaluated graphically using a flow chart. A representation similar to the one shown in figure 8.1 is used. It can be defined which rule sets should be evaluated in sequence or in parallel.

In the conditional clause, an arbitrary combination of a profile and a plate are selected for evaluation, see line 4 f. As no plate with an active design state should be present in the data model, see line 4, it can be guaranteed that this rule is executed only once for each valid selection. The rule aborts having reached a candidate activation. Hence, the generation of multiple activations in parallel is prevented. An intersection of plate and profile needs to be given for a valid design problem. For this purpose, an external manager component is executed, see line 6, which evaluates the topological relation of the selected plate and profile. If no connection is given, the current selection is rejected and further combinations are tested by the rule engine. Also, the plate the profile is located on, needs to have a direct connection with the other plate, see line 7.

In line 8, a test for an existing cutout for the selected plate and profile is performed. For this purpose, topological relations are explored, i.e. the *connectedTo* statement is evaluated. If there is no cutout which is connected to both the selected plate and profile, a probable candidate activation has been found. As a consequence, a possible design problem has been identified, so that the design state of the selected plate and profile can be set to active. A cutout definition located at the intersection of plate and profile can then be generated by the following design activity as presented in section 7.3.

Design Elements with a Role DesignState.Involved For the test cases of cutout design, the selection of suitable cutout types and instances is governed by the design context. Here, the orientation of the selected profile with respect to the neighboring profiles needs to be considered.

```
1  rule "Mark involved profiles"
2  ruleflow-group "Cutout CPlate Activation"
3  no-loop true
4  when
5    plate : KModelGenericPlate(designState == DesignState.Active)
6    prof : KModelGenericProfile(designState == DesignState.Active
7                                profOn : profileOn)
8    p : KModelGenericProfile(designState != DesignState.Active
9                             profileOn == profOn)
10   GeometryManager.hasIntersection(plate, p)
11 then
12   p.setDesignState(DesignState.Involved);
```

```
13    update(p);
14  end
```

In line 6, the activated profile is found. The topological entity the profile is mounted on, i.e. the plate, is assigned to the local variable *profOn*. In line 8 f., a second profile *p* is chosen. This profile *p* needs to be mounted on the same topological entity with the active profile *prof*. The profiles share a common edge of the plate. Therefore, an intersection of the selected, inactive profile and the selected plate needs to be given, see line 10.

If these conditions are met, the profile is designated as involved in the design process and the design state is set accordingly. As the rule shown above is called for each profile given in the product model, all profiles that have an influence on the design process can be determined. Hence, cutout and clips design can be performed while taking the relevant design context as introduced in figure 7.16 in section 7.3.2 into account.

With a candidate activation given, validation steps can be performed and standards compliance of the proposal can be evaluated. Costs associated with the selected configuration in comparison to other candidate activations may be explored. Relevant aspects can either be used in the generation of activations or during the validation step. A thorough evaluation of the different formulations possible should be performed.

For bracket design as shown in section 7.1, as validation the location of brackets could be evaluated. Under the assumption that brackets should only be defined for every second profile on an edge, a rule formulation as follows could be used.

```
1  rule "Brackets on every second profile"
2  ruleflow-group "Bracket Validation"
3  no-loop true
4  when
5    pl : KModelGenericPlate(designState == DesignState.Active)
6    p : KModelGenericProfile(designState == DesignState.Active)
7    pNeighbour : KModelGenericProfile(designState != DesignState.Active)
8    GeometryMgr.isNeighbour(profile, profileN)
9    bracket : KModelGenericBracket(connectedTo contains (pl and pNeighbour),
10 then
11   profile.setDesignState(DesignState.Unknown);
12   update(profile);
13   plate.setDesignState(DesignState.Unknown);
14   update(plate);
15
16   dummyBracket = KModelDummyBracket(profile, plate);
17   insert(dummyBracket);
18 end
```

The proposed active plate and profile are assigned to the local variables *pl* and *p* respectively in line 5 and 6. An unselected profile is searched for, see line 7. In the following statement it is tested whether the second profile is a direct neighbor of the active profile. For this purpose, the external manager component *GeometryMgr* is used. In line 9, an already existing bracket definition defined on the neighboring profile is sought. The bracket definition references the active plate and the neighboring profile. It it connected to these parts.

The rule is executed for each profile available in the data model. If all statements evaluate to true, a bracket definition is given for a neighboring profile. No bracket should be designed for the selected profile and plate. The design state of profile and plate is reset to the default value *DesignState.Unknown*. The working memory is updated accordingly.

To prevent multiple evaluation of an identical candidate proposal a so-called *KModelDummyBracket* design solution is created and inserted into the working memory, see line 16 f.

With this dummy object, the information present in the product data model is enriched. It is marked that no bracket should be defined in this location. In case of subsequent executions of any rules which search for feasible bracket locations, this placeholder object prevents the subsequent activation of the profile and plate configuration referenced. Hence, no bracket is defined here.

8.2. Multiple Design Activities as One Design Task

Commonly, a design activity for a given type of design problem is executed multiple times. Hence, a so-called design task is given.

Definition 12. *In detailed ship structural design, a design task consists of individual yet similar design activities. In each design activity, a design solution for a given type of design problem is determined.*

Therefore, a design task like, e.g., *Bracket Design* consists of multiple design activities for the definition of a single bracket. Each design activity operates on a different design problem instance. Using the methods for activation introduced in the previous section, for each design activity all constraints and requirements required for the actual design process are known, i.e. the design context is fully determined. Defined as separate ruleflow group, the design activity can be executed as presented in the previous chapter.

A KBE-based execution of multiple similar design activities, hence, requires an automatic selection of the corresponding design contexts, each followed by the execution of the actual design activity. For the design of cutouts and collar plates, therefore, an approach towards automatic design as shown in figure 8.3 can be used.

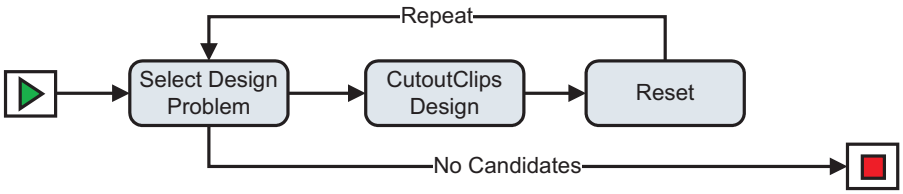


Figure 8.3.: Knowledge-Based Definition of Multiple Cutouts and Collar Plates

Based on a populated working memory, the process of design activation, see the *Select Design Problem* process in figure 8.3, is repeatedly executed to determine the design problem to act on. Once a design problem is determined and evaluated as valid, the actual design activity of cutout and clips design is performed as presented in section 7.2. Based on the standards catalog present as part of the working memory, here, valid cutouts and collar plates are identified and defined. As a result, a solution for the selected problem is determined and added to the product model. Finally, a *Reset* stage is used to clear the working memory from temporary data and to prepare the rule system for another iteration, i.e. for the next design problem.

As a result, for each feasible combination of profile and plate, the corresponding cutouts and clips are designed leading to a design situation as already shown in figure 7.20. In contrast

to the manual approach presented in chapter 7, here, the design pattern can be applied to obtain said results fully automatically. The complete design task terminates, if no further activations can be generated, i. e. if no design candidates can be found.

As part of the rule formulations used, so-called ruleflow groups are used to logically partition these three distinct set of rules. E.g., for the activity of the actual cutout and clips design, see the *Cutout Clips Design* stage in figure 8.3, a ruleflow group named *Design* is used, which consists of multiple agenda-groups that are used to distinguish and hence separate different stages of the design activity, see section 7.2. Similarly, the activation process and the reset stage are encapsulated as separate ruleflow-groups *Activation* and *Reset* respectively.

At runtime, these three different ruleflow-groups need to be alternatively selected for further processing by the rule-engine. The selection stage may terminate once two elements of the design problem are marked as *DesignState.Active*. As a consequence, the following ruleflow-group needs to be selected. This can be performed via the *switchRuleflowGroup*-statement as follows:

```
1 rule "Activation found; Switch to Design Stage"
2 ruleflow-group "Cutout Activation"
3 salience -50
4 when
5   plate : KModelGenericPlate(designState == DesignState.Active)
6   prof : KModelGenericProfile(designState == DesignState.Active
7 then
8   switchRuleflowGroup("Design")
9 end
```

For this rule formulation, a lower level of importance is defined, see the salience statement on line 3. This approach has been chosen to ensure that all rules defined as part of the activation procedure are guaranteed to be executed before the transition to the following design stage is performed. Otherwise, additional rules defining further tests on the generated activation might not be called and hence, incorrect activations might be treated as design candidates.

For each ruleflow-group selected, the initial agenda group to be used within the ruleflow group needs to be defined. A primitive rule without an agenda group can be used for this purpose:

```
1 rule "Cutout CPlate Definition - Set Focus"
2 ruleflow-group "Cutout CPlate Execution"
3 when
4   eval(1==1)
5 then
6   setFocus("Cutouts");
7 end
```

The evaluate statement on line 4 guarantees that the rule is always executed upon entering the ruleflow group. As a consequence, the corresponding agenda group *Cutouts* is activated.

In a following step, a *Reset* ruleflow group is executed. Temporary variables or intermediate objects are removed. Design states, i.e. the selection of elements, are set to a default state. Where required, information is removed from the working memory. Information for reuse may be stored as part of the external meta database. The process is repeated. For each design problem identified by the first set of rules, a design solution is sought. The process terminates if no more design problems are found by the activation task. Solutions for all design problems that are part of the product model have been found.

With such an automated approach, a common rule set and the given ruleflow can be used to determine multiple solutions at once. Design can be automated. A manual identification

of the design problems present is not required. For design problems where the standardized solutions are not applicable, the design pattern aborts and the user is informed accordingly. Here, a custom solution needs to be designed by the engineer. As non-standard design scenarios are identified, therefore, the approach presented can also be used for quality control and the design can be changed to increase the coverage of the design standard. Series effects can thus be increased.

8.3. Knowledge-Based Design Patterns

With the multiple execution of a single design activity, i.e. by an electronic definition of a specific design task, design solutions of a given type can be determined. If multiple design tasks with differing objectives are defined such that these are executed automatically using KBE algorithms, *Knowledge-based Design Patterns* are given:

Definition 13. *Operating on a known general design problem, design patterns can be applied to define the execution of a sequence of design tasks. Therefore, for an unambiguous context and for a clearly identified design objective, workflows are used to define solution strategies.*

Put more simply, design patterns are used to define a sequence of design tasks. Using the rule-based formulations for the definition of such tasks as building blocks, see the previous section, a formalized description of a specific part of the design process can hence be achieved.

For the detailed design of the connection of a wall and a deck structure, as given as part of the test case in section 5.5.3, a design pattern as shown in figure 8.4 can be used. Here, the three design tasks of *Bracket Design*, *Cutout and Clips Design* and *Notch Design* are executed consecutively. For each design task, design elements are selected and activated as candidates, see section 8.1. For these candidates, the corresponding design activities as introduced in chapter 7 are executed and hence solutions are generated. In figure 8.5, the result after the execution of all three design tasks is shown. More detailed explanations about the result of each design task are given at the end of each respective section in chapter 7.

Figure 8.4.: Design Pattern for the Detailing of Connecting Plates

In this example, the execution order of the last two design tasks is not of importance. Alternatively, an inverse order, i.e., *Notch Design* followed by *Cutout and Clips Design* could be used. The design of brackets is defined as first design task, as thus the orientation can be freely chosen. Cutouts and clips are defined according to the context, i.e. for the placement, e.g., of a watertight collar plate, the orientation of the bracket is regarded so that no interference of these two parts occurs.

From a technical point of view, knowledge-based design patterns are defined graphically as independent ruleflow groups within the *Drools* rule engine, see [156]. Here, each design task calls the corresponding ruleflow definition for the task. Transitions between different tasks

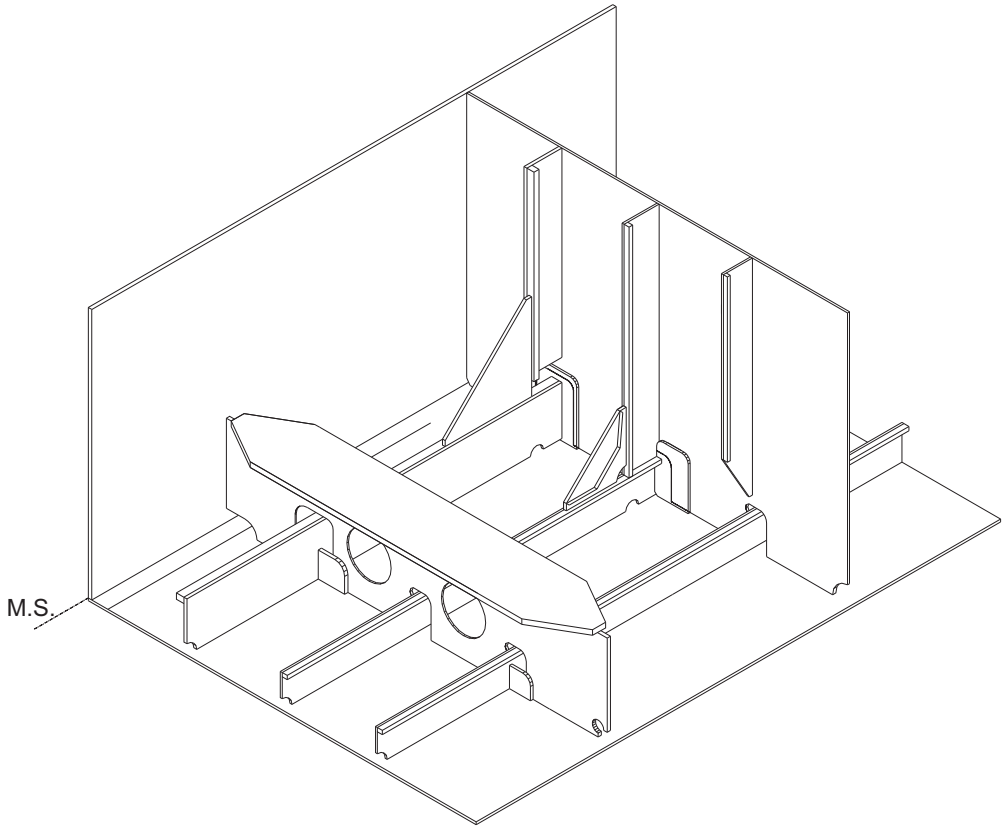


Figure 8.5.: Result of the Complete Design Pattern

are described where conditions for the transition can be defined. Following business modeling methods, complex scenarios can hence be represented. For the implementation, i.e for the realization of these concepts by a rule-based approach, rule-formulations as presented in the previous chapters are used.

While the design of a wall-deck connection is focused on a straightforward design method, a more elaborate definition of a design process can, e.g., be used for the design of the structural details of floor plates. As shown in figure 8.6, two different types of floor plates can be identified. *Full Floorplates* do not have any major openings. Holes for pipe clearance might exist, though. They are used where water- or oil-tightness is required or to provide an improved strength with respect to shear and vertical loads. *Light Floorplates* are defined with an objective of weight reduction. A large opening is defined within the floor plate.

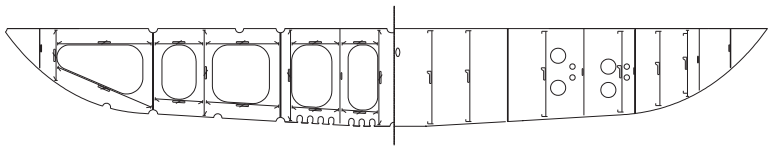


Figure 8.6.: Examples for Light (left) and Full (right) Floorplates

For the design of any floor plate, the major design tasks to be performed in detailed ship structural design as well as their order are known. Therefore, a knowledge-based design

pattern can be defined for the design of the details for such a floorplate structure as shown in figure 8.7.

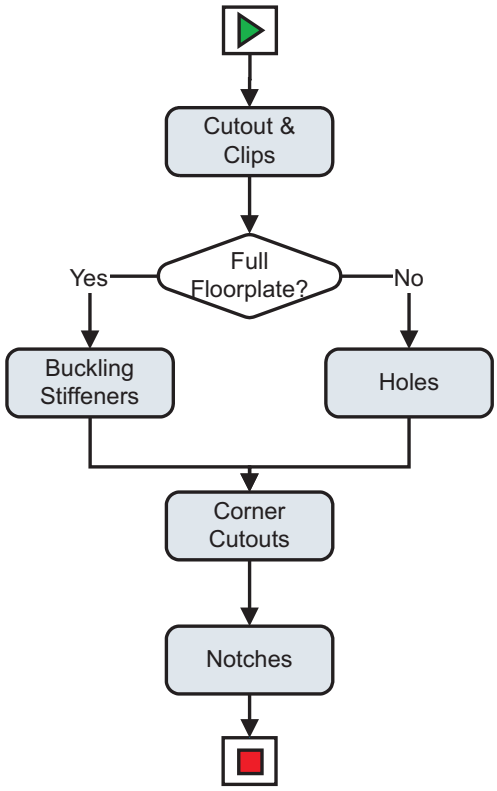


Figure 8.7.: The Design Pattern for the Design of a Floor Plate

Based on the basic steel structure, the information about strength and stress requirements as well as the knowledge about the function of the compartments bounded by the floor plates, different design tasks are given. Depending on the room configuration, either a full watertight floor plate structure or a lightweight structure with holes is defined. Also, shipyard and class society regulations need to be taken into account. E.g., a project-specific regulation that every third floor plate should be designed as full floor plate structure can be regarded.

As part of the design pattern, firstly, for each floor plate, cutouts are defined for intersecting stiffeners. For watertight tank walls, adequate collar plates are selected. The connection of the stiffeners to the plate structure is defined so that an adequate cross section for load transfer is given. Depending on the type of floorplate given, two different design activities follow. On the one hand, holes are defined for lightweight floor plate structures according to the shipyard standards. Here, rules are used to determine the distance from the edge of the hole to the edge of the plate. The corner radius is determined. On the other hand, for full floor plates, buckling stiffeners are defined on the plate, if required. Finally, for both types of floorplates, corner cutouts as well as cutouts for weld seams need to be defined. As a result, the detailed design of a floor plate structure for a standardized context can be performed automatically and rule-driven. Again, only standards-compliant cases are handled by the KBE system. In case of ambiguities, the expertise of the user is asked for.

As a result, using such design patterns, the standardized detailing of complete structures of a vessel can be performed automatically. Where possible, the design pattern generates the

relevant standardized solutions for brackets, cutouts etc. If a special situation is given and no standardized solution can be found, the user is informed accordingly. Using the generated structures as a starting point for further design work, the ship structure can be formalized or it can be adapted to issues not considered by the standard. E.g., piping might require additional openings in the floorplates etc.

9. Summary and Conclusions

The marine industry today is characterized by high requirements regarding built quality and a severe competition with respect to total cost of design and construction and - sometimes - with respect to time to delivery. In the field of detailed ship structural design, for this purpose, standardization is one means to ensure the rapid application of approved solutions of guaranteed high quality for clearly defined design problems.

Based on these general conditions, the work performed as part of this thesis was governed by the question, how an eased and improved application of design standards in the domain of detailed ship structural design could be achieved, see the research questions postulated as part of the introduction in chapter 1. For this purpose, the application of advanced knowledge-based algorithms for the automatic application of such design standards has been investigated. Taking an interdisciplinary approach, concepts from the domains of knowledge, logic and computer science are called upon. The major concepts and assumptions from these three fields are given as part of the theoretical groundwork, see chapters 2 and 3.

Based on a definition of knowledge that is compatible with the domain of engineering, core concepts regarding *Knowledge Representations* as needed in the following chapters are introduced and methods for an IT-based representation of said knowledge are discussed. The notion of *Applied Problem Solving* as a means of knowledge application is introduced. Here, three different approaches are presented, namely *Heuristic-based Modeling*, *Case-based Reasoning* and *Rule-based Systems* where only the latter can guarantee reproducibility and tractability, two key requirements for the automatic application of design standards in detailed ship structural design. With the introduction of the so-called *Configuration Design*, the objective of the thesis is further refined in so far as a concentration on the use of rule-based systems for the application of a fixed set of standardized parts to given problems is taken.

Taking an engineering perspective, design as a constructive and sometimes creative design task is defined. The notion of a *Design Activity* as a single so-called atomic design operation is given as a major concept used throughout the following work. Similar or identical design activities can be grouped into so-called *Design Tasks* where each individual design activity is determined by the *Design Objective* and operates in a specific *Design Context*.

With these rather theoretical basics in mind, a short summary of the design procedure as applied in ship design is used to introduce different views on the steel structure of a vessel. Also, a classification of ship structural elements based on geometric and functional properties is given and the role of standardization is characterized. Using three test cases, major characteristics of knowledge-based design in detailed ship structural design are captured, see chapter 5. Here, quantitative and qualitative evaluations are used and hence realistic, yet suitable cases are selected.

With the help of these test cases, requirements for an IT-based infrastructure for the automatic or semi-automatic application of knowledge-based algorithms in the domain of de-

tailed design are developed in chapter 6. Using the concepts introduced in the previous chapters, the rule-based system of choice *Drools* is introduced and its capabilities with respect to knowledge-based engineering algorithms are evaluated. Also, the integration of such a system into the complete design process is discussed; three different approaches for the interaction with *Tribon* as selected CAD-system are developed. Different means for a suitable knowledge representation are discussed. The core structure of the ontology-based knowledge models are introduced.

As all these approaches do not offer suitable means to access or manipulate the topology of the product data model, the focus of the work needs to be set on the development of KBE algorithms only. Hence, only preliminary solutions for the handling of topological aspects are used where manual intervention is required. Therefore, for an automatic interaction of KBE and CAD systems, a different, more advanced approach needs to be found. For this purpose, the application of a so-called *Wrapper* is discussed, yet, further development is needed to find suitable formulations. Here, a compromise of flexibility and complexity of the interfaces needs to be found. This can be problem dependent, so that a generic solution might not be an advisable option.

Also, the role of additional information not present in the CAD-system is evaluated. With KBE seen as the integration of information from different data sources, the handling of the so-called meta-information, i.e. of information not directly related to shape, topology and material, is identified as one important challenge for a successful application of IT-based design algorithms. In addition to the data models for the representation of design standards, a preliminary approach for the system-independent storage of this kind of information is developed. Yet, as the allocation of meta-data to objects in the data-model imported is currently based on an auto-generated ID, a full round-trip support is not guaranteed for certain cases. Also, the expressibility of the approach chosen is limited.

For the identified test cases as atomic design problems, context-sensitive rule-based solutions are developed as part of chapter 7 where different methods are introduced for each test case.

For the simple case of bracket design, a direct selection process is introduced, where rules are formulated for the given requirements or constraints. Being a suitable approach for the handling of hard constraints, with this so-called culling process requirements can be handled individually and, if violated, lead to the exclusion of design standards from any further processing steps.

In contrast, for the test case of notch design a two-step approach is chosen. Here, in a first step, hard constraints as formulated by the design context are extracted and analyzed. With this context being known, so-called candidate solutions are fetched from an external database of permissible standardized design details in a second step and are then evaluated with respect to additional constraints. With this approach, a design activity can be split into a sequence of individual actions. Hence maintenance and the general understanding of the solution logic can be improved.

Finally, for complex parts that are made up of features and parts, the test case of collar and cutout design serves as example. For an improved readability of the chosen approach, here, a multi-step approach is developed by means of agenda groups. Rule partitioning is applied. In addition to the known handling of hard constraints, an approach for the handling of qualitative (soft) constraints is introduced so that a measure for the overall design quality of a final solution can be calculated. For this purpose, a class of weight factors is introduced

into the decision process. As a result, for the set of feasible design solutions a ranking of these solutions is given such that an optimal solution with respect to given design criteria can be determined automatically.

For the identification of quality criteria and for the determination of the corresponding weight factors, a thorough understanding of the design process is required. As multiple objectives are compiled into a single number designating the overall quality of the result, a compromise regarding cost, build quality, manufacturability etc. needs to be found. While this primarily depends on the engineer who is responsible for the development or maintenance of the KBE system, test cases can be used to explore the suitability of the chosen weight and quality definitions.

Taking the rule-based formulations of atomic design activities as building blocks, the knowledge-based engineering approach is extended in chapter 8. With the introduction of the so-called *Design Patterns*, the process of standards-compliant detailing can be executed fully automatically where the system uses a given workflow of design tasks like bracket design or notch design. For each design task, all relevant design problems are identified, i.e. the locations a design activity needs to be carried out are determined and the given design activity is executed. As a result, based on a preliminary design, a fully detailed design solution as shown in figure 8.5 on page 119 can be achieved automatically.

Rule-based systems offer a means to increase automatization and standards compliance within the design stage of detailed ship structural design. Hence, the workload of the engineer is eased such that more time can be spent on innovative work. Also, series effects can be explored and the design quality can be improved. The use of solutions with optimal performance can be achieved. In case of violations of the design standards, the user is informed accordingly. Therefore, this approach can not only be applied with the objective of a reduction of design time needed but can also be used for quality control.

With the application of a rule-based approach, the development of IT-based design algorithms does not require actual programming skills but is performed with the rule dialect presented. Hence, development complexity is reduced as only the logics describing the design standards and not a complete software architecture needs to be formulated. Therefore, rule-development might be performed by trained engineers, i.e. by the personnel which is directly involved in the design process and hence most familiar with the design problems and design standards. Also, the reuse of individual design activities at different levels of complexity is feasible due to the layered, bottom-up approach chosen. Hence, depending on the given design problem a suitable level of automation can be chosen.

Yet, knowledge acquisition and knowledge explication are difficult; often, existing yard standards or class regulations are not formulated clearly. Dependencies on the context are not always given. Therefore, a formulation of a standard that is suitable for the implementation of a rule-based system and which can be applied to a sufficiently large number of cases can be difficult to obtain. Also the transformation of this formulation into the corresponding rule-based representation can be further complicated as a naval engineer may lack the relevant understanding of such rule-based systems. The unambiguous definition of the design activities has proven to be a problem in real life, therefore. An adaptation of existing knowledge-based engineering standards to changing requirements on the shipyard is hence cost and time intensive so that the management of these electronic formulations is an important issue to consider.

For a successful application of knowledge-based design algorithms, a clearly defined design context is required next to sufficient numbers of occurrences of this standard. KBE algorithms therefore are only applicable for common design problems where the context can be clearly identified. In case of special purpose vessels, a major type of vessel to be built on German shipyards in the coming years, this is often not the case, as the steel structure is primarily not governed by strength or structural requirements but also by requirements mandated by equipment and accommodation design. In this case, the efficient use of automatization tools needs to be seen critically. This is in particular true for more complex approaches like the fully automatic execution of design patterns.

Finally, the use of the generated results within a naval CAD system has not been considered in detail as part of this thesis. Here, it is assumed that the presented rules are manually migrated to the CAD system used. For a successful application, though, a transparent and fully automatic solution for import and export of data is required. As the integration with existing CAD systems is complex and time intensive today, this is a significant task. Also, often different data sources like a global strength model, compartmentation plans etc. are not connected or are only available as plain drawings which are not analyzable electronically so that the common access to these diverse data sources is also required.

With a more integrative approach, an improved application of knowledge inherent in these information sources throughout the complete design process could be achieved and the application of knowledge-based design algorithms could be eased. Changes could hence automatically be propagated throughout the product model. As a result, the KBE algorithms presented could also be applied to automatically reevaluate all design problems with changing requirements or with a changed design context.

Bibliography

[1] E. S. C. Weiner, J. A. Simpson, and Oxford University Press., eds. *The Oxford English dictionary*. English. 2nd ed. / prepared by J.A. Simpson and E.S.C. Weiner. Clarendon Press; New York : Oxford : Oxford ; Oxford University Press, 1989, pp. 20 v. ; ISBN: 0198611862.

[2] Auguste Comte. *Rede uber den Geist des Positivismus*. Meiner, Hamburg, 1994.

[3] Rolf Franken and Andreas Gadatsch. *Integriertes Knowledge Management : Konzepte, Methoden,Instrumente und Fallbeispiele*. 1. Aufl. Braunschweig [u.a.]: Vieweg, 2002. ISBN: 3-528-05779-3.

[4] Gerold Riempp. *Integrierte Wissensmanagement-Systeme : Architektur und praktische Anwendung*. Springer, 2004. ISBN: 3-540-20495-4.

[5] J Sowa. "The challenge of knowledge soup". In: *Research Trends in Science, Technology and Mathematics Education* 1 (2004), p. 15.

[6] Charles Peirce. "A System of Logic, Considered as Semiotic." 1902.

[7] G van Heijst, R van der Spek, and E Kruizinga. "Organizing Corporate Memories". In: *KAW'96, Special Track on Corporate Memory and Enterprise Modeling*. 1996.

[8] E. Motta. *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 1999. ISBN: 1586030035.

[9] R. M. Grant. "Toward a knowledge-based theory of the firm". In: *Strategic management journal* 17.WINTER (1996), pp. 109–122.

[10] G. von Krogh, I. Nonaka, and M. Aben. "Making the Most of Your Company's Knowledge: A Strategic Framework". In: *Long Range Planning* 34.4 (2001), pp. 421–439.

[11] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, May 2004. ISBN: 1558609326.

[12] R. Davis, H. Shrobe, and P. Szolovits. "What is a Knowledge Representation?" In: *AI Magazine* 14.1 (1993), pp. 17–33.

[13] Ian Horrocks and Ulrike Sattler. *Description Logics: Basics, Applications, and More*. ECAI-2002. `http://www.cs.man.ac.uk/~horrocks/Slides/IJCAR-tutorial/Print`. 2002.

[14] David Scuse. *Jess, The Rule Engine for the Java Platform*. `http://www.jessrules.com/jess/docs/Jess71p2.pdf`. 2008.

[15] Daniele Nardi and Ronald J. Brachman. "An introduction to description logics". In: New York, NY, USA: Cambridge University Press, 2003, pp. 1–40. ISBN: 0-521-78176-0.

[16] James G. Schmolze, Bolt Beranek, and Newman Inc. “An overview of the KL-ONE knowledge representation system”. In: *Cognitive Science* 9 (1985), pp. 171–216.

[17] *Loom Project (Website)*. http://www.isi.edu/isd/LOOM/.

[18] D. J. Power. *A Brief History of Decision Support Systems*. DSS Resources. http://dssresources.com/history/dsshistory.html. 2007.

[19] *Extensible Markup Language (W3C Recommendation)*. W3C. http://www.w3.org/XML.

[20] *XML Schema (W3C Recommendation)*. W3C. http://www.w3.org/XML/Schema.

[21] *World Wide Web Consortium (Website)*. W3C. http://www.w3c.org/.

[22] Li Ding, Pranam Kolari, Zhongli Ding, and Sasikanth Avancha. “Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems”. In: *Ontologies in the Context of Information Systems*. Springer, Oct. 2007. Chap. Using Ontologies in the Semantic Web: A Survey, pp. 79–113.

[23] *Resource Description Framework (W3C Recommendation)*. W3C. http://www.w3.org/RDF/.

[24] *RDF Vocabulary Description Language 1.0: RDF Schema (W3C Recommendation)*. W3C. http://www.w3.org/TR/rdf-schema/.

[25] Shelley Powers. *Practical RDF*. O’Reilly, Aug. 2003. ISBN: 0596002637.

[26] *OWL Web Ontology Language (W3C Recommendation)*. W3C. http://www.w3.org/TR/owl-semantics/.

[27] Dieter Fensel. *Ontologies: a silver bullet for knowledge management and electronic commerce*. 2nd. Springer, 2003.

[28] Michael C Daconta, Leo J Obrst, and Kevin T Smith. *The Semantic Web: A guide to the future of XML, Web Services and Knowledge Management*. Wiley, 2003.

[29] *The Rule Markup Initiative (Website)*. http://www.ruleml.org/.

[30] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML (W3C Member Submission)*. W3C. http://www.w3.org/Submission/SWRL/.

[31] Heiko Kattenstroth. “Knowledge-Management with OWL and F-Logic: A Combination of Description Logic Reasoning with F-Logic Rules”. MA thesis. Universität Göttingen, 2007.

[32] Martin O’connor, Holger Knublauch, Samson Tu, and Mark Musen. “Writing Rules for the Semantic Web Using SWRL and Jess”. In: *8th International Protege Conference, Protege with Rules Workshop*. 2005.

[33] Grigoris Antoniou, Carlos V. Damásio, et al. *Combining Rules and Ontologies: A survey*. Rewerse. deliverable. http://idefix.pms.ifi.lmu.de:8080/rewerse/index.html\#REWERSE-DEL-2005-I3-D3. 2005.

[34] Ullrich Hustadt. “Do we need the closed-world assumption in knowledge representation”. In: *Working Notes of the KI’94 Workshop: Reasoning about Structured Objects: Knowledge Representation Meets Databases (KRDB’94)*. Ed. by Franz Baader, Martin Buchheit, Manfred A. Jeusfeld, and Werner Nutt. Vol. D-94-11. Document. DFKI, Nov. 1994, pp. 24–26.

[35] Nick Drummond and Rob Shearer. *The Open World Assumption*. University of Manchester. http://www.cs.man.ac.uk/~drummond/presentations/OWA.pdf. 2006.

[36] *SFI Group System*. https://www.xantic.net/internet/files/products/amos/sfi/supportdocuments/Product\%20Description.pdf. 2001.

[37] *Das SFI-System*. Maritime Cooperation Network. http://mariconet.de/ca/fd/diw/.

[38] J. Fowler. *STEP for Data Management, Exchange and Sharing*. Technology Appraisals Ltd., UK. ISBN 1-871802-36-9, 1995.

[39] *ISO TC 184/SC 4/WG 3/T 23 (Website)*. International Organisation of Standardization. http://www.nsrp.org/t23/.

[40] David Loffredo. *Fundamentals of STEP Implementation*. www.steptools.com/library/fundimpl.pdf. 1999.

[41] Allison Barnard Feeney. "The STEP Modular Architecture". In: *Journal of Computing and Information Science in Engineering* 2.2 (2002), pp. 132–135. DOI: 10.1115/1.1511520.

[42] ISO10303. *International automation systems and integration - Product data representation and exchange*. Tech. rep. Genf: International Organisation of Standardization, 1994.

[43] Martin Hardwick. *The STEP Standard*. http://www.steptools.com/impforum/info.html.

[44] Lother Klein. *Critcism of the STEP Application Module Approach*. Mailing List. http://lists.steptools.com/pipermail/wg11-owl/2005/000012.html. 2005.

[45] Nicola Guarino, Stefano Borgo, and Claudio Masolo. "Logical modelling of product knowledge: Towards a well-founded semantics for step". In: *Proceedings of European Conference on Product Data Technology (PDT Days 97). Sophia Antipolis*. 1997, pp. 183–190.

[46] *STEP Application Handbook*. SCRA. http://www.uspro.org/documents/STEP_application_hdbk_63006_BF.pdf. 2006.

[47] David Leal and Lothar Klein. *Methodology for the derivation of ontologies from ISO 10303*. Tech. rep. S-TEN, 2007.

[48] Lothar Klein. *Linking STEP with OWL - Results of the Technology Validation*. Tech. rep. S-TEN, 2007.

[49] Mathias Grau. *Software Architectures for Ship Product Data Integration and Exchange*. The SeaSpriter Consortium. 1999.

[50] *Integrated Shipbuilding Environment*. National Shipbuilding Research Program. http://www.nsrp.org/Project_Information/major_projects/deliverable_pages/ise_deliverables.html.

[51] Ted Briggs, Steffen Baum, and Tammi Thomas. "Interoperability Framework". In: *2004 Ship Production Symposium, Washington*. 2004.

[52] Ted Briggs and Thomas Rando. "XML Schemas for Shipbuilding". In: *11th ICCAS*. 2002.

[53] *PLIB - International Industrial Data Standard (Website)*. International Standardization Organization. http://www.plib.ensma.fr/.

[54] Hercules Dalianis, Hercules Dalianis, and Eduard Hovy. "Integrating STEP Schemata using Automatic Methods". In: *in Proceedings of the ECAI-98 Workshop on Applications of Ontologies and Problem-Solving Methods*. 1998, pp. 54–66.

[55] Guy Pierra. "Context-Explicitation in Conceptual Ontologies: PLIB Ontologies and their Use for Industrial Data". In: *Journal of Advanced Manufacturing Systems* 5 (2006), pp. 243–254.

[56] Guy Pierra, Yamine Ait-Ameur, et al. "A General Framework for Parametric Product Models within STEP and Parts Library". In: *PDTAG Product Data Technology Days '96*. 1996.

[57] G Pierra, E Sardet, et al. "Exchange of component data : The PLIB model, standard and tools". In: *In CALS Europe '98*. 1998, pp. 160–176.

[58] Jérôme Chochon, Yamine Aït Ameur, Guy Pierra, and Jean-Claude Potier. "Reducing parts diversity in product design: a data centered approach". In: *ISPE CE*. Ed. by Ricardo Jardim-Gonçalves, Jianzhong Cha, and Adolfo Steiger-Garção. A. A. Balkema Publishers, 2003, pp. 311–318. ISBN: 90-5809-623-8.

[59] Douglas A. Schenk and Peter R. Wilson. *Information Modelling: The EXPRESS way*. Oxford University Press, 1994.

[60] David Price. *A Brief Foray into Semantic Web Technology and STEP*. Engineering eXchange For Free. http://www.exff.org/exff_legacy/docs/semweb_step.html. 2003.

[61] B. Chandrasekaran. "Design problem solving: a task analysis". In: *AI Mag.* 11.4 (1990), pp. 59–71. ISSN: 0738-4602.

[62] Enrico Motta and Zdenek Zdrahal. "A library of problem-solving components based on the integration of the search paradigm with task and method ontologies". In: *Int. J. Hum.-Comput. Stud.* 49.4 (1998), pp. 437–470. ISSN: 1071-5819. DOI: http://dx.doi.org/10.1006/ijhc.1998.0214.

[63] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme : Grundlagen, Algorithmen,Anwendungen ; [mit Online-Service zum Buch]*. 2., überarb. u. erw. Aufl. Braunschweig [u.a.]: Vieweg, 2000. ISBN: 3-528-15723-2.

[64] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, Sept. 2000. ISBN: 1558604898.

[65] T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), Oct. 1997. ISBN: 0071154671.

[66] Jarrod Moss, Jonathan Cagan, and Kenneth Kotovsky. "Learning from Design Experience in an Agent-Based Design System". In: *Research in Engineering Design* 15.2 (2005), pp. 77–92. DOI: 10.1007/s00163-004-0042-4.

[67] J.L. Metzger, F. Le Ber, and A. Napoli. "Using DL for a Case-Based Explanation System". In: *2002 International Workshop on Description Logics*. 2002, pp. 203–210.

[68] R Barták. "Constraint programming: In pursuit of the holy grail". In: *Proceedings of WDS99*. 1999, pp. 555–564.

[69] Roman Barták. *Online Guide to Constraint Programming*. http://kti.mff.cuni.cz/~bartak/constraints. 1998.

[70] Slim Abdennadher. "Rule-Based Constraint Programming: Theory and Practice". PhD thesis. Ludwig-Maximilians-Universität München, 2001.

[71] Silvie Spreeuwenberg and Rik Gerrits. "Business Rules in the Semantic Web, Are There Any or Are They Different?" In: *Reasoning Web*. 2006, pp. 152–163.

[72] Ian H. Witten and Eibe Frank. *Data mining : practical machine learning tools and techniques*. 2. ed. Amsterdam [u.a.]: Elsevier, Morgan Kaufman, 2005. ISBN: 0-12-088407-0.

[73] T Bilgic and M S Fox. "Constraint-Based Retrieval of Engineering Design Cases: Context as Constraints". In: *Stanford University*. Kluwer Academic Publishers, 1996.

[74] Frank Puppe. *Systematic Introduction to Expert Systems: Knowledge Representations and Problem Solving Methods*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1993. ISBN: 0387562559.

[75] N Pena, E Garcia, and J M Lazaro. *Configuration Ontology & Multi-product Configuration Tool (I)*. OBELIX. http://www.cs.vu.nl/~obelix/. 2003.

[76] B J Wielinga and A Th Schreiber. "Configuration design problem solving". In: *Special issue on AI and design*. 1997, pp. 49–56.

[77] Ulrich Sendler. *Das PLM Kompendium*. Springer, 2009.

[78] Mihai Ciocoiu, Dana S. Nau, and Michael Gruninger. "Ontologies for integrating engineering applications". In: *J. Comput. Info. Sci. Eng.* 1.1 (2001), pp. 12–22. ISSN: 1530-9827. DOI: http://dx.doi.org/10.1115/1.1344878.

[79] G. Huang and J. Brandon. "Agents for cooperating expert systems in concurrent engineering design". In: *Artificial intelligence for engineering design, analysis and manufacturing* 7 (1993), pp. 145–158.

[80] Sehyun Myung and Soonhung Han. "Knowledge-based parametric design of mechanical products based on configuration design method". In: *Expert Systems with Applications* 21.2 (2001), pp. 99–107.

[81] V Hubka and W Eder and. *Design Science*. Springer, 1993.

[82] Siang Kok Sim and Alex H B Duffy. "Towards an Ontology of Generic Design Activities". In: *Research in Engineering Design* 14.4 (2003), pp. 200–223. DOI: 10.1007/s00163-003-0037-1.

[83] SM Kannapan and KM Marshek. "Mechanical Design: Theory & Methodology". In: Springer, 1996. Chap. A comparative analysis of techniques in engineering, pp. 209–235.

[84] Nick Milton. *Knowledge Technologies*. Vol. 3. Publishing Studies. POLIMETRICA, 2008.

[85] Peter F. Tropschuh. *Rechnerunterstützung für das Projektieren mit Hilfe wissensbasierter Systeme*. Hanser, 1989.

[86] *CIS/2 - CIMsteel Integration Standards (Website)*. Computer Integrated Manufacturing of Constructional Steelwork (CIMsteel). http://www.cis2.org/.

[87] *CIS/2 and IFC - Product Data Standards for Structural Steel (Website)*. Natioanl Institute of Standards and Technology (NIST). http://cic.nist.gov/vrml/cis2.html.

[88] Carlos Toro, Jorge Posada, et al. "Knowledge Based Tools to Support the Structural Design Process." In: *Knowledge-Based Intelligent Information and Engineering Systems*. Ed. by Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain. Vol. 4251. Lecture Notes in Computer Science. Springer, Nov. 15, 2006, pp. 679–686. ISBN: 3-540-46535-9.

[89] Jorge Posada, Carlos Toro, Stefan Wundrak, and Andre Stork. "Using ontologies and STEP standards for the semantic simplification of CAD models in different engineering domains". In: *Applied Ontology* 1.3-4 (2006), pp. 263–279.

[90] Silvia Ansaldi, Paolo Bragatto, et al. "A Knowledge-based Tool for Risk Prevention on Pressure Equipments". In: *Computer-Aided Design & Applications* 3 (2006), pp. 99–108.

[91] G La Rocca, LA Krakers, and van Tooren. "Development of an ICAD Generative Model for Aircraft Design, Analysis and Optimisation". In: *International ICAD User Group Conference*. 2002, pp. 1–23.

[92] H. Ohtsubo and Y. Sumi, eds. *Proceedings of the 14th International Ship and Offshore Structure Congress*. Vol. 1. Elsevier, 2000.

[93] Beom-Seon Jang, Benedikte Kallak, et al. "A process-centric ship design management framework". In: *Journal of Marine Science and Technology* 15.1 (2010), pp. 23–33.

[94] Stefan Krüger. "The Role of IT in Shipbuilding". In: *Hansa* 141 (2004), pp. 46–49.

[95] Jan Nieuwenhuis and Ubald Nienhuis. "Knowledge and Data Reuse in Ship Design and Engineering". In: *3rd International Conference on Computer Applications and Information Technology in the Maritime Industries*. 2004.

[96] J.H. Park and R.L Storch. "Overview of ship-design expert systems". In: *Expert Systems* 19 (2002), pp. 136–141.

[97] *Aveva Tribon.com (Website)*. Aveva AB. http://www.tribon.com/.

[98] *Siemens PLM Software (Website)*. http://www.plm.automation.siemens.com/.

[99] *Intergraph Shipbuilding (Website)*. http://www.intergraph.com/shipbuilding/default.aspx.

[100] Reidar Tronstad and Antonio Roodrigues. "Automation Tools in the Design Process". In: *3rd International Conference on Computer Applications and Information Technology in the Maritime Industries*. 2004.

[101] M Toyoda and Y Nakajima. "Competitive Conceptual Structural Design System Using 3D PLM Technology". In: *Ishikawajima Harima Engineering Review* 4 (2005), pp. 152–157.

[102] S. Kang and S. Han. "A design expert system for autorouting of ship pipes". In: *Journal of Ship Production* 15 (1999), pp. 1–9.

[103] C. Kuo, J. Wu, and H. Shaw. “Collision avoidance schemes for orthogonal pipe routing”. In: *Journal of Ship Production* 15 (1999), pp. 198–206.

[104] A. Asmara and U. Nienhuis. “Automatic Piping System Implementation: A Real Case”. In: *6th International Conference on Computer Applications and Information Technology in the Maritime Industries*. 2007.

[105] *Qnowledge : Modelling Technologies (Website)*. http://www.qnowledge.nl/.

[106] *Maritime Research Institute Netherlands (Website)*. http://www.marin.nl/.

[107] O. Kienzle. *Normung und Wissenschaft*. Schweiz. Techn. Z., 1943.

[108] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote. *Konstruktionslehre - Grundlagen erfolgreicher Produktentwicklung. Methoden und Anwendung*. Springer, 2007.

[109] Jean-Peer Lorenz. “Analyse, Bewertung und Klassifizierung von schiffbaulichen Strukturelementen als Basis für eine teilautomatisierte geometrische Erzeugung dieser Strukturen innerhalb CAD-Systemen”. MA thesis. Fachhochschule Kiel, 2007.

[110] Dan Braha and Yoram Reich. “Topological Structure for Modelling Engineering Design Processes”. In: *Research in Engineering Design* 14 (2003), pp. 185–199.

[111] Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. “Workflow Data Patterns: Identification, Representation and Tool Support”. In: *24th International Conference on Conceptual Modeling*. 2005, pp. 353–368.

[112] Eike Lehmann. *Grundzüge des Schiffbaus - Band II*. 2005.

[113] Wolfgang Fricke. “Vorlesungsskript Schiffskonstruktion I”. Technische Universität Hamburg-Harburg. 2006.

[114] Alaa Mansour and Donald Liu. *Principles of Naval Architecture: Strength of Ships and Ocean Structures*. SNAME. 2008.

[115] Wolfgang Fricke. “Grundlagen der Schiffskonstruktion”. Technische Universität Hamburg-Harburg. 2006.

[116] Eike Lehmann. *Grundzüge des Schiffbaus - Band I*. 2002.

[117] Stefan Hoffmann. “Vereinfachter Einsatz von Knieblechen im CAD-System TRIBON”. MA thesis. Universität Rostock, 2005.

[118] Robert Bronsart, Andreas Geitmann, et al. *Konstruktionsstandards für schiffbauliche Strukturen zum Einsatz in CAD-Systemen - Abschlussbericht*. Tech. rep. Universität Rostock, 2007.

[119] *Aveva (Website)*. Aveva Group Plc. http://www.aveva.com.

[120] Thom Frühwirth and Slim Abdennadher. *Essentials of Constraint Programming*. Springer, 2003. ISBN: 3540676236.

[121] Thom Frühwirth and Slim Abdennadher. *Constraint-Programmierung*. Springer, 1997.

[122] *The Java Business Rules Community (Website)*. http://www.javarules.org/.

[123] *JBoss Drools (Website)*. JBoss Inc. http://www.jboss.org/drools/.

[124] *The Workflow Reference Model*. Workflow Management Coalition. http://www.wfmc.org/standards/docs/tc003v11.pdf. 1995.

[125] Van Der Aalst, L. Aldred, M. Dumas, and A. H. M. Ter Hofstede. "Design and implementation of the YAWL system". In: *Proceedings of CAiSE*. 2004.

[126] Claudia Sommer. "MoKon - Ein Ansatz zur wissensbasierten Konfiguration von Variantenerzeugnissen". PhD thesis. Universität Erlangen-Nürnberg, 1992.

[127] Michael Grossniklaus and Moira C. Norrie. "An Object-Oriented Version Model for Context-Aware Data Management". In: *WISE*. 2007, pp. 398–409.

[128] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, Apr. 2004. ISBN: 0262012103.

[129] Yannis Kalfoglou and Marco Schorlemmer. "Ontology Mapping: The State of the Art". In: *Semantic Interoperability and Integration*. Ed. by Y. Kalfoglou, M. Schorlemmer, et al. Dagstuhl Seminar Proceedings 04391. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[130] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and Complexity of SPARQL". In: *International Semantic Web Conference*. 2006.

[131] *Jena – A Semantic Web Framework for Java (Website)*. `http://jena.sourceforge.net/`.

[132] *Pellet: The Open Source OWL DL Reasoner (Website)*. clark & parsia. `http://clarkparsia.com/pellet/`.

[133] Aditya Kalyanpur, Daniel J. Pastor, Steve Battle, and Julian A. Padget. "Automatic Mapping of OWL Ontologies into Java". In: *Proceedings of the 16th Int'l Conference on Software Engineering & Knowledge Engineering (SEKE'2004)*. Ed. by Frank Maurer and Günther Ruhe. Banff, Alberta, Canada, June 2004, pp. 98–103.

[134] *An Introduction to RDF and the Jena RDF API*. `http://jena.sourceforge.net/tutorial/RDF_API/index.html`.

[135] *Db4O: Native Java & .NET Open Source Object Database (Website)*. Versant Corporation. `http://www.db4o.com`.

[136] *Data Models vs. Ontologies (Thread)*. Ontolog-Forum Mailing-List. May 2008.

[137] *Context - Classes vs. Instances (Thread)*. Ontolog-Forum Mailing-List. `http://suo.ieee.org/email/msg13345.html`. Apr. 2006.

[138] *Thing and Classes*. Ontolog-Forum Mailing-List. `http://ontolog.cim3.net/forum/ontolog-forum/2008-09`. Sept. 2008.

[139] *Topbraid Composer Getting Startd Guide*. Topquadrant Website. `http://topquadrant.com/docs/marcom/TBC-Getting-Started-Guide.pdf`.

[140] *Atlantec Enterpise Solutions (Website)*. Atlantec ES. `http://www.atlantec-es.com/`.

[141] Yoshinobu Kitamura and Riichiro Mizoguchi. "Ontology-Based Systematization of Functional Knowledge". In: *Journal of Engineering Design* 15.04 (2004), pp. 327–351. ISSN: 1466-1387.

[142] Yumi Iwasaki, Richard Fikes, Marcos Vescovi, and B. Chandrasekaran. "How Things are Intended to Work: Capturing Functional Knowledge in Device Design". In: *IJCAI*. 1993, pp. 1516–1522.

[143] Martin Schönhoff, Markus Strässler, and Klaus R. Dittrich. "Data Integration in Engineering Environments". In: *EFDBS*. 1997, pp. 45–56.

[144] *SPARQL Query Language for RDF (W3C Recommendation)*. W3C. http://www.w3.org/TR/rdf-sparql-query/.

[145] Holger Knublauch. *Ontology Mapping with SPARQL CONSTRUCT*. Composing the Semantic Web (Blog). http://composing-the-semantic-web.blogspot.com/2006/09/ontology-mapping-with-sparql-construct.html. 2006.

[146] *Smooks Data Integration (Website)*. http://www.smooks.org/mediawiki/index.php?title=Main_Page.

[147] *QualiSHIP - Produktivitätssteigerung durch Qualitätssicherung schiffstechnischer Produktdaten (Website)*. Universität Rostock. http://www.qualiship.de.

[148] *Core J2EE Patterns - Data Access Objects*. Sun Microsystems. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

[149] *MVEL Expression Language*. http://mvel.codehaus.org/.

[150] Holger Knublauch. *Composite Design Pattern in RDF/OWL*. Composing the Semantic Web (Blog). http://composing-the-semantic-web.blogspot.com/2007/07/composite-design-pattern-in-rdfowl.html. 2007.

[151] V.C. Liang and C.J.J. Paredis. "A port ontology for conceptual design of systems". In: *Journal of Computing and Information Science in Engineering v4 i3. 206-217.* 4 (2004), pp. 206–217.

[152] Ina Brenner. *Eine kurze Einführung in db4o*. Versant Corporation. http://www.theserverside.de/eine-kurze-einfuhrung-in-db4o/. 2007.

[153] *Java Generics*. Sun Microsystems. http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html.

[154] *Smooks User Guide*. Milyn Website. http://docs.codehaus.org/display/MILYN/Smooks+User+Guide.

[155] J. Lieber. "Strong, Fuzzy and Smooth Hierarchical Classification for Case-Based Problem Solving". In: *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*. Ed. by F. van Harmelen. IOS Press, Amsterdam, 2002.

[156] *RuleFlow*. Drools Blog. http://blog.athico.com/2007/03/ruleflow.html.

[157] Kees van Es and Martin van Hees. "Application of Knowledge Management in Conceptual Naval Design". In: *3rd International Conference on Computer Applications and Information Technology in the Maritime Industries*. 2003.

[158] S. Ahmed, K. Wallace, and L.T.M. Blessing. "Understanding the Differences between How Novice and Experienced Designers Approach Design Tasks". In: *Research in Engineering Design* 14.1 (2003), pp. 1–11. DOI: 10.1007/s00163-002-0023-z.

[159] Ander Altuna, SOnia Bilbao, et al. *Obelix - Multi-product Configuration Tool*. Tech. rep. LABEIN, 2004.

[160] Paolo Bouquet, Fausto Giunchiglia, et al. "C-OWL: Contextualizing ontologies". In: *International Semantic Web Conference 2003*. 2003.

[161] Robert Bronsart, Sebastian Bade, et al. “Knowledge Based Quality Checking of Ship Product Data”. In: *13th International Conference on Computer Applications in Shipbuilding (ICCAS)*. 2007.

[162] Stéphane Demri and Ewa Orłowska. *Incomplete Information: Structure, Inference, Complexity*. EATCS Monographs. Springer, 2002.

[163] Steffen Gau. “Ein Informationsmodell für die kollaborative schiffbauliche Produktentwicklung”. PhD thesis. Universität Rostock, 2005.

[164] S. R. Gorti, A. Gupta, et al. “An Object-Oriented Representation for Product and Design Processes”. In: *Computer-Aided Design* 30 (1997), pp. 489–501.

[165] Imre Horvath, Jeroen Pulles, Aad Bremer, and Joris Vergeest. “Towards an ontology-based definition of design features”. In: *SIAM workshop on mathematical foundations for features in computer aided design, engineering, and manufacturing*. 1998.

[166] Matthias Jörg. “Die semantische Auswertung von Produktanforderungen mit Hilfe von GermaNet”. In: *LDV Forum* 19.1/2 (2004), pp. 31–41.

[167] A. Katzenbach. “Feature-basierte Produktentwicklung - eine Form der Wissensverarbeitung”. In: *2. Forum Wissensbasierte Konstruktion und Data Management*. 2002.

[168] Y. Kitamura, Mizoguchi, Y. Kitamura, and R Mizoguchi. “Ontology-based description of functional design knowledge and its use in a functional way server”. In: *Expert Systems with Application* 24(2) (2003), pp. 153–166.

[169] Jaehyun Lee and Hyowon Suh. “Ontology-Based Multi-level Knowledge Framework for a Knowledge Management System for Discrete-Product Development”. In: *International Journal of CAD/CAM,* 5 (2005), pp. 99–109.

[170] Jinxin Lin, Mark S. Fox, and Taner Bilgic. “A Requirement Ontology for Engineering Design”. In: *Concurrent Engineering: Research and Applications* 4 (1996), pp. 279–291.

[171] Haoyang Liu and T. Igusa. “Feature-based classifiers for design optimization”. In: *Research in Engineering Design* 17.4 (Mar. 2007), pp. 189–206. ISSN: 0934-9839. DOI: 10.1007/s00163-006-0024-4.

[172] Oleg Lukibanov. “Use of Ontologies to Support Design Activities at DaimlerChrysler”. In: *8th Intl. Protégé Conference*. 2005.

[173] Riichiro Mizoguchi and Yoshinobu Kitamura. “Foundation of Knowledge Systematization: Role of Ontological Engineering”. In: *Industrial Knowledge Management - A Micro Level Approach, Rajkumar Roy Ed., Chapter* 1 (2000), pp. 17–36.

[174] Dirk John Pons and John Kenneth Raine. “Design Mechanism and Constrain Diagram”. In: *Research in Engineering Design* 16.1-2 (2005), pp. 73–85. DOI: 10.1007/s00163-004-0008-9.

[175] David Price and Rob Bodington. “Applying Semantic Web Technology to the Life Cycle Support of Complex Engineering Assets”. In: *International Semantic Web Conference*. 2004, pp. 812–822.

[176] Yoram Reich. “A Critical Review of General Design Theory”. In: *Research in Engineering Design* 7.1 (1995), pp. 1–18.

[177] Thomas Russ. *Ontology Development and the Domain Expert*. Newsgroup comp.misc.ontology.protege.owl. http://thread.gmane.org/gmane.comp.misc.ontology.protege.owl/24502/focus=24514. Mar. 2008.

[178] John Sowa. "Knowledge Contributors". In: ed. by V. F. Hendricks, K. F. Jørgensen, and S. A. Pedersen. Kluwer Academic Publishers, 2004. Chap. Laws, facts, and contexts: Foundations for multimodal reasoning, pp. 145–185.

[179] Reinhard Staebler, Bryan Miller, Paul Rakow, and Thomas Koch. "Connector Architecture for COmputer-Assisted Design and Manufacturing Systems". In: *Journal of Ship Production* 20 (2004), pp. 262–268.

[180] Karsten Stenzel. *STEP-konforme Integration von Konstruktionssoftware in den Schiffsstrukturentwurfprozeß*. Tech. rep. Technische Universität Hamburg-Harburg, 2000.

[181] Susan Thomas. "The Role of Terminology Management in Creating Ontologies". In: *8th International Protégé Conference*. 2005.

[182] Rainer Weigel, Boi V. Faltings, and Berthe Y. Choueiry. "Context in Discrete Constraint Satisfaction Problems". In: *In lth European Conference on Artificial Intelligence, ECAI'96*. John Wiley & Sons, Ltd, 1996, pp. 205–209.

[183] Yiang-Wi Wu, Jeiu-Joui Shaw, and Wei Tann. "Knowledge Management With XML Integrated Within the Full Specification in Ship Design Processes". In: *Journal of Ship Production* 20 (2004), pp. 256–261.

[184] W. T. Zhang, S. Y. Tor, and G. A Britton. "Managing Modularity in Product Family Design with Functional Modeling". In: *The International Journal of Advanced Manufacturing Technology* 1 (2005), pp. 579–588.

[185] L. Zhongtu, W. Qifu, and C. Liping. "A knowledge-based approach for the task implementation in mechanical product design". In: *International Journal of Advanced Manufacturing Systems* 29 (2006), pp. 837–845.

[186] *Knowledge based Engineering and the ICAD System*. KtiWorld. http://www.ktiworld.com/pdf/understanding-tis-2.pdf. 2004.

[187] *Product Data Exchange - Part 1: Assembly Data Exchange*. ProStep e. V. http://www.prostep.org/fileadmin/freie_downloads/Empfehlungen-Standards/VDA/VDA_4956-1_Product-Data-Management_1.0.pdf. 2002.

[188] Holger Knublauch. *Where OWL fails*. Composing the Semantic Web (Blog). http://composing-the-semantic-web.blogspot.com/2010/04/where-owl-fails.html. 2010.

[189] *Pellet Integrity Constraints: Validating RDF with OWL*. Clark & Parsia (Blog). http://clarkparsia.com/pellet/icv/. 2009.

[190] *Imperfect Evaluations*. Drools Blog. http://blog.athico.com/2009/05/imperfect-evaluations.html.

[191] *What is inference and how does it facilitate good rule design and maintenance*. Drools Blog. http://blog.athico.com/2009/11/what-is-inference-and-how-does-it.html.

[192] *Kazuki (Website)*. SemWebCentral. http://projects.semwebcentral.org/projects/kazuki/.

[193] *RDFReactor (Website)*. Ontoware. http://ontoware.org/projects/rdfreactor/.

[194] *Jena bean - A library for persisting java beans to RDF (Website)*. Google Code.

[195] *The Apache Velocity Project (Website)*. Apache Foundation. http://velocity.apache.org/.

[196] Michael Zimmermann. *Owl2Java - A Java Code Generator for OWL (Website)*. http://www.incunabulum.de/projects/it/owl2java/owl2java-a-owl2java-generator.

[197] *JBoss Drools Documentation (Website)*. JBoss Inc. www.jboss.org/drools/documentation.html.

Zusammenfassung

Insbesondere in der heutigen Situation ist im Schiffbau ein starker Wettbewerb zwischen den einzelnen Werften zu verzeichnen. Marktvorteile sind hierbei in dieser oft interdisziplinär ausgerichteten Branche nicht nur über die Baukosten, sondern ebenfalls durch eine konstant hohe Bauqualität wie auch durch schnelle Liefertermine zu erzielen. Neben anderen Maßnahmen, die z. B. im Bereich der Fertigung oder bei der Projektentwicklung zu diesem Zweck ergriffen werden, wird im Bereich der stahlschiffbaulichen Detailkonstruktion Standardisierung erfolgreich eingesetzt. Mit solchen sog. Baustandards ist die eindeutige Abbildung von Best-Practice-Lösungen möglich, so dass eine durchgängige Anwendung qualitätsgeprüfter Lösungen für genau definierte Einsatzszenarien sichergestellt werden kann.

Ausgehend von diesen Rahmenbedingungen wird in dieser Arbeit die Problematik der Standardisierung in der stahlschiffbaulichen Detailkonstruktion aufgegriffen und die Einsatzmöglichkeiten von rechnerbasierten Methoden für eine vereinfachte, beschleunigte und reproduzierbare Anwendung von Baustandards evaluiert. Im Rahmen der durchgeführten Untersuchungen wird hierbei ein besonderes Augenmerk auf die Rolle von Konstruktionswissen wie auch Problemlösungswissen sowie deren jeweilige Abbildung mit Hilfe rechnerbasierter Verfahren gelegt. Unter Verwendung eines interdisziplinären Ansatzes werden Konzepte aus den Bereichen Wissenstheorie, Logik und Informatik kombiniert und auf den schiffbaulichen Kontext angewendet. Die Arbeit sieht sich somit als Bindeglied zwischen diesen Disziplinen. Im Rahmen einer theoretischen Vorbetrachtung werden in einem ersten Schritt relevante Grundlagen oben benannter Disziplinen zusammengefasst und für die weitere Bearbeitung relevante Begrifflichkeiten definiert. Hieraus aufbauend werden sodann in mehreren Schritten Lösungsansätze entwickelt und danach am Beispiel schiffbaulicher Testfälle evaluiert.

Da der Entwicklungsprozess im Schiffbau in erster Linie durch Wissen vorangetrieben wird, ist für die IT-basierte Abbildung derartiger Prozesse eine mit der ingenieurwissenschaftlichen Sichtweise verträgliche Herangehensweise, d. h. eine positivistische Sichtweise, erforderlich. Wissen wird somit als verifizierbar und allgemeingültig betrachtet. Aufbauend auf allgemein anerkannten Axiomen und grundlegenden Übereinkommen kann weiteres Wissen mit Hilfe von sog. Fakten und qualifizierten Relationen zwischen diesen Fakten definiert werden. Dieses Netzwerk aus Fakten und Relationen lässt sich elektronisch abbilden, so dass hiermit eine IT-basierte, elektronisch auswertbare Wissensrepräsentation erstellt werden kann.

Im Rahmen des Entwurfsprozesses im Schiffbaus, d.h. während des Designs, wird Wissen zur Problemlösung herangezogen. Lösungen werden erarbeitet, bewertet und verworfen, modifiziert oder angewendet. Für den Einsatz in rechnerbasierten Systemen können hierfür generell drei verschiedene Verfahrensweisen zur Problemlösung identifiziert werden, nämlich *Fallbasiertes Schließen*, *Statistische Methoden* sowie *Regelbasierte Verfahren*. Da im Bereich der Detailkonstruktion, also in einem Bereich des schiffbaulichen Entwurfsprozesses, der sich nicht primär durch Innovation auszeichnet, Nachvollziehbarkeit und Reproduzierbarkeit zwingend erforderlich sind, sind hier regelbasierte Methoden klar zu bevorzugen. Unter Ver-

wendung von Begrifflichkeiten aus dem Gebiet der Variantenentwicklung kann die Zielsetzung der Arbeit somit so präzisiert werden, dass eine automatisierte Anwendung von Konstruktionsstandards sich primär mit der regelbasierten Auswahl und Platzierung bekannter Bauteile oder Lösungen in der stahlschiffbaulichen Struktur befasst. Es ist also ein Problem aus dem Bereich des sogenannten *Configuration Design* oder *Skeleton Design* zu lösen. Der Entwurfsprozess wird hierbei als konstruktive Suche nach Lösungen verstanden.

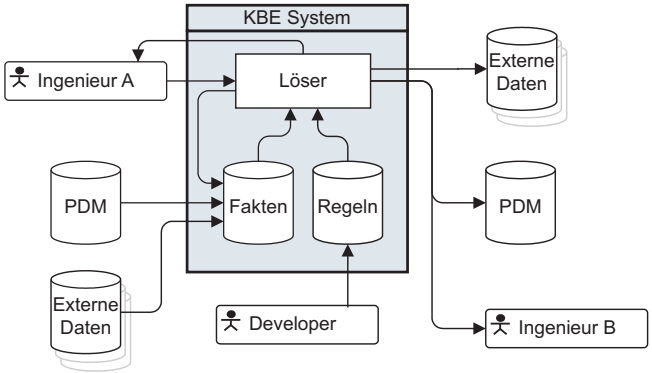
Im Rahmen des Konstruktionsprozesses im Schiffbau werden verschiedene Tätigkeiten, sogenannte *Entwurfsaufgaben*, parallel wie auch konsekutiv ausgeführt. Als typisches Beispiel einer Entwurfsaufgabe kann z. B. das komplette Bracket-Design als Ganzes genannt werden. Die Abfolge der einzelnen Tätigkeiten ist hierbei von vielfältigen gegenseitigen Abhängigkeiten gekennzeichnet, so dass eine geschlossene Lösung vielfach nicht möglich ist sondern stattdessen vielfach eine iterative Arbeitsweise angewendet werden muss. Jede Entwurfsaufgabe besteht aus mehreren sogenannten *Entwurfsaktivitäten*. Eine Entwurfsaktivität repräsentiert hierbei eine atomare Entwurfseinheit, also die kleinste geschlossen definierbare Handlungsweise eines Ingenieurs, für die das Entwicklungsziel, also die Motivation, sinnvoll benannt werden kann. Im Rahmen des Bracket-Design z. B. wird durch eine Entwurfsaktivität ein Knieblech ausgewählt und positioniert.

Atomare Entwurfsaktivitäten sind somit für einen wissensbasierten Konstruktionsansatz prädestiniert, da bei derartigen Prozessen verhältnismäßig einfache Zusammenhänge vorliegen. Der sog. *Designkontext*, d. h. die Abhängigkeit der Aktivität von externen Informationen wie z. B. Plattendicken der umliegenden Stahlstruktur oder auch funktionale Aspekte, ist begrenzt. Eine explizite Formulierung des Entwurfsziels wie auch eine geschlossene Formulierung der Verfahrensweise zur Lösung der Entwurfsaufgabe ist möglich. Viele derartige Designprobleme kommen ausserdem in ausreichend hohen Stückzahlen vor, so dass die für die Entwicklung der entsprechenden wissensbasierter Formulierungen erforderlichen Aufwände wirtschaftlich vertretbar sind.

Für die Entwicklung benannter wissensbasierter Methoden kommen beispielhaft ausgewählte Testfälle zum Einsatz. Mit diesen Testfällen, die an häufig auftretende Konstellationen im stahlschiffbaulichen Detailkonstruktionsprozess angelehnt sind, werden übliche Anwendungsszenarien abgebildet, so dass vielfältige Aspekte der Entwicklung geeigneter Algorithmen untersucht werden können. Neben der Berücksichtigung wirtschaftlicher Aspekte, siehe oben, wurde die Auswahl unter anderem beeinflusst durch die am Prozess beteiligten Konstruktionselemente, durch die Komplexität der Designaktivität und durch die für die jeweilige Aufgabenstellung erforderliche Lösungsstrategie, so dass letztlich folgende drei Testfälle ausgewählt wurden:

- Knieblechverbindung zwischen zwei Profilen
- Eckradien (Corner Cutouts) in Platten
- Cutouts und Clips für Profilausschnitte

Für die Entwicklung wissensbasierter Lösungsalgorithmen sind, wie oben benannt, in erster Linie regelbasierte Methoden geeignet. Hierfür kommt eine Systemarchitektur wie in nachfolgender Abbildung gezeigt zum Einsatz.



Überblick über die Systemarchitektur

Kernstück der Architektur ist als *Knowledge-based Engineering* (KBE) System die sog. *Regelmaschine*. Diese besteht aus einem Löser, der für die Abarbeitung der auf ein ausgewähltes Problem zutreffenden Regeln zuständig ist. Die Entwicklung der Regeln erfolgt durch die zuständigen Entwickler. Die für die Problemlösung erforderlichen Informationen wie z. B. die relevante Schiffsstruktur, zutreffende Konstruktionsstandards etc., werden über geeignete Schnittstellen aus einem CAD-System importiert bzw. können aus weiteren Datenquellen manuell oder automatisch geladen werden. Ergebnisse der Regelmaschine werden zur Weiterverarbeitung gespeichert bzw. werden dem Anwender mitgeteilt. Über geeignete Schnittstellen werden die Ergebnisse einer Konstruktionsaktivität für externe Systeme zur Verfügung gestellt, so dass Ergebnisse von diesen Systemen direkt weiterverwendet werden können.

Als Regelsystem kommt im Rahmen dieser Arbeit eine der bekanntesten Lösungen aus dem Open-Source-Bereich, die Regelmaschine *Drools*, zum Einsatz. Diese Komponente bietet alle gängigen Möglichkeiten zur Abbildung regelbasierter Entscheidungsprozesse, wobei der Lösungsweg feinfühlig gesteuert werden kann. Dank eines modularen Aufbaus ist die Erweiterung der Lösung mit Hilfe von in Java geschriebenen Komponenten einfach möglich.

Für die elektronische Abbildung von Konstruktionsstandards, der für das jeweilige Konstruktionsproblem relevanten Stahlstruktur wie auch weiterer für den Entscheidungsprozess erforderlicher Informationen kommen für den jeweiligen Einsatz entwickelte Datenmodelle zum Einsatz. Dieser Ansatz wurde gewählt, da durch aufgabenspezifisch definierte Modelle eine klare und übersichtliche Struktur erreicht werden kann, welche sich in IT-basierten Algorithmen entsprechend gut einbinden läßt. Auch wenn die Modellierung, wo möglich, an standardisierte Formate wie z.B. STEP angelehnt ist bzw. relevante Konzepte aufgreift, so wurde aus Gründen der damit einhergehenden Komplexität eine direkte Implementierung mittels derartiger Formate ausgeschlossen. Die Datenmodelle wurden mit Hilfe der *Web Ontology Language* OWL entwickelt. Die Verfügbarkeit von vielfältigen Softwarekomponenten zur Anwendung derartiger Modelle, eine klare und entscheidbare Syntax und Semantik wie auch die Möglichkeit einer computerunterstützten Validierung der erzeugten Datenmodelle sprachen hierbei für diesen Ansatz.

Für die Hinterlegung der Testfälle mit Informationen bezüglich der Standardisierung wie auch für die Entwicklung der wissensbasierten Algorithmen zur automatisierten Konstruktion der jeweils gewählten Stahlkonstruktionsdetails wurde der Bauteilkatalog eines Containerschiffes mit ca. 2500 TEU einer größeren deutschen Werft herangezogen und detailliert

analysiert. Somit konnte sichergestellt werden, dass die entwickelten Lösungen als praxisrelevant anzusehen sind. Weiterhin konnten durch die Auswahl der oben genannten drei Testfälle unterschiedliche Aspekte des wissensbasierten Entwurfes fokussiert werden.

Testfall No. 1, d.h. der Knieblechentwurf, zeichnet sich durch eine einfache Aufgabenstellung aus. Die Anwendung standardisierter Kniebleche ist hierbei durch einen einfachen Auswahlprozess möglich, wobei unabhängig von der Aufgabenstellung maximal eine gültige Lösung erzielt wird. Weiterhin ist der Designkontext für diesen Testfall einfach beschrieben insofern, als das ausschließlich die Geometrie der direkt beteiligten Profile sowie eventuelle Anforderungen hinsichtlich des Einbauraumes eine Rolle spielen. Auch dies erleichtert die Formulierung der Lösungsmethodik mittels Regeln. Es kann ein direktes, eindeutiges Lösungsverfahren beschrieben werden.

Die gleiche grundlegende Konstellation ist für die Auswahl geeigneter Eckradien in Platten gegeben. Auch hier wird, sofern der Designkontext vollständig ausgewertet werden konnte, höchstens eine einzige valide Lösung generiert. Da derartige Eckradien primär von funktionellen Gesichtspunkten beeinflusst werden, spielt in diesem Testfall daher die Auswertung der Funktion der Stahlstruktur eine bedeutende Rolle; die Geometrie der Stahlstruktur ist nur von sekundärer Bedeutung. Neben den direkt involvierten Bauteilen wie in diesem Testfall der Platte, auf welcher der Eckradius zu platzieren ist, sind allerdings benachbarte Bauteile wie auch die Funktion der angrenzenden Räume auszuwerten. Fungiert die Platte als Tankwand, so sind z. B. ausschließlich wasserdichte Eckradien möglich. Dies kann durch geeignete Formulierungen der Regeln abgebildet werden, so dass der Designalgorithmus korrekte Lösungen aus dem Standardkatalog auswählen kann. Weiterhin wurde bei der Entwicklung der Regeln Wert darauf gelegt, dass im Falle von sich widersprechenden Anforderungen eine entsprechende Warnung ausgegeben wird. Somit dient das System der Qualitätssicherung und weist frühzeitig auf Unklarheiten hin.

Die wissensbasierte Auswahl und Auslegung von Profildurchführungen aus einer Liste standardisierter Lösungen stellt die komplexeste Entwurfsaktivität dar, deren Umsetzung im Rahmen dieser Arbeit untersucht wurde. Schneidet ein Profil eine Platte, wie es bei einer Profildurchführung der Fall ist, so ist ein geeigneter Ausschnitt in dieser Platte erforderlich. Die Form und Lage dieses Ausschnittes wird hierbei unter anderem bestimmt durch den Typ und die Abmessungen des Profils, aber auch durch den gesamten Designkontext. So beeinflussen u. a. die Bauweise (Platte aufgesetzt vs. Profile eingeschoben), Anforderungen hinsichtlich der Festigkeit einer Verbindung von Platte und Profil wie auch unterschiedliche Fertigungskosten für verschiedene Lösungen den Entwurfsprozess. Weiterhin ist in einigen Fällen ein Dicht- oder Riegelblech erforderlich. Es sind somit zwei verschiedene Konstruktionselemente aufeinander abgestimmt auszuwählen. Im Gegensatz zu den beiden ersten Testfällen können in diesem Fall verschiedene gültige Lösungen für eine gegebene Aufgabenstellung ermittelt werden. Die Auswahl obliegt hierbei dem Anwender. Um diesen zu unterstützen, wurde ein gewichteter Ansatz zur Qualitätsbewertung der erzeugten Lösungen entwickelt. Mit diesem ist es möglich, verschiedene Aspekte der Lösung wie auch der in dieser Lösung verwendeten Bauteile qualitativ zu bewerten und mit Hilfe der durch den Anwender erfahrungsbasiert definierten Wichtungsfaktoren eine globale Qualitätskennzahl zu ermitteln. Letztere kann im Rahmen einer vollautomatischen Konstruktion herangezogen werden. Die Notwendigkeit einer manuellen Auswahl aus Designvorschlägen kann somit entfallen.

Die Ausführung einer Entwurfsaktivität befasst sich mit der Ermittlung einer konstruktiven Lösung für eine bekannte Aufgabenstellung in einem eindeutig definierten Designkontext.

Diese Informationen werden hierbei durch den Anwender festgelegt. In einem weiteren Schritt wurde im Rahmen der Arbeit dieser Ansatz erweitert. Ist die Entwurfsaufgabe bekannt, z. B. Detailierung von Profildurchführungen, so kann der Grad der Automatisierung in der Konstruktion mit Hilfe von sogenannten wissensbasierten *Entwurfsmustern* (Design Patterns) weiter erhöht werden. Für obiges Beispiel lassen sich z. B. Algorithmen entwickeln, mit denen automatisch sukzessive alle vorhandenen Durchdringungen von Profilen und Platten ermittelt, untersucht und bewertet werden können. Es findet also jeweils eine automatische Bestimmung des Entwurfskontexts statt. Ist dieser ermittelt, so kann anschließend hierfür ebenfalls automatisiert eine Ermittlung geeigneter konstruktiver Lösungen erfolgen, indem die zugehörige Entwurfsaktivität ausgeführt wird.

Ist die Abfolge von Entwurfsaufgaben bekannt, so ist eine weitere Automatisierung der Detailkonstruktion möglich. Werden z. B. wie beim Design einer Bodenwrange Kniebleche, Profildurchführungen und Eckradien üblicherweise nacheinander definiert, so können die entsprechenden Entwurfsmuster durch geeignete Regelformulierungen so verknüpft werden, dass durch die Regelmaschine diese Entwurfsmuster nacheinander abgearbeitet werden. Unter Verwendung der vorgestellten Designaktivitäten kann z. B. ein Entwurfsmuster definiert sein wie folgt gezeigt.

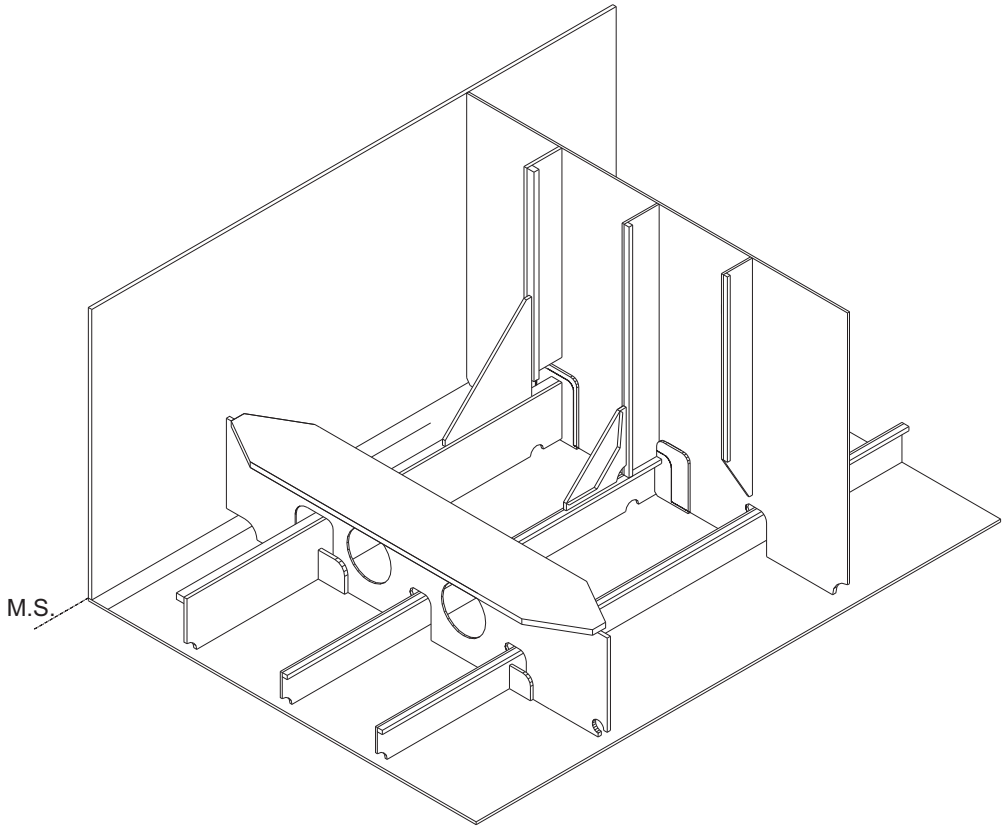
Entwurfsmuster für die Detailierung von Platten

Wird durch den Anwender eine vorhandene Bodenwrange ausgewählt, so wird mittels der beschriebenen Algorithmen die Detailierung vollautomatisiert wissensbasiert durchgeführt. Für die ausgewählten Testfälle ergibt sich hierbei ein Ergebnis wie folgt dargestellt.

Bei der Definition einer Abfolge der Entwurfsmuster ist es möglich, Bedingungen hinsichtlich der Ausführung bestimmter Muster zu definieren, so dass z. B. nur auf ungeraden Spanten leichte Bodenwrangen erzeugt werden. Nur in diesen Fall wird ein entsprechendes Entwurfsmuster ausgeführt. Der weitergehende Designkontext kann somit berücksichtigt werden.

Mit Hilfe wissensbasierter Entwurfsmethoden ist folglich eine Beschleunigung der Konstruktion möglich. Für jede Aufgabenstellung werden standardisierte, qualitativ hochwertige wie auch zugleich aus Fertigungssicht optimale Lösungen ermittelt. Werden Aufgabenstellungen identifiziert, für die keine geeigneten Baustandards vorliegen, so wird der Anwender entsprechend informiert. Der Standardisierungsgrad kann somit erhöht werden, wodurch durch die erzielten Serieneffekte ein Beitrag zur Qualitätssicherung wie auch zur Kostensenkung geleistet werden kann.

Die im Rahmen dieser Arbeit entwickelten Lösungen fokussierten in erster Linie auf die Entwicklung einer Softwarearchitektur zur wissensbasierten Problembeschreibung wie auch darauf aufbauend auf die Entwicklung und Evaluation geeigneter Algorithmen zur Problemformulierung. Im Gegensatz zur vorgestellten prototypischen Realisierung ist für eine vollständige Integration in den Konstruktionsprozess allerdings eine Verbesserung der Möglichkeit zur Integration externer Komponenten wie der hier eingesetzten Regelmaschine Drools in das eingesetzte CAD-System erforderlich. Nur mit derartigen Schnittstellen ist es möglich, Geometrie sowie Topologie der vorhandenen Stahlstruktur per Software allein, also ohne



Ergebnis des Entwurfsmusters zur Automatisierten Detailierung der Testfälle

manuelle Datenaufbereitung, zu untersuchen und, so gewünscht, zu modifizieren. Weiterhin stellt die Integration von Informationen aus verschiedenen Systemen bzw. Datenquellen nach wie vor ein großes Problem dar. So wird z. B. die funktionale Beschreibung von Räumen selten im CAD-System vorgehalten, sondern muss manuell dem Tankplan und Raumplan entnommen werden. Anforderungen hinsichtlich Festigkeit sind in Klassevorschriften definiert sowie, wenn vorhanden, in den zugehörigen Klasse-Tools in Software abgebildet. Schnittstellen zur externen Ansteuerung dieser Komponenten sind im Allgemeinen nicht vorhanden. Dies sind Themen, die einer weiteren Bearbeitung bedürfen, um die Anwendbarkeit der hier vorstellten Ansätzen im produktiven Einsatz zu ermöglichen.

Declaration of Primary Authorship

With this statement I declare that I have independently completed the above thesis entitled *Knowledge-Based Design Patterns for Detailed Steel Structural Design*. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

Rostock, May 2010

Michael Zimmermann

Acknowledgement

With the thesis mostly being worked out during my stay at the Chair of Naval Architecture at the University of Rostock, my first and foremost profound thanks go to Prof. Robert Bronsart for giving me this opportunity and for his encouragement and support throughout the complete time. I am heartily thankful to Prof. Wolfgang Fricke from the Technical University of Hamburg-Harburg for many insights into steel structural design from a perspective of strength and vibrations. Also, I would also like to thank the complete team of the Chair of Naval Architecture for many valuable discussions and the ongoing support, particularly Gernot Knieling and Peter Kolarov for a critical review of this thesis.

Finally, to my family words alone cannot express what I owe them for their encouragement and whose patience enabled me to complete this thesis. A special thanks to my parent for editing and proof-reading, to Jessica for keeping up with my moods during final editing, to my young one for always remembering me about life behind the monitor and to that big pile of wood somewhere out in the country.

Michael Zimmermann

A. The Owl2Java Generator

Similar to alternative projects like *Kazuki* [192] or *RdfReactor* [193], for the handling of OWL instance data *Owl2Java* generates a Java API from a given ontology schema. This allows the native, programmatic use of existing information provided in an ontology. For an alternative approach based on annotated Java classes, see e.g. JenaBean [194]. Also, Owl2Java includes a generator to transform OWL ontologies into a db4o object database with the instance data given by the ontology. The automatic mapping of instance data is supported.

Based on *Jena* [131] the API allows for the seamless integration of OWL ontologies into applications and makes persistent data stored in an ontology available transparently. Native Java objects are used. These interact with the ontology via accessor methods like e.g. *person.setName("Homer Simpson")*. A direct manipulation of triples is not required. Methods of the API are directly translated into the corresponding triple constructs of OWL data. Using one of the local or remote database options of Jena distributed systems can be developed.

A.1. Application Design

Similar to RdfReactor Owl2Java uses an intermediate meta-model. With such an approach additional transformation steps can be applied to the meta model without changing the underlying ontology. The originating information is not changed.

The objective of the meta model is to capture all relevant information of the ontology as native Java objects. For this purpose the meta model (JModel) offers classes to handle packages (JPackage), classes (JClass), properties (JProperty) and property restrictions (JOnPropertyRestriction) as well as property representations (JPropertyRepresentation).

As Owl2Java generates an individual Java package for each namespace defined as part of an ontology, a *JPackage* contains all classes, properties and restrictions for a certain namespace. A *JClass* instance holds information about an OWL class like e.g. name, URI, it's properties or any restrictions defined. *JProperty* instances are used to define the characteristics of a property like domain, range or URI. Restrictions for a property are captured with a *JOnPropertyRestriction* and link the restricted property with the corresponding class. A *JPropertyRepresentation* is the representation of a property in a certain class with all valid restrictions taken into account.

Using the enhanced graph features offered by Jena, generated classes are subclassed from *IndividualImpl* and are registered as *BuiltinPersonality* instances. Hence, an easy conversion of OWL data to Java classes is achieved.

The generation process can be summarized as follows:

- Read the model: The model is read from an URI and a Jena OntModel is created. All imports are processed
- Analyze the model: As an intermediary step the model is analyzed. Namespaces are determined and additional prefixes for unprefixed namespaces are generated.
- Create the JModel: For each class and property in the ontology the corresponding JModel representation is created as JClass or JProperty and assigned to the relevant package. If needed, anonymous classes (union classes etc.) are converted to named classes. Restrictions are analyzed. Relevant JOnPropertyRestrictions are created. The hierarchy of properties and classes is stored as graphs.
- Prepare the JModel: An additional step is performed to prepare the JModel for export. Consistency checks are performed. For each class all relevant properties and restrictions are aggregated, i.e. properties of parent classes are copied to the class etc.
- Write to Disk: The final JModel is written to disk. The package structure is created. For each class the corresponding Java class and interface files are generated. For this purpose, the *Velocity Template Engine* [195] is used. Vocabulary and factory classes are written as part of a helper package.

A.2. Classes and Properties

In contrast to Java OWL allows the definition of classes and properties independently. For both multiple language constructs are defined. The implications of these constructs for Owl2Java and the solutions developed needs to be taken into account. Accessor methods for OWL ontologies in Java need to bridge the gap in the semantics of the different languages. Next to Closed-World vs. Open-World and the Unique-Name-Assumption in OWL, the underlying understanding of inheritance in OWL and Java needs to be regarded.

In Java inheritance allows to modify and extend a parent object. E.g. additional attributes or methods are defined. Existing methods are overwritten and hence changed. In OWL a different approach is chosen. Here, subclasses are assumed to restrict the degrees of freedom of a parent class. Examples:

- A property with domain *ClassA* is disabled in a subclass *ClassB* via a restriction *myProperty max 0*.
- Using a reasoner, a Class *MyClass intersectionOf(ClassA, ClassB)* is classified as subclass of *ClassA* and *ClassB*. From a Java perspective *MyClass* as intersection should be modeled as parent class of *ClassA* and *ClassB*.

Therefore, not all concepts from OWL can be applied to native Java interfaces.

In the following, selected details about the relevant constructs of OWL Lite and OWL DL are presented. More information about the mapping approach can be found in [133]. For more information about the implementation, see [196].

Classes In Owl2Java for each OWL Class a Java interface and the implementing Java class is generated. Properties are defined as class attributes with accessor methods following the Java bean notation.

As Java does not support multiple inheritance, interfaces are used to define *rdfs:subClassOf* relationships. Using interface definitions, OWL class definitions like e.g. *Cls subClassOf ClassA* and *Class subClassOf ClassB* can be expressed as follows:

```
1    interface ICls extends IClassA , IClassB {}
```

The corresponding Java class *Cls* then implements the interface *ICls* hereby inheriting all methods and properties defined in *IClassA* and *IClassB*.

Properties In OWL properties can be defined independently from OWL classes and without any definition of the type of the property. All unbound properties are assumed to have Owl:Thing as domain and a range of Owl:Thing for object properties or XMLLiteral for datatype properties.

In Owl2Java a base class and interface named *Thing* and *IThing* are defined in the base Java package. Here, all properties without a domain are defined. All defined classes in an ontology have the *IThing* interface as parent. For properties without a range the range is set to *IThing* for object properties or *java.lang.String* for data-type properties.

If a domain is defined for a property the property is added to the corresponding interface. According to the OWL standard, for properties with multiple domains the intersection of the domain classes should be used. In contrast to the technical recommendation, a property with multiple domains is added to each interface and class directly. Intersection classes are neither generated nor used as this approach presents a better fit with the Java semantics.

A range defines the types a property can have. For object properties the corresponding interface is used. For datatype properties the Jena data types or a custom mapping from XSD datatype to Java class can be used. For the latter, see the configuration options in *XsdUtils.Java*. Multiple ranges are handled as intersections for object properties. Union classes are generated.

A property can either be an Object Property or a Data-type Property. Object properties reference class instances while datatype properties contain values of a type according to XML Schema, see above.

Functional properties are properties that allow only a single instance of this property for a class instance, i.e. for an instance *MS Victory* the property *ownedBy* can only have a single Item. In Owl2Java for functional properties as accessor methods only get-, set- and remove-methods are generated. List-type Accessor methods are removed hence restricting the property instance count to one. For more information about more elaborate property types, see [196].

Class Features and Axioms In addition to the simple *rdfs:subClassOf* construct, OWL Lite and especially OWL DL allows more complex constructs to define classes, namely with *intersectionOf*, *unionOf*, *complementOf*, *disjointWith* and *oneOf*.

For the first two types a new class is generated as the union or intersection of two or more primitive classes. For e.g. if a class *MyClass* is an intersection of *ClassA* and *ClassB* in OWL this is expressed via an internal anonymous class as follows:

```
1    AnonClass intersectionOf(ClassA , ClassB)
2    MyClass subClassOf AnonClass
```

The anonymous class does not have a valid URI and is thus not reference-able directly via a qualified name.

The Java implementation creates new classes and interfaces, marked as anonymous, to represent anonymous classes and uses these interfaces as parents. For the naming schema used see *NamingUtils.Java*. Identical anonymous class definitions in multiple OWL constructs are merged into a single Java class and interface representation. E.g. the following OWL definition:

```
1  A1 unionOf(ClassA, ClassB)
2  MyClass subClassOf A1
3  A2 unionOf(ClassA, ClassB)
4  MyClass2 subClassOf A2
```

is defined in Java as:

```
1  interface A1 extends ClassA, ClassB
2  interface MyClass extends A1
3  interface MyClass2 extends A1
```

A class defined as *unionOf* of two or more parent classes inherits all properties from all parent classes. Using an anonymous class for an owl construct *MyClass unionOf(ClassA, ClassB)* in Java, this is defined as follows:

```
1  interface AnonClass extends ClassA, ClassB
2  interface MyClass extends AnonClass
```

In contrast to the unionOf construct an *intersectionOf* defines a subclass that inherits only the properties and characteristics that are present in both parent classes or their instances. Using an anonymous class for an owl construct *MyClass intersectionOf(ClassA, ClassB)* in Java this is defined as follows:

```
1  interface AnonClass
2  interface ClassA extends AnonClass
3  interface ClassB extends AnonClass
4  interface MyClass extends AnonClass
```

with the domain of all common properties changed to AnonClass.

A.3. Limitations

Classes

- Any distinction between necessary (rdfs:subClassOf) and necessary and sufficient (owl: equivalentClass) declarations as stated by OWL is ignored.
- For *rdfs:equivalentClass* only OWL Lite is supported. Complex restrictions via rdfs: equivalentClass - e.g. (*Ferry rdfs:eqivalentClass (Ship and hasCargo Cars)* - are not supported
- Complex constructs for *unionOf*, *intersectionOf* are currently not supported. The same holds true for the constructs *complementOf*, *disjointWith* and *oneOf*.

Properties While the meta-model already has information about the following property types or constructs, currently these are not used by the code generator.

- Inverse functional properties are treated as primitive properties.

- Transitive properties are not supported and are treated as primitive properties.
- Multiple ranges for properties are treated as UnionClass.
- Range definitions of subproperties are not "inherited" to the parent properties.

Restrictions

- Complex anonymous restrictions - e.g. *hasPerson allValues (ClassA and (ClassB or ClassC))* are not evaluated and are ignored.
- Multiple *hasValue* restrictions are equivalent to anonymous restrictions and hence ignored.
- Constructs *hasValue* and *someValuesFrom* are present in the meta model yet not supported by the template engine.
- Construct *minCardinality* is currently ignored.

B. An Open-World Comparator for Drools

For the comparison $A == B$ the equal operator == in Drools return true only if A is equal to B or $A == B == null$. As introduced in section 7.3.4.2 the "equal or null" *eon* operator extends this concept of equality in such a way that *eon* returns true if

- either A is equal to B
- or $A == B == null$
- or $B == null$.

The last aspect extends the comparison to adhere to certain cases of the open world assumption and returns true even if $B == null$ as there may be some object B with $A == B$ that is not known yet.

In Java strings are defined as empty strings *""* even if no constructor has been called explicitly. For subclasses of *java.lang.Number* any variable is set to zero upon instantiation. Boolean variables are false by default, arrays may be empty. This requires special consideration implementing the algorithms for evaluation and comparison.

All evaluator definitions present in Drools 5 as well as supplementary classes and definitions are part of the *drools-core* module. For each operator like == an *evaluator definition* class is present that implements *org.drools.base.evaluators.EvaluatorDefinition*. This class can contain multiple *evaluator* classes that are responsible to evaluate the operator for a certain object type. E.g. for the operator *contains* different comparison strategies may be required depending on whether the objects to compare are string or arrays. Evaluator classes are derived from *org.drools.base.BaseEvaluator*.

Implementing the Evaluator Definition The basic framework for any custom operator is as follows:

```
1  public class EonEvaluatorDefinition implements EvaluatorDefinition {
2    public static final Operator EON = Operator.addOperatorToRegistry("eon", false);
3    public static final String[] SUPPORTED_IDS = { EON.getOperatorString(), };
4
5    private EvaluatorCache evaluators = new EvaluatorCache() {
6      private static final long serialVersionUID = 400L;
7
8      {
9        // register custom evaluators
10     }
11   };
12 }
```

In this example, the evaluator definition is defined by the class *EonEvaluatorDefinition*. The second line defines the operator and adds the custom operator "eon" to the Drools operator vocabulary. The second argument in the call to *addOperatorToRegistry()* defines whether and how the operator handles the negated statement *not eon*. The *EvaluatorCache* instance

evaluators contains all instances of evaluator classes, i.e. the instances that are responsible for the evaluation process for known and registered object types.

Also, the evaluator definition class needs to implement the support methods from the *EvaluatorDefinition* interface as shown in the Drools documentation [197].

Implementing the Evaluator for Objects The evaluation of an operator with respect to the values left and right of the operator is handled by custom classes derived from *BaseEvaluator*. In this example, these classes are defined as static inner classes of the main class *EonEvaluatorDefinition*. For the evaluation of generic java object, i.e. *java.lang.Object*, the evaluator class is defined as follows:

```
1   public static class ObjectEonEvaluator extends BaseEONEvaluator {
2     private static final long serialVersionUID = 400L;
3
4     public final static Evaluator OBJECT_INSTANCE = new ObjectEonEvaluator();
5     public final static Evaluator DATE_INSTANCE =
6                                   new ObjectEonEvaluator(ValueType.DATE_TYPE, EON);
7
8     public ObjectEonEvaluator() {
9             super(ValueType.OBJECT_TYPE, EON);
10    }
11
12    public ObjectEonEvaluator(ValueType type, Operator operator) {
13            super(type, operator);
14    }
15
16    public String toString() {
17      return getValueType().getName() + " " +
18             getOperator().getOperatorString();
19    }
20
21    public boolean evaluate(InternalWorkingMemory workingMemory,
22                            final InternalReadAccessor extractor,
23                            final Object object1,
24                            final FieldValue object2) { }
25
26    public boolean evaluateCachedRight(InternalWorkingMemory workingMemory,
27                                       final VariableContextEntry context,
28                                       final Object left) { }
29
30    public boolean evaluateCachedLeft(InternalWorkingMemory workingMemory,
31                                      final VariableContextEntry context,
32                                      final Object right) { }
33
34    public boolean evaluate(InternalWorkingMemory workingMemory,
35                            final InternalReadAccessor extractor1,
36                            final Object object1,
37                            final InternalReadAccessor extractor2,
38                            final Object object2) { }
39  }
```

The static instance of the class is used to register the evaluator in the static constructor block of the *EvalautorCache* instance *evaluators* as follows:

```
1   addEvaluator(ValueType.OBJECT_TYPE, EON, ObjectEONEvaluator.OBJECT_INSTANCE);
2   addEvaluator(ValueType.DATE_TYPE, EON, ObjectEONEvaluator.DATE_INSTANCE);
```

If wanted, a single evaluator instance can be registered for multiple object types or ValueType types.

Implementing the Evaluator Functions Due to the way drools handles objects internally four different evaluator functions need to be implemented, see the documentation for *org.drools.spi*.

Evaluator. As these are some of the most highly stressed parts of the Drools code base these methods should be designed for optimal performance.

```
1   public boolean evaluateCachedRight(InternalWorkingMemory workingMemory,
2                                      final VariableContextEntry context,
3                                      final Object left) {
4     final Object object =
5           context.declaration.getExtractor().getValue(workingMemory, left);
6     if (object == null) {
7           return true;
8     }
9     final Object value = ((ObjectVariableContextEntry) context).right;
10    if (value == null) {
11          return object == null;
12    }
13  return object.equals(value);
14  }
```

On line 2 the left object is extracted. If this object is null the *eon* operator returns true always, see lines 3 - 5. On line 6 the right object is extracted from the variable context. If the right value is null the evaluate functions returns true only if *left* == *right* == *null*, see lines 7-9. If both objects are not null, the java *equals()* is called and the result is returned, see line 10.