University of Applied Sciences Northwestern Switzerland
School of Business

Expressiveness of Enterprise Modelling Languages

Master Thesis

Author: Simon Nikles

Date: February 28, 2010

Supervisor: Prof. Dr. Knut Hinkelmann

Study Programme: Master of Science in Business Information Systems

Executive Summary

This master thesis investigates expressiveness required to support the most important applications of enterprise architecture and to represent corresponding models. The work follows a design research procedure and relies on literature as well as on an interview in the field of Business Service Management (BSM). As formal representation languages RDF(S), OWL and SWRL are considered. The initial assumption was that one or a combination of these representation languages would be adequate to fulfill the requirements of the most important applications. Thus, the thesis statement is formulated accordingly.

Besides the representation of models (i.e. aspects of BPMN and UML), this thesis studies applications such as verification and analysis of models, compliance management, model execution and monitoring. These applications are discussed with respect to possible representation formalisms in order to fullfill the application requirements, i.e. the capability of deducing the required knowledge.

A second line of action addresses the requirements gathered from the interview and describes the development and implementation of a corresponding scenario. The scenario actually has several similarities to the more general applications. In particular parts from dependency analysis and execution monitoring are related to the scenario.

One of the results is a catalogue of representation requirements which was continuously updated during the development phase. Some of these requirements summarise a couple of language features (e.g. different types of restrictions). On this level, representation requirements are assigned to application requirements. Further the representation requirements are mapped to the representation languages - RDF(S), OWL and SWRL. On this basis, suitable languages can be selected.

The more generally approached investigation of applications based on literature as well as the more specific application which was developed consistently led to the insight, that none of the languages (or a combination of them) actually fulfills all requirements.

Major limitations are related to the open world assumption. One basic example is that OWL for example has the expressive power to define different types of cardinality restrictions. If syntactic rules of a modelling language make use of them, open world reasoning will not deduce all deviations. Several cases such as exclusive disjunction, defaults or priorisation are also identified as requirements. These are related to negation and thus could be supported with a closed world assumption.

A further issue is meta-modelling. Even though RDF(S) and OWL-Full allow for meta-modelling, there is a trade-off between decidability and expressive power.

In particular analysis comprises for example calculations which can not be performed, even with the built-ins of SWRL.

The work compares the suitability of the representation languages based on the result of the development phases and concludes that the representation languages are not adequate. Therefore, future work is suggested in order to evaluate and select alternative representation languages.

Statement of Authenticity

I confirm that I performed this work autonomously using only the sources, aids and assistance stated in the thesis, and that quotes are readily identifiable as such.

Date: February 28, 2011

Signature: Simon Nikles

Table of Contents

1 Introduction

This chapter introduces the topic, starting with related fields of research and application to arouse awareness of the problem. After motivating the work by pointing out the gap, the thesis statement and research questions are presented. Next, some terms of importance for the document at hand are discussed to clarify their notion within the work. Finally, the chapter map is presented to offer a clear guidance through the document.

1.1 Background and Problem Statement

Enterprise Architecture (EA) is widely known and the management's need to gain a transparent view on the enterprise's structures and especially the importance of aligning the business and IT is generally acknowledged. The documentation of an enterprise is already valuable, e.g. to gain insight into the existing structures or as a mean for communication, and may be a necessity with respect to regulatory compliance or certifications. However, this is a rather static view. EA as a management instrument provides far higher potentials. The architecture is changing over time and should therefore be planned and controlled, thus also the documentation has to be maintained. There exist several Enterprise Architecture Frameworks to support the management of an enterprise's architecture; whereas Hanschke (2010) states that the existing frameworks were abstract, complex and lack practical orientation. As this stands in contrast to the fact, that there exists tool-support for the methodologies of some frameworks and because the frameworks reflect longstanding experience and development, they are respected as valuable input for this thesis.

An important mean to represent the views and aspects of an enterprise are graphical models such as business process models. Several modelling tools support the design (partially also implementation and controlling) of enterprise architectures; a considerable overview and selection guide is provided by the Institute For Enterprise Architecture Developments[1].

Apart from the graphical representation interesting work has been done with respect to the abstract representation and formalisation of enterprise models. A formal representation mainly targets at the deductive potential.

ATHENE is a tool developed at the FHNW that combines both fields by providing meta-modelling and graphical modelling facilities whilst generating an ontology in the background.

Current research, such as the EU funded project plugIT[2] (Business and IT Alignment using a Model-Based Plug-In Framework) is also dealing with the capabilities of combining graphical models and semantic technologies for the purpose of aligning business and IT.

In the field of semantic technologies (e.g. the semantic web community) or logics respectively there is a longstanding issue concerning expressive power of formal languages. Whilst having a highly expressive language was desirable in general, expressive power increases at the expense of performance or even decidability. For this reason, it is advisable not to simply choose the most powerful language to leave the door open but to choose an adequate level of expressivity needed for a certain application. This is the problem this thesis deals with in the context of enterprise architecture models.

1.2 Research Motivation

The previous section provided an overview on the context relevant to this work. There exists a lot of work on enterprise architectures and formal representations of an enterprise or enterprise

[1] Enterprise Architecture Tools: http://www.enterprise-architecture.info/EA_Tools.htm (Accessed on 2010-10-16)

[2] plugIT Project: http://plug-it.org (Accessed on 2010-10-16)

architecture models respectively. Even though it is important for efficient reasoning support, no work was found that explicitly analyses the necessary expressive power for ontology based support of practical applications of Enterprise Architecture.

The goal of this thesis is therefore to find a suiting level of expressive power (in terms of formal languages) to support practical oriented applications of Enterprise Architecture by means of formal representations, i.e. ontologies. The corresponding thesis statement and research questions will be detailed hereinafter.

The results may provide valuable input for the re-design of the above mentioned ATHENE System and in general for the purpose of creating enterprise ontologies and related applications.

As this work will consider related languages (see section 3.2 for details) one could argue, that modelling the ontology for a certain application would automatically lead to the required language level. However, there are several differences in designing an RDF based system compared to one using OWL, for example. Beside some basic changes such as the use of different constructs, not any RDF(S) based ontology can be simply translated into any OWL dialect. Also switching between OWL 2 profiles is not necessarily possible. Further on it will be investigated, whether it may be advantageous to use rules instead of directly lifting the level of the ontology language.

1.3 Thesis Statement

The expressiveness of one or a combination of RDF(S), OWL dialects and SWRL is adequate to formally represent the most relevant models used for enterprise architecture modelling up to the level that is required for the major application and purpose of enterprise architecture and the models themselves considering the necessary features of the corresponding language specifications.

1.4 Research Questions

In order to affirm or reject the thesis statement, a suitable formalism to represent models and applications has to be determined and tested. This is reflected in the main research question.

Because the thesis is not all-encompassing but places value on practical relevance and focuses on the most important models of an enterprise architecture, the relevant applications in that field and the models required for that have to be identified which is the aim of the first sub question.

From an application-oriented point of view, there is not necessarily a need to cover certain language specifications completely, as, for example, there possibly exist redundant mechanisms to represent the same meaning or some constructs are not required to reach the application goal. In turn, there may result requirements from the application, which are not reflected in language specifications, for example with respect to the relations between different models. What the requirements for a formal representation actually are will be the answer to the second research question.

Once the requirements are identified, the proposed languages can be evaluated against them. The answer of the corresponding third sub question is the basis to derive an answer to the main research question.

The importance of the fourth sub question depends on the results of the research done to answer the previous questions. If several languages or combinations cover all requirements, the answer may influence the answer to the main research question or at least found the basis for a recommendation. If either none or only one approach covers all requirements, the answer will be rather informative, showing the limits of languages or solutions.

1.4.1 Main Research Question

Which language or combination out of RDF(S), OWL and SWRL is adequate to represent the relevant aspects of the most relevant enterprise architecture models?

1.4.2 Sub Question 1

What are the most relevant applications for enterprise architecture and what models (and corresponding languages or subsets of them) are required for those applications?

1.4.3 Sub Question 2

What are the relevant requirements originating from the purposes and applications of the models and specifications of the modelling languages?

1.4.4 Sub Question 3

How can these requirements been fulfilled by one of the languages RDF(S), OWL and SWRL or a combination?

1.4.5 Sub Question 4

What are the shortcomings and strengths of either one representation (or combination)?

1.5 Terms and Definitions

The following sections provide definitions of and discussions about important terms used in this thesis. The goal is to clarify their common meaning and, in particular, if there is no consistent terminology, how the concepts are used in this work. The literature review in chapter 3 will provide more details about specific aspects.

1.5.1 Enterprise Architecture

Lankhorst (2009: 3) defines Enterprise Architecture (EA) as "a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise's organisational structure, business processes, information systems, and infrastructure."

The wording "coherent whole" is seen as very important, because this is probably the key characteristic that may enable what Bernus (2003) point out as belief promoted by EA: "[…] that an enterprise, as a complex system, can be designed or improved in an orderly fashion achieving better overall results than ad-hoc organisation and design." In contrast to an ad-hoc design, an Enterprise Architecture may provide a big picture of the enterprise with its elements and their relations. Also Lankhorst (2009: 3) names its holistic view as the most important characteristic of an EA.

Another characteristic of Enterprise Architecture highlighted by many authors is the fact, that several roles and views are involved. Bernus (2003) for example formulates that "EA is a co-operative effort of designers, analysts and managers and uses enterprise models in the process."

The work at hand will concentrate on models which, as a whole, represent an enterprise or its relevant aspects respectively. The term Enterprise Architecture will therefore usually reflect the notion of a large model (probably compound of a set of related models) representing an enterprise and the relevant parts of its environment.

1.5.2 Model

Lankhorst (2009: 3) refers to a model as "a purposely abstracted and unambiguous conception of a domain" and defines a domain as "any subset of a conception (being a set of elements) of the universe that is conceived of as being some 'part' or 'aspect' of the universe". This definition seems to be in line with the discussion of Kühne (2005) who attempted to establish consensus in the terminology of modelling and therefore outlined several definitions and characteristics

such as the following: A model is based on an original (which does not necessarily have to exist), it should be exact but underlies an abstraction, meaning that only the properties are represented which are of importance for its purpose.

In this work, models will have to be represented in a formal language. Therefore also the modelling languages have to be treated. In accordance with (Karagiannis & Kühn 2002; Kühne 2005) this work considers a model to be expressed in (or created with) a modelling language which bases on a metamodelling language that is again expressed in a language, i.e. a meta^2 language (cp. Figure 1). The model is seen as an indirect model of a metamodel which itself is a (direct) model of the language and an (indirect) model of a meta-meta-model.

Kühne (2005) distinguishes token models which refer to individuals and their singular aspects (such as an UML object model) and type models which refer to classified elements and capture universal aspects (such as an UML class diagram).

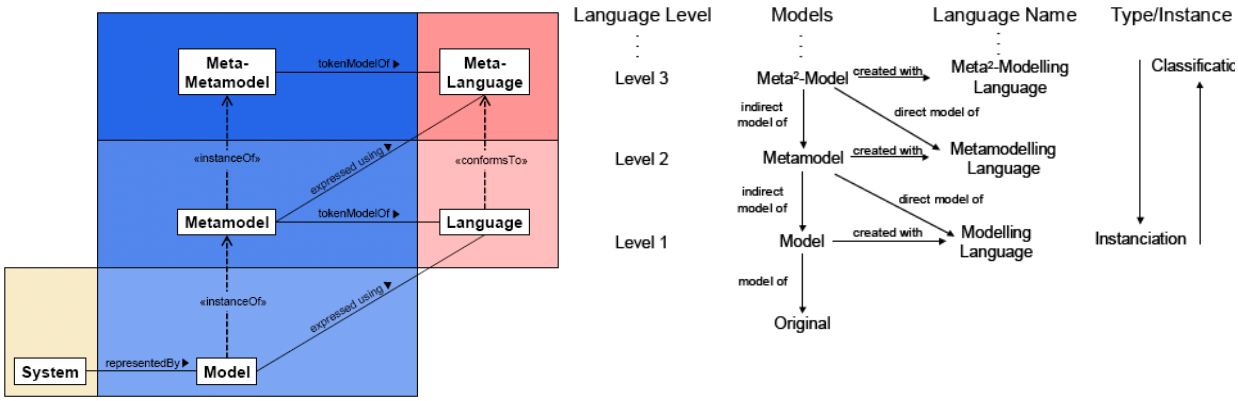

Figure 1: (meta-)models and (meta-)languages; (Kühne 2005) left, (Karagiannis & Kühn 2002), right

A (modelling) language is described by its syntax and its semantics whereas the syntax defines the elements and their allowed compositions (the grammar) and the semantics specify the interpretation of the syntactic expressions (e.g. that the syntactic element "+" denotes addition) (Harel & Rumpe 2004). In accordance with (Karagiannis & Kühn 2002; He et al. 2007), this work additionally distinguishes between abstract syntax and concrete syntax, whereas the concrete syntax of a graphical language defines is its notation. This work will actually deal with abstract syntax and semantics and - if not further qualified - the term "syntax" refers to the abstract syntax.

1.5.3 Enterprise Model

> "An Enterprise Model is a computational representation of the structure, activities, processes, information, resources, people, behaviour, goals and constraints of a business, government, or other enterprise. It can be both descriptive and definitional - spanning what is and what should be. The role of an enterprise model is to achieve model-driven enterprise design, analysis and operation." (Fox & Grüninger 1997; Fox & Gruninger 1998)

This definition puts a strong emphasis on the computational representation, what is also reflected when the authors (Fox & Grüninger 1997; Fox & Gruninger 1998) refer to enterprise models behind information systems. Although this work deals with computer interpretable models, the term itself is not interpreted such strictly in this regard. The form of representation and whether model is computable or not is rather seen as question of usefulness. Apart from that, the definition seems sound and is actually the only definition found.

1.5.4 Ontology

At least in relation with the languages to be considered in this work, the term "ontology" will occur (i.e. OWL, the web ontology language). Therefore, a short introduction will be given, before certain ontology languages will be discussed.

There are numerous definitions and several discussions about the term "ontology". (Guarino & Giaretta 1995) make a major distinction between "Ontology" as a philosophical discipline and several interpretations from the knowledge sharing community. The work at hand will not deal with the nature and organisation of reality. Thus, the definitions used in the area of knowledge representation will be rather relevant in this document. (Guarino & Giaretta 1995) propose the following interpretations of "ontology": either "a logical theory which gives an explicit, partial account of a conceptualization;" or "synonym of conceptualization" where "conceptualization" is interpreted as *"an intensional semantic structure which encodes the implicit rules constraining the structure of a piece of reality"*. Another in-depth discussion about the term is given by (Guarino et al. 2009) who in particular refer to the definition given by (Studer et al. 1998): *"An ontology is a formal, explicit specification of a shared conceptualisation"*. In this context, (Guarino et al. 2009) discuss among other things conceptualisation, formal and explicit specification and shared conceptualisation. (Corcho et al. 2004) also compared several definitions of the word ontology and concluded with some properties that differ from those previously mentioned by focusing on the characteristic of ontologies to aim at capturing consensual knowledge, being reused and cooperatively built.

In this work basically the definition by (Studer et al. 1998) is accepted whilst conceptualisation is understood according to (Guarino & Giaretta 1995). However, the notion slightly diverges with respect to "reality" and "shared": First, in this work an ontology is not considered to necessarily represent reality and second, the conceptualisation is not necessarily a shared one, even though it is not questioned that a common understanding of stakeholders is highly relevant. An adequate formal and explicit representation is assumed by using well defined languages like RDF(S), OWL and SWRL.

1.6 Chapter overview

Figure 2 provides an overview on the basic structure of this thesis. The chapters are shortly introduced hereinafter:

Chapter 1 introduces the topic and the research questions based on which the thesis is structured to a large extent.

Chapter 2 will introduce the research design which has been followed during this research work. Chapter 3 contains a literature review which influenced most of the subsequent chapters, what is indicated with the large arrow.

Chapter 4 presents the results of the interview conducted and in particular provides input for chapter 68.

Chapter 5 selects the applications and modelling languages considered as most important, what provides the answer to the first research question.

Chapter 6 presents the representation requirements with respect to the applications and answers the second research question.

Chapter 7 analyses the applications and discusses representations, leading to one part of the third research question. The results are used to iteratively refine the results of chapter 6.

Chapter 8 describes the development and implementation of a scenario based on the interview. This founds the second part of the third answer.

9 evaluates the results of the precedent chapters in order to answer the fourth research question. Based on this, the main research question can be answered.

Chapter 10 finally concludes about the work and its results and actually answers the main research question.

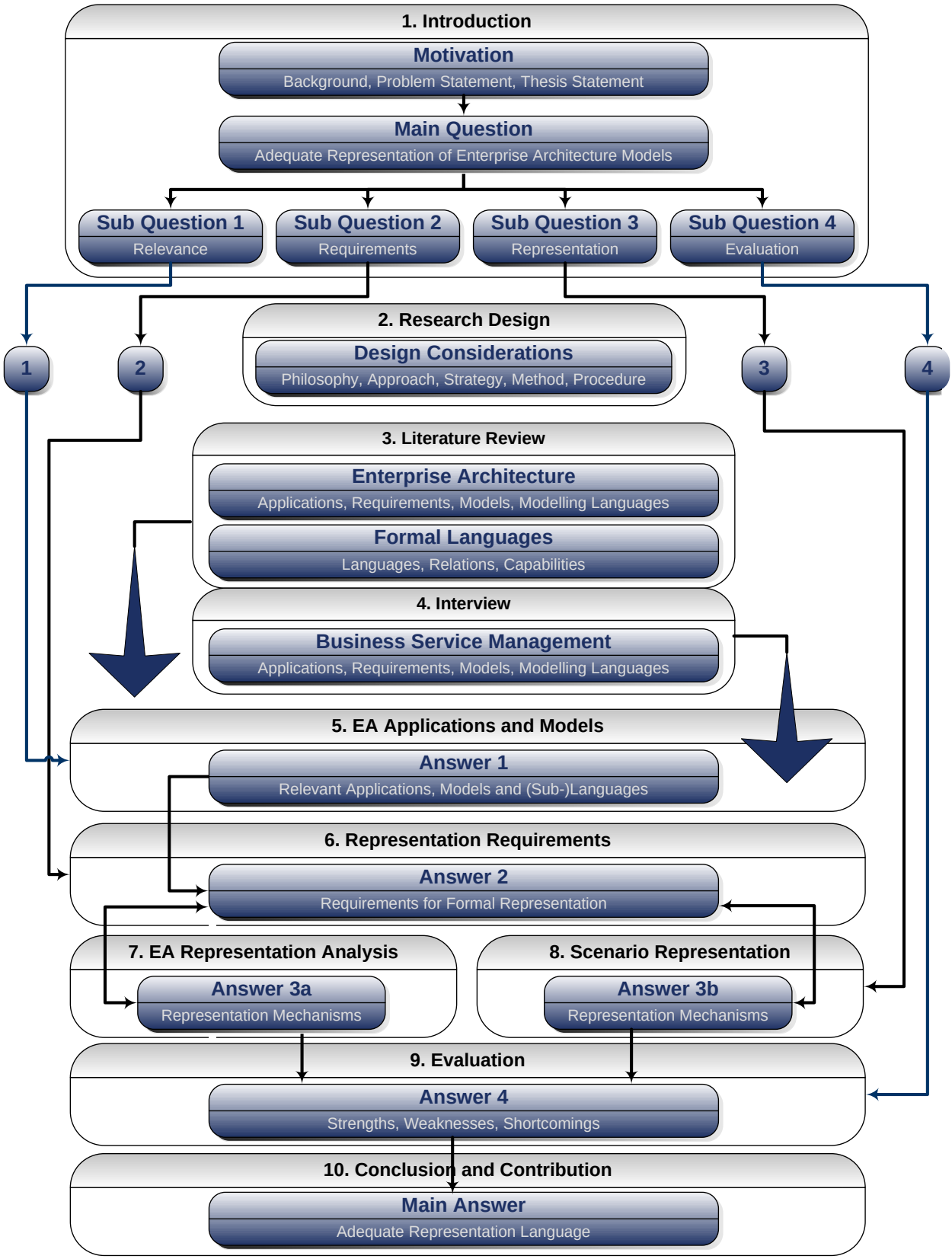

Figure 2: Chapter Map

2 Research Design and Methodology

This chapter first discusses the selection of techniques and procedures. The second part then concretely describes the application and procedure of the research based on the considerations of the first part. Additionally, the scope of the project is further specified and limitations are pointed out.

2.1 Design Considerations

The following sections discuss the selection of techniques and procedures based on which the research method will be designed consequently. The research onion (Saunders et al. 2009) depicted in Figure 3 builds the basis for this design process by working inwards through all layers. In accordance with Saunders et al. (2009: 109) the research question (and sub questions) will be the major consideration.

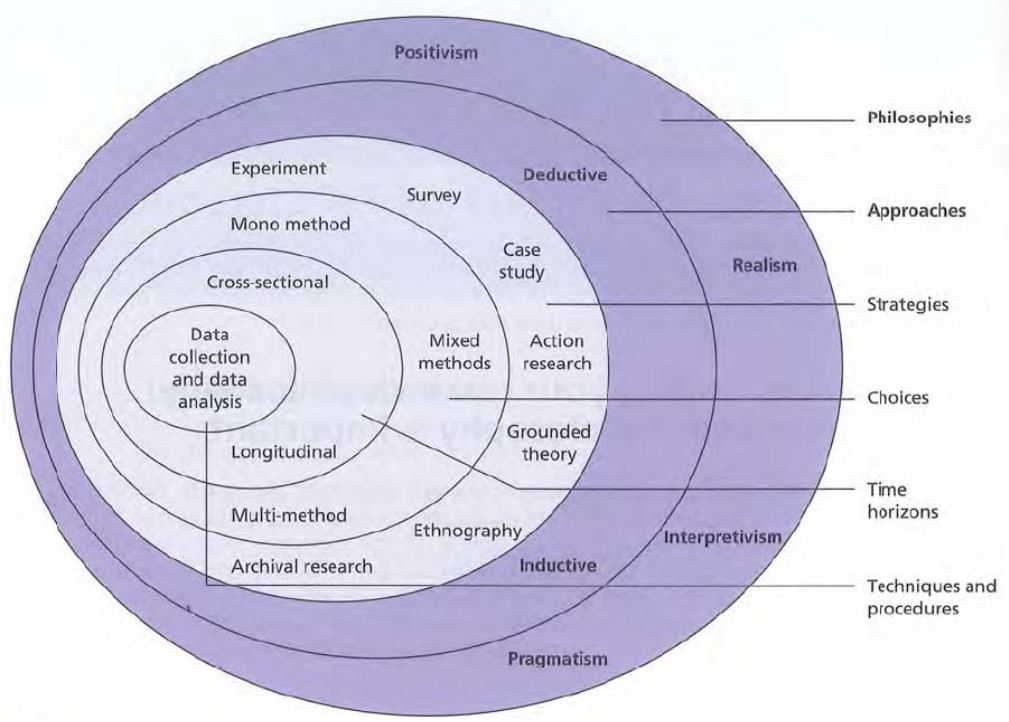

Figure 3: Research Onion (Saunders 2009)

2.1.1 Research Philosophy

Saunders et al. (2009: 119) compare the four philosophies positivism, realism, interpretivism and pragmatism with respect to ontology, epistemology and axiology. Guided by the research questions, none of the philosophies seem to match completely: The work has to determine the relevance of applications and models, to derive requirements for a formal representation and to develop corresponding representations. In the following paragraphs the explications of Saunders et al. (2009: 107-119) and additionally, the philosophical assumptions of design research according to Vaishnavi & Kuechler (2004) are discussed roughly with respect to the research questions.

Interpretivism emphasises on social phenomena and subjective points of view. Even though there may be social aspects related to some facts, the research questions predominantly deal with facts directly. For example, indicators of relevance may be influenced by social aspects, but it is not a goal to analyse these influences.

Realism is related to the perception of reality. Direct realism assumes that reality is adequately reflected as we observe it, whilst critical realists argue that additionally a mental process of social conditioning is required. With its objective view, the consideration of a context and the inter-

pretation of perceived facts, critical realism may be a suitable philosophy to find answers to the first and second research question (applications, models, requirements).

Positivism is also characterised by an objectivistic view and relies on observable phenomena and thus partially fits to this research. The philosophy focuses on measurable facts to detect causalities and to find generalisations in a value-free manner. These characteristics are especially desirable with respect to considerations of relevance.

Pragmatism follows the argument that the ontology, epistemology and axiology has to be most appropriate to answer the research question and thus, also mixed approaches and no distinct decision for either positivism or interpretivism is necessary.

Even if there are some useful characteristics of positivism and realism, the main research question cannot be answered by deriving knowledge from observed data. Observing the field and concluding about importance and requirements can be supported but how to represent models with the given languages and to conclude about possible and advantageous representations requires a more constructive philosophy.

Vaishnavi & Kuechler (2004) describe "philosophical assumptions" of design research including ontology, epistemology and axiology. Especially with respect to research question 3 (how the requirements can be fulfilled), these characteristics suit best:

Concerning ontological belief Vaishnavi & Kuechler (2004) point out that "Design research by definition changes the state-of-the-world through the introduction of novel artifacts". Therefore, in contrast to the positivist, design research does not base on a single world state. These world states in design research depend on the context. For this work, the broad context is enterprise architecture modelling and will narrow down to the level of requirements. From the epistemological view, "Knowing through making" seems adequate, as representation mechanisms may be constructed and tested on whether they behave as expected. Axiologically, Vaishnavi & Kuechler (2004) name values like creative manipulation, control and understanding. Design research has an iterative characteristic where, according to Vaishnavi & Kuechler (2004) even the research philosophy may change during the research process.

The research philosophy could consequently be seen as pragmatism, as the beliefs are guided by the research problems. Actually, the mentioned characteristics of design research as well as some beliefs underlying positivism and critical realism were applied.

2.1.2 Research Approach

On one hand, the main research question, what language was adequate to represent enterprise models, cannot be answered with a setting that covers the characteristics of a deductive approach mentioned by Saunders et al. (2009: 125), as for example the explanation of causal relationships of variables, the use quantitative data or an operational hypothesis test.

An inductive approach that, according to Saunders et al. (2009: 126-127), puts emphasis on understanding the context and human factors leading to an observable reality is rather suitable, at least with respect to the procedural view, where first the field (enterprise architecture and its context and purpose) has to be analysed to conclude about the importance of certain applications, languages and consequent requirements.

To determine the importance of applications and models quantitative data could be helpful, but actually, the project setting would probably not allow for selecting large samples and would not lead to a hypothesis testing. The results of the core research have to be tested but this is not seen as a statistical or mathematical proof but as development of language constructs which show the ability to fulfil the requirements.

From a more procedural view, a rather inductive approach seems adequate: First the field is analysed to discover applications, purposes and models and to conclude about their importance. Based on these results requirements can be derived. By constructing and evaluating solutions that fulfil the requirements the main research question can be answered.

2.1.3 Strategies

With respect to sub question 1 and 2, a case study strategy could be discussable, as case studies are suitable for understanding the context and would support the valuation of practical relevance. On the contrary, this work is not motivated by a certain organisation and to obtain information which allow for more generalisation, several case studies were beneficial. This is not seen as feasible in the scope of this project in particular for reasons of time as the strategy would mainly contribute to sub questions.

Action Research contains a solution finding process and therefore could be related with the main research question. The strategy is rather focusing problems in concrete cases, diagnosed, solved and tested in practice. But however, with emphasis on its iterative nature as described in Saunders et al. (2009: 147), it could support the synthesising character of the research questions.

In Design Research the knowledge emerges through making something (Vaishnavi & Kuechler 2004) and therefore was suitable to find representation mechanisms to answer the main research question. The steps (problem awareness, suggestion, development, evaluation and conclusion) of the general methodology with their possible iterations described in (Vaishnavi and Kuechler, 2004) and some of the outputs (constructs, models, methods, instantiations) may require a slightly adapted interpretation but seem most appropriate with respect to the main research question.

2.1.4 Choices

Basically qualitative data is of interest to answer the research questions. With secondary data from literature, language specifications and documentations and additional primary data, the research choice falls into what Saunders et al. (2009, 152) classify as multi-method qualitative study. Some quantitative information from secondary data sources is also used.

2.1.5 Time Horizon

Within the duration of the project no relevant changes with respect to the practical relevance of models, the applications and the specifications was expected. Thus, no longitudinal but rather a cross-sectional horizon has been selected.

2.2 Research Methodology

The following sections describe the actual research method with the used techniques and procedures in detail.

2.2.1 Data collection

In particular secondary data is used in this work but supplemented by valuable inputs from an interview.

1. Primary Data

An impersonal technique such as questionnaires comprises risks and difficulties with respect to the design of the questionnaire and the response. Interviews are personal and thus allow for discussing or broaching some subjects again or ask unplanned questions in the case of unexpected answers or inputs. Further on, by selecting an interview partner from the consulting domain, it was possible to gather information that reflects the experience of several years with several companies.

Therefore an interview with an expert of a company in the domain of enterprise modelling (business service management) was conducted. The questions focused on applications, goals and used models (i.e. why companies model, what they capture and how they use the models then). Thus, the result provided input for the requirements investigated in sub questions 1 and 2 but also for a concrete scenario.

Because language specifications and documentations were analysed and interpreted they can be regarded as primary data too. These sources provided specific requirements (sub question 2).

2. Secondary Data
Literature review founded the basis for the theoretical background of this project and provided basic input for the first and second sub question. Therefore, the following topics were covered and analysed:

- Enterprise Architecture (Frameworks)
- Ontology/Rule languages
- Applications of ontologies in the context of business modelling

2.2.2 General Methodology of Design Research

As described earlier, design research seems most adequate with respect to the main research question. The following paragraphs outline the general methodology of design research according to Vaishnavi & Kuechler (2004) before the adapted concrete procedure that guided this research is presented in the subsequent section.

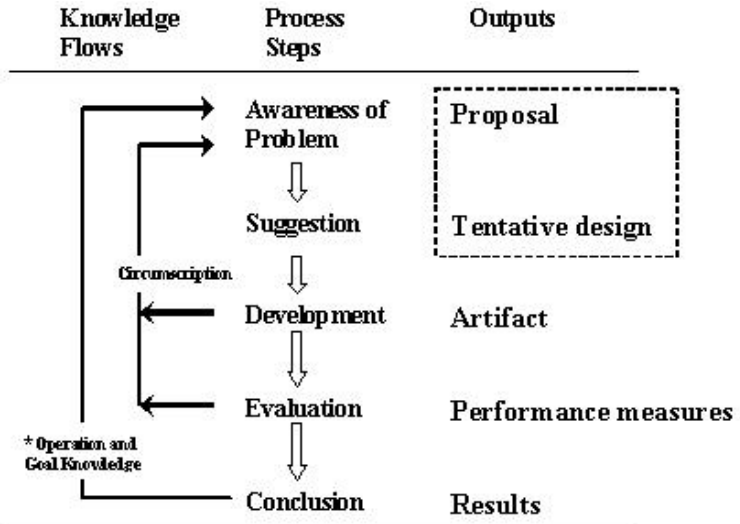

Figure 4: The General Methodology of Design Research (Vaishnavi and Kuechler, 2004)

- **Awareness of Problem**: Based on the awareness of a problem and possibly a recognised opportunity from other fields to apply to a problem a proposal for research effort is created.
- **Suggestion** is a creative step, where a solution for a problem is depicted. In this phase, new functionality and configurations are outlined. The result is a tentative design that may be sometimes part of a formal proposal of the previous step.
- In the **development** phase, the tentative design is implemented. The actual development depends on the resulting artifact and may be a formal proof or software, for example.
- **Evaluation**: The qualitative and quantitative criteria for evaluating the artifact may be already defined in the proposal. In this phase, hypotheses about the expected behaviour of the artifact are made and deviations are noted and tentatively explained. Results from the evaluation together with information from the construction phase may bring insights leading to modified hypothesis, further literature research and a new suggestion phase.
- **Conclusion** is the final phase where the development is considered as sufficient, even if the results still deviate from the expected behaviour. Results are consolidated and the gained knowledge is possibly categorised and written down, being it learned facts or subjects for further research.

Vaishnavi & Kuechler (2004) also describe an example of conceptual development which has characteristics comparable to this research. The first phase resulted in a concrete research

question that changed after working in subsequent steps. In the suggestion phase a high level concept emerged that was further detailed in the development phase resulting in a conceptual model. Whilst some micro-level evaluations where part of design decisions in previous phases, the evaluation phase comprised the creation of models as input for the execution of rules as simulation.

2.2.3 Procedure

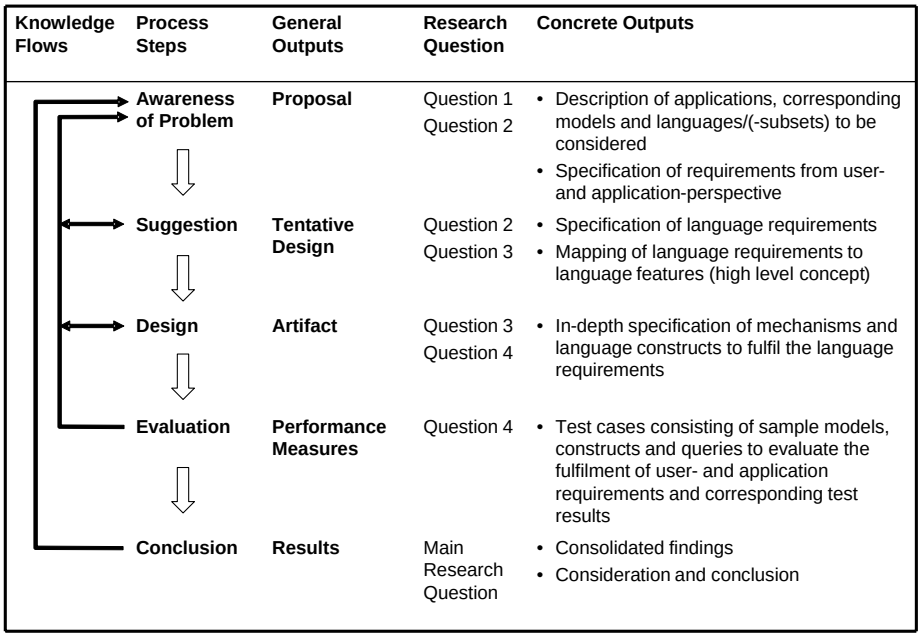

Figure 5: Design Research Procedure

- **Problem awareness**: In this research, at least the main research question was not desired to change. But actually, by answering the first research question, the boundaries for further research was defined, namely, which applications and models or modelling languages were further investigated. The same is true for the corresponding result. By defining the requirements, the ***actual problems became more concrete***. Therefore, these two research questions were elaborated in the first step to found the basis for the constructive and creative step of suggestion. Several application requirements became clear later during the design phase what led to iterations.
- In the **suggestion** phase mainly the third research question was dealt with. Whilst requirements resulting from language specifications are relatively concrete, more abstraction was necessary to come from general application requirements to certain requirements for a formal representation language. Because this step is thought of as more constructive, the definition of language requirements was assigned to this phase too, although it could be considered as part of the problem awareness. The most important result of this phase is denoted as "mapping" in section 6.4.
- The design phase mainly deepens the answer to the third research question. Whilst the suggestion mapped applications and application requirements to language constructs (or feature groups), the design developed a more comprehensive view by analysing coherences between requirements and representations.
- During the evaluation phase the interview scenario was implemented. This implementation proved that the designed artifacts were applicable - even if limitations were foreseeable. The main evaluation actually searched for an answer to the fourth research question based on the development phase.

- The procedure model as shown in Figure 5 explicitly allows for iterating back from any step to any previous one. The conclusion is rather thought of as final step of the research where the arrow to problem awareness represents inputs for future research.

2.3 Delineations and Limitations

As it largely becomes apparent in the thesis statement, the research questions and the research design, the project has several limitations and exclusions. The following sections focus on the limitations and consequences.

2.3.1 Research Data

Applications, purposes and requirements of modelling languages were determined mainly on basis of literature and language specifications. Additional input resulted from an interview.

- The assumptions with respect to practical relevance is therefore not representative in terms of frequencies or alike and is not necessarily applicable to any other country or company.
- An interview further on rather allows the valuation of things mentioned to be practically relevant than valuing something as irrelevant if it was not mentioned.

2.3.2 Language scope and coverage

- Only the most widespread model types were investigated (except the interview model).
  - Thus, neither the thesis statement nor the answers to the research questions are applicable to models in general.
- Only parts of the selected languages were analysed in order to determine the representative features which have to be formally represented, however, existing work has been considered.
  - Through the application focus, there are limits in the validity of the results regarding the modelling tool ATHENE: Whilst it was claimed, that ATHENE is capable of representing any process- data- or organisation structure model (Hinkelmann, Nikles & L. V. Arx 2007), this work concentrated on the aspect identified as practically relevant, thus not guarantees, that all languages may be completely represented (in the sense of complying with the complete specification).
- Only the named languages RDF, RDFS, OWL and SWRL were considered to find adequate representation formalisms.

3 Literature Review

The literature founds the theoretical framework of this work and provides major inputs for the development work. The chapter is divided into three main sections:

First, the topic of enterprise architecture is investigated with the main purpose of identifying the most relevant goals, applications and models (or modelling languages).

Second, the representation languages selected for this work, RDF(S), OWL and SWRL are discussed, related and compared.

Third, existing work regarding the use of semantic technologies (i.e. ontologies and rules) related to enterprise modelling is presented.

3.1 Enterprise Architecture and Frameworks

"There is no way to change automobiles, computers, integrated circuits, oil refineries, battleships, telephone networks, programs, Enterprises, or any other complex thing quickly (or

safely) without starting with the descriptive representations of the thing you want to change." (Zachman 1996)

3.1.1 Goals and Applications of Enterprise Architecture

On a high level, applications and corresponding requirements can possibly be derived from purposes and goals (as they are reached by using the architectural description and thus, are related to an application). Some authors also directly mention uses and applications.

Fischer et al. (2007) motivate Enterprise Architecture in that businesses face an increasing complexity of transactions, often changing business models, more regulations and increasing dependency from information technology. They highlight the acceptance as approach for managing change and business IT alignment by bringing changes from the strategy- and process-level down to the IT, by supporting business transformation and by decoupling Business- and Technology-Architectures. The goals of EA are reduced to three main ones: **documentation and communication** of as-is, **design** of as-is and project support for the **transformation** from as-is to to-be structures and processes.

Regarding the use of EA, Lankhorst (2009) refers to the IEEE 1471 specification which includes the following uses of architecture (IEEE 2000):

- **Analysis** of alternative architectures and business planning for transition to a new architecture.
- **Communications** among stakeholders (i.e. organizations involved in activities from development up to maintenance of a system) as well as between acquirers and developers as a part of contract negotiations.
- **Development and maintenance documentation**, including material for reuse repositories and training materials as input to subsequent system design and development activities.
- **Input to system generation and analysis** tools to support operation and infrastructure, configuration management as well as redesign and maintenance of systems.
- **Support for planning** and preparation of acquisition documents (e.g., requests for proposal and statements of work) and review, analysis and evaluation of the system across the life cycle.

Lankhorst (2009) also categorises several needs of companies regarding architecture:

- **Design:** Besides a common language, means such as methods, guidelines and best practices to support quality and support of decision- and change-tracking are needed.
- **Communication:** Visualisations should adequately represent the relevant aspects for different stakeholders but especially also comprehensibly support the communication of relations between different domains.
- **Realisation:** Higher level architecture models should be interlinked with more detailed models. With that, the realisation should be supported and results can flow back. Related to that, the use of different tools and concepts should be defined and integrated.
- **Change:** The consequences of change in one architectural domain have to be assessable to support the change and evolution of the enterprise (architecture).

Hanschke (2010) presents a long list of objectives and (qualitative) benefits of enterprise architecture management consisting of four categories (and stakeholder views), summarised in the following:

- **Design and optimisation** of the business architecture (from the view of business responsibles): e.g. transparency of structures and relations.
- **Business alignment of the IT** (view of the strategic IT management): e.g. having a common language, linkage between business and IT and input for business orientation.

- **Strategic planning and controlling** of the IT (view of the strategic IT management): e.g. overview of the IT landscape, strategic guidelines, technical standardisation.
- **Operative management and controlling** of the IT (view of the operational IT management): e.g. input for operational planning and performance- and supplier management.

Further noteworthy topics Hanschke (2010) addressed several times are compliance issues and the identification of potential improvements and needs for action. She also points out communication (i.e. founding on a common language) as important mean for business and IT alignment.

Infosys Technologies Ltd. conducted a survey among their customers in 2008 (Obitz & Babu K 2009) and analysed the responses from 173, predominantly large (>= 1000 employees) American (northern America 66%, 22% European) companies. With respect to the objectives, benefits as well as to the importance of frameworks and tools, the following results are considered as relevant to the work at hand:

- Business and IT alignment was the most mentioned objective, followed by the standardisation and improvement of business processes (both > 35%) and flexibility of business and processes (> 25%). Also above 20% were the following objectives: Simplify the technology and application portfolio, enable business transformation, reducing IT costs and reducing the time or risk to deliver IT projects.
- Regarding benefits, more than 30% answered that customer satisfaction improved. More than 25 % stated reduced IT costs as well as standardisation and improvement of business processes. Further important benefits indicated by 20% or more respondents were: Business- IT alignment, business & process flexibility, optimisation of value generated from IT investments and an improvement of the ability to exchange information between business units.
- The major contribution of enterprise architecture was perceived in IT strategy, enterprise technology standards, information architecture (enterprise data model and data integration architecture), technology roadmaps and business architecture (business goals and objectives and business processes).
- The most remarkable frameworks are TOGAF and Zachman, adopted by 32% or 25% of the respondents respectively. TOGAF even distinctively led against the federal architecture framework FEAF within the government sector. The remaining frameworks mentioned seem therefore rather insignificant: FEAF with 7%, DODAF with 5% and GERAM with a 1% adoption rate. In the field of IT Specific Frameworks, ITIL (47% of the respondents) and Cobit for governance (23%) are widely adopted.
- 60% of the analysed responding companies use specific EA tools, 44% one and 16% more than one tool. The reasons for not using EA tools are: licensing costs (>35%), sufficient capabilities of existing design tools (>30%), small IT portfolio doesn't justify a tool (>15%) and unaccomplished requirements (15%). Although the study identifies a market leadership of Framework Software and Telelogic System Architect (with unclear figures, as the percentage differs between the textual explanations and the graph), there is no market domination.

3.1.2 Tool Support for EA

Another indicator for practical applications and requirements is the functionality of enterprise architecture tools. Tool providers basically have to meet market needs and, at least when the product is coupled with consultancy services, are close to their customers, the enterprises. Therefore it seems worthy to consider the capabilities of tools in the market of enterprise architecture. An overview on the tools and capabilities is presented hereinafter.

Lankhorst (2009) points out the following reasons for tool support in enterprise architecture:

- The power to move towards standardisation of languages and practices (if a tool is used company-wide),

- Supporting consistent and correct models (if a tool supports automated checking of constraints and architecture principles)
- Support in the application and reuse of architecture patterns
- Comparison of alternatives (e.g. as-is and to-be) and analysis of models
- Support of migration paths (e.g. from as-is to to-be)

Lankhorst (2009) classifies architecture analysis techniques according to four aspects: (a) quantitative (e.g. with respect to time and costs of performing things), (b) functional (structure and behaviour), (c) simulation (simulate the execution) and (d) analytical (such as critical path or execution time). Both, the quantitative (a) and the functional (b) analysis may be performed with either with simulation (c) or with analytical techniques (d) (Lankhorst 2009).

There are numerous tools and the magic quadrants in Figure 6 and **Figure 7** only contain an excerpt of the existing tool market. As minimal requirements for EA tools, (Handler & Wilson 2009) mention the creation or import of models and artefacts, the presentation of repository information graphically, textually and in executable form and namely the support of BPMN, BPEL, ERD, DDL and UML. Additionally, "[…] metamodel(s) that support often-changing relationships between objects within and between myriad viewpoints or architectures, as well as capturing temporal relationships and changes". Further on, administrative capabilities are required with respect to e.g. security, audit and versioning.

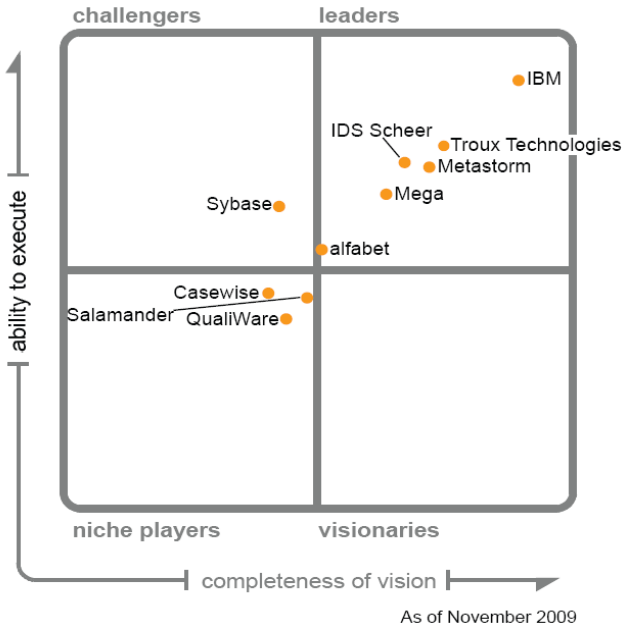

Figure 6: Gartner Magic Quadrant 2009 (Handler & Wilson 2009)

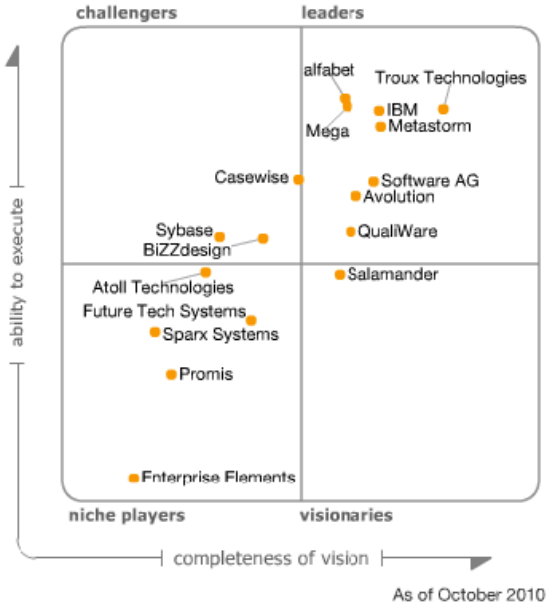

Figure 7: Gartner Magic Quadrant 2010 (Wilson & Short 2010)

The Institute for Enterprise Architecture Developments (IFEAD)[3] published a tool overview and an EA tool selection guide (Schekkerman 2009) showing that the focus of the tools and the corresponding functionalities are widespread, this is also reflected in the candidate requirements for tool evaluation listed in the selection guide. Besides operational and technical requirements (such as the supported platform, performance or security), support requirements (e.g. training and documentation), methodical requirements (supported frameworks and languages) and so on, several concrete functional requirements which reflect the uses of EA modelling are contained. The guide does not indicate a general importance of requirements and it does not explain in which cases or situations certain requirements were important. However, several requirements from (Schekkerman 2009) seem to be of interest with respect to the identi-

[3] http://www.enterprise-architecture.info (Accessed on 2010-11-20)

fied goals and applications and may relate to the expressiveness of the language representing the corresponding information - the following list condenses them:

- **Reference models and design patterns**
    - Management of enterprise architecture design patterns: A system may store and allow for searching and proposing design patterns to be applied.
- **Analysis of models**
    - To align different areas of an enterprise and to improve the architectures, the following functional requirements seem to be of importance:
        - Detect the impact of changes and thus enable the representation of relations between elements and models throughout all architecture domains.
        - Check whether a model (or the whole architecture) complies with architecture principles and modelling language specifications and whether the model is semantically correct and consistent with other models.
        - Analysing bottlenecks to identify need for action.
- **Model Definition**
    - Having a common meta-model, marking object timelines (e.g. validity period), handle different stages of objects (e.g. discussion, valid, operation), produce time-related output (e.g. models at a certain time).
- **Simulation**
    - Processes, alternative scenarios, landscapes, impact etc.
- **Models and Languages**
    - Common meta-model, diagrams of different domains, customisation, meta-modelling and reuse.
- **Version Management**
    - E.g. versioning of objects, compare versions, conflict handling and merging.
- **Code Generation**
    - Ability to generate code in different languages.

(Hysom 2003) describes the "problem space" of an enterprise as a set of cubes related to each other. Each cube has several dimensions. The primary one, the "Enterprise Cube", for example has the dimensions structure, behaviour and value which are considered as further cubes. The structure cube corresponds to the objects of the enterprise, where they are located and how they are connected to each other. The behaviour cube examines the states of objects, their interaction and the behaviour and interaction over time. The value cube deals with the capabilities of objects, their costs and risks. Further on, there are the following cubes: The perspective cube considers the objective reality, the perceived and the expected reality. The lifecycle cube spots the stages of an enterprise, e.g. as-is, should-be and to-be. Finally, the knowledge cube questions the knowledge about the enterprise and the mentioned aspects of the cubes with respect to predictability, precision, certainty and completeness. Hysom (2003) further depicts roughly a generic process of enterprise modelling and five capability assessment levels to finally support an enterprise to select appropriate enterprise modelling technology. The generic process consists of seven stages: 1. Capture & represent the as-is and could-be as well as measures and (target-) criteria. 2. Store & access models. 3. Visualise and understand static and dynamic aspects of the enterprise. 4. Analyze & design, e.g. by means of simulation, animation, behaviour and problem analysis. 5. Verify & validate in terms of e.g. traceability of models and metrics of behaviour. 6. Evaluate & select improved versions. 7. Implement & monitor changed and newly deployed systems. The capability levels range from 0 (No capability) to 4 (World class capability). Based on this process and aspects, Hysom (2003) defines several criteria and correspond-

ing capability levels. The criteria are presented hereinafter in a shortened form, omitting the capability levels:

- **Capture and Represent**
    - Represent models in different stages of the life-cycle (e.g. separate or combined with transitions between as-is and should-be). Enter, capture, import and link (external) data. Support of determining the necessary completeness of knowledge in models. Represent or determine predictability and precision of enterprise knowledge and represent or calculate certainty of enterprise knowledge. Model (changed) entity locations combined with traversing times. Model entity states (e.g. distinguish states, defining state sequences, quantitative statements about entities occupying states), interactions (representation, conditioning, linking and quantifying interactions), temporality (sequencing, durations, temporal relationships, temporal quantification), capability (entity capabilities, levels of capabilities, capability impact through interaction, quantifying capability in life-cycle), and cost (unit costs, deriving costs through interactions, cost impacts, cost throughout the lifecycle). Model enterprise risk (e.g. probabilities, threats, and impacts). Perform model abstraction (abstract models to higher-level model) and refine abstracted models into a set of lower-level models.
- **Store and Access**
    - Consistent maintenance of data in terms of e.g. (automated) checks of data types and common glossary. Partitioning and segmentation (e.g. create and track hierarchy or criteria-based extracts of models). Identification of conflicts between different models and resolution support. Support to integrate different models into a common model
- **Visualize and understand**
    - Tailor the visual representation of entities and relationships. Model editing functionality such as copy, paste, move, zoom, hide, layout guidelines. Visualisation of model behaviour.
- **Analyse and design**
    - Construction of an analysis model and support of static analysis such as integrity, consistency, connectivity, critical path and user-defined or tailored analysis. Dynamic analysis for diagnosis such as calculation, simulation, inferencing, animation and user-defined or tailored analysis. Discovery of causes for behaviour. Configuration of models such as selection of a model subset to meet certain requirements. Support of design approaches to meet behavioural requirements. Support of adjustment approaches to improve behaviour. Support of optimisation of model behaviour.
- **Validate and Verify**
    - Assessment of formalisms to meet modelling goals. Identify and quantify the applicability of formalisms for certain situations in a specific enterprise. Support to verify models to their data sources manually (e.g. review comments) or automatically (e.g. parameter verification check). Support manual or automatic validation of the model structure compared to the actual enterprise. Support of validating the model behaviour against the actual enterprise.
- **Evaluate and Select**
    - Capability to define metrics and parameters to assess added value of a model. Support of model evaluation approaches. Support of displaying evaluation metrics in a model. Support of model improvement based on evaluation results.
- **Implement and Monitor**

- Support for implementation of to-be models such as generation of workflow instructions, software code and configurations. Support for collecting behavioural data and (real-time) measurements. Support for attributing models with real-time data to exercise behaviour.

- **Manageability Assessment Criteria**
  - Distinguishing specific modelling capability requirements (e.g. based on the organisation). Identification of areas to improve modelling. Support the definition of modelling standards. Comparison of models to standards (model consistency). Configuration support such as versioning and management of alternatives.

- **Usability Assessment Criteria**
  - This includes e.g. the tool distribution, learning curve and documentation.

- **Deployability Assessment Criteria**
  - This includes platform criteria, supported size of models, tool versions regarding capability, maturity, compatibility, training, support and price, but also adjustment of modelling capabilities such as adding types, attributes and functionality.

3.1.3 EA Models

In accordance to Lankhorst (2009), Hanschke (2010) points out the importance of representing information according to stakeholder needs. As frequently used visualisations in enterprise architecture management she mentions lists (e.g. analysis results with properties), mapping tables (e.g. to model dependencies), and several portfolio-, cluster- and landscape-diagrams. According to Lankhorst (2009), UML is the dominant language for software, whilst for the organisation and process models a multitude of languages exist. Lankhorst (2009) actually argues for a specific language for enterprise architecture complementing more detailed languages, namely he introduces the ArchiMate language, whose specification is meanwhile published by the Open Group[4]. Reasons are that an architecture language should be precise and have a formal foundation whilst being comprehensible for non-experts, further being capable of representing domain structures and relations between domains and allowing for different views on the same model. With respect to the argument of having commonly understandable models, Bernus (2003) rather expects a tool to support different languages and to translate between them.

3.1.4 EA Frameworks

With respect to their adoption in companies, the most important EA frameworks identified so far are the Zachman framework and TOGAF. The following sections provide a brief overview on them and a comparative view on further frameworks often referred to in corresponding literature.

Zachman

According to (Lankhorst 2009), the framework introduced by John Zachman (Zachman 1987) as "Framework for Information Systems Architecture" is the first and best-known EA framework. Also (Hanschke 2010) considers it as headstone for further approaches and ideas. Zachman (1987) described different perspectives and aspects of information systems architecture in comparison with classical architecture of buildings. In (Sowa & Zachman 1992) the work was refined and the commonly known perspectives (Planner, Owner, Designer, Builder and Sub-Contractor) and aspects (What, How, Where, Who, When and Why) as shown in Figure 8 were presented.

(Lankhorst 2009) points out, that the framework was easy to understand but also that the number of cells was a problem for applicability and that the relations between the cells could be defined better. (Hanschke 2010) remarks the missing methodology and lacking tool support for the introduction.

[4] ArchiMate Forum of The Open Group: http://www.opengroup.org/archimate/ (Accessed on 2010-11-20)

	Data *What*	**Function** *How*	**Network** *Where*	**People** *Who*	**Time** *When*	**Motivation** *Why*
Scope (Context) *Planner*	List of Things Important to the Business	List of Processes the Business Performs	List of Locations in which the Business Operates	List of Organisations Important to the Business	List of Events/Cycles Significant to the Business	List of Business Goals/Strategies
Business Model (Concepts) *Owner*	e.g. Semantic Model	e.g. Business Process Model	e.g. Business Logistics System	e.g. Work Flow Model	e.g. Master Schedule	e.g. Business Plan
System Model (Logic) *Designer*	e.g. Logical Data Model	e.g. Application Architecture	e.g. Distributed System Architecture	e.g. Human Interface	e.g. Processing Structure	e.g. Business Rule Model
Technology Model (Physics) *Builder*	e.g. Physical Data Model	e.g. System Design	e.g. Technology Architecture	e.g. Presentation Architecture	e.g. Control Structure	e.g. Rule Design
Detailed Representations (Out-of-context) *Sub-Contractor*	e.g. Data Definition	e.g. Program	e.g. Network Architecture	e.g. Security Architecture	e.g. Timing Definition	e.g. Rule Specification
Functioning Enterprise	e.g. Data	e.g. Function	e.g. Network	e.g. Organisation	e.g. Schedule	e.g. Strategy

Figure 8: Framework for Enterprise Architecture (according to zachmaninternational.com[5])

TOGAF

The TOGAF framework (The Open Group 2009) documentation as depicted in Figure 9 consists of six parts (plus the introduction) or four main components respectively:

- The Architecture Capability Framework provides a set of references regarding organisational structures, processes, roles and responsibilities to establish architecture capabilities in a company.
- The Architecture Development Method (ADM) describes an iterative process following a development cycle of eight phases: A) Initial phase including the identification of stakeholders, definition of scope and the actual creation of an architecture vision. B) Business Architecture development including business processes, use-cases and class models (i.e. logical data models) for example. C) Information Systems Architecture development including the identification and definition of applications (in terms of capabilities) and data to support the business architecture. D) In the Technology Architecture phase, the physical realisation of the architecture is defined by mapping components of the Information Systems Architecture to technology components. E) The Opportunities and Solutions phase is concerned with the implementation of the target architecture and comprises the review of previous phases, the derivation of transition architectures and a migration strategy. F) In the Migration Planning phase the Implementation and Migration is planned in detail in terms of corresponding projects and their co-ordination and prioritisation. G) The Implementation Governance is concerned with the governance of the overall development and implementation process (e.g. ensuring compliance with the target architecture). F) The Architecture Change Management phase comprises the as-

[5] Zachman Framework for Enterprise Architecture: http://zachmaninternational.com/2/production/C4/downloads/Zachman_Framework.pdf (Accessed on 2011-02-05)

sessment and governance of the implemented architecture and managing changes in terms of new development cycles.

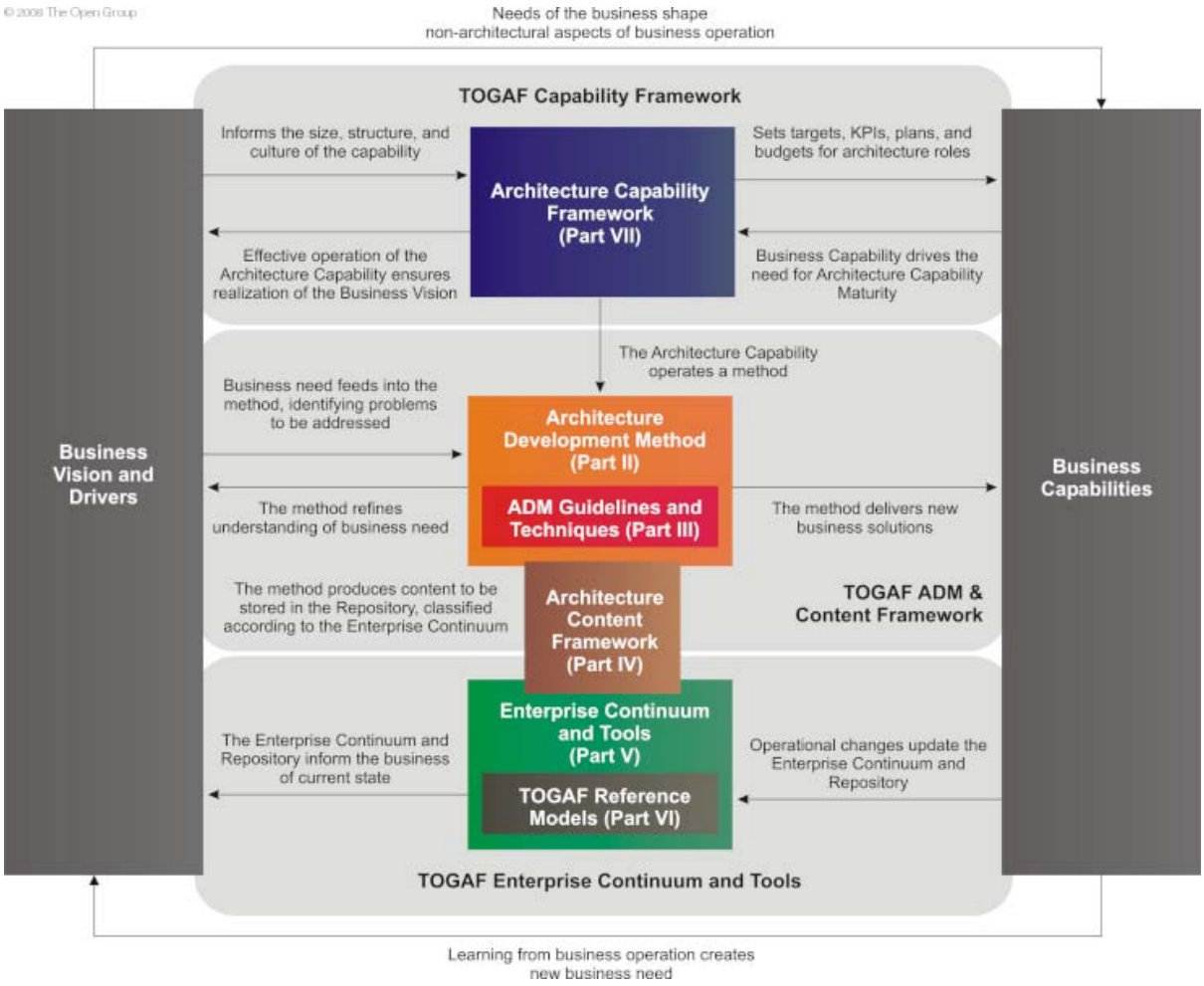

Figure 9: Structure of TOGAF 9 (The Open Group 2009)

- The Architecture Content Framework describes the TOGAF meta-model that structures the different contents of the framework such as architecture principles, requirements and realisation as well as the three architectures mentioned in the ADM cycle: The Business architecture which addresses facts regarding motivation, organisation and functions. The Information Architecture consisting of facts regarding data and application and the Technology Architecture. Besides possible deliverables from architecture projects, TOGAF further describes concrete artefacts which may be created throughout the ADM phases. These include numerous catalogues (e.g. principles, organisations, actors, roles, contracts, applications, etc.), matrices (e.g. System/Technology, Stakeholder Map, Actor/Role) and a multitude of core- and extensional diagrams (e.g. value-chain, business-footprint, functional decomposition, class diagram, application- and user locations).
- Enterprise Continuum and Tools addresses enterprise repositories with respect to classification and re-use of artefacts and characteristics to partition architectures. Further, a set of evaluation criteria to select architecture software tools is provided, comparable to the criteria presented in section 3.1.2. In addition, some reference models are provided, mainly in the form of categorisations of architectural elements (i.e. taxonomies).

Overview on further EA Frameworks

Figure 10 shows the comparison of seven EA Frameworks of (Franke et al. 2009) based on their proposed framework for classifying EA frameworks according to different user goals and

needs, called EAF^2. The literature references of (Franke et al. 2009) indicate that not the latest specifications were considered as available at the time of this work and the work at hand further considers Archimate, as described in (Lankhorst 2009), as modelling language, rather than as an EA Framework. However, the classification is even though regarded as good overview on the focus areas of the contained frameworks.

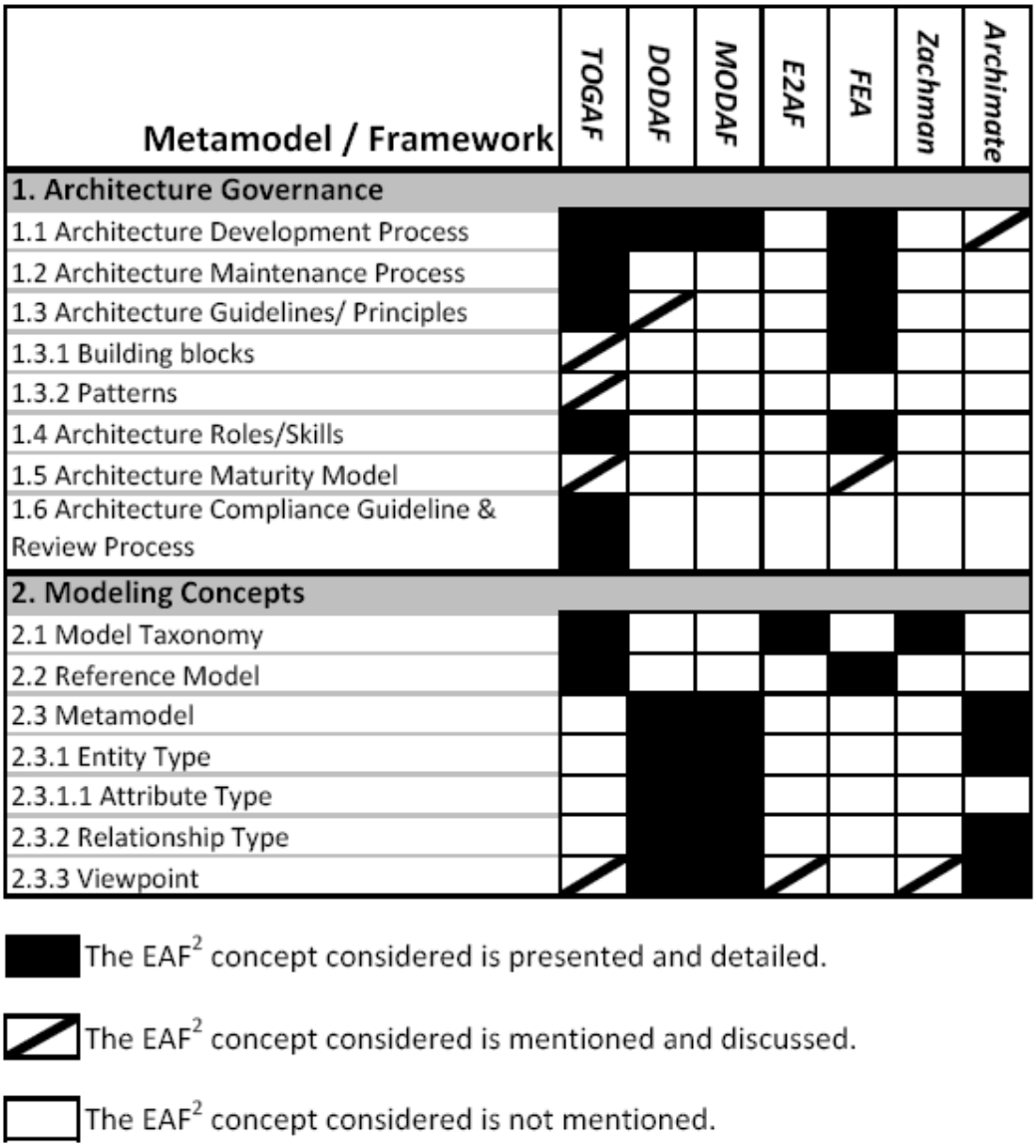

Figure 10: Classification of EA Frameworks (Franke et al. 2009)

A comparison of a few frameworks regarding aspects, perspectives and lifecycle-phases is provided in (Urbaczewski & Mrdalj 2006) but not presented here, because several frameworks are under continuous development and thus, an up-to date comparison seems to be of importance. Instead, Figure 11 shows a comparison regarding the perspectives only.

plugIT	*Zachman*	*BPMS*	*ARIS*	*TOGAF*	*OMG*	*PROMET*
Strategy	Scope	Strategy	Conceptual Model			Strategy
Business	Business Model	Re-Engineering		Business Architecture	CIM	Processes
System	System Model	Resource Allocation	IT Concept	Information/ Application Architecture	PIM	Information Systems
Technology	Technology Model	Workflow	Implemen-tation	Technology Architecture	PSM	
	Detailed Model	Performance Evaluation				

Figure 11: Related perspectives of different Frameworks (Hinkelmann et al. 2010)

The frameworks contained in the two comparisons presented are briefly introduced in alphabetical order (except the ones already described):

- ARIS: Architecture of Integrated Information Systems (ARIS) comprises a method, models and a software suite and is with that comparable to BPMS. ARIS comprises five views as represented in Figure 12 (Organisation, Data, Control, Function and Product/Service) and three perspectives for each (requirements, design and implementation - corresponding to the conceptual model, the IT concept and the implementation in other publications). The models used within ARIS include the well known EPC (Event Driven Process Chains) as well as Application System Type diagram, Function Trees and Function Allocation Diagrams (involving resources to execute functions), Entity Relationship Models, Knowledge Maps, Knowledge structures, Organisational Charts (Davis 2008).
- BPMS: The Business Process Management Systems approach as shown in the comparison of Figure 11 refers to the approach presented in (Karagiannis 1995) and implemented in ADONIS[6]. Besides ADONIS for business process management, there evolved further tools such as ADOscore (strategy and performance management), ADOlog (supply chain management) and ADOit (IT Management). With that, the BOC Management Office[7] covers a broad range of models such as goals, measures, processes, organisational aspects, services and applications.
- DODAF: The Department of Defense Architecture Framework (DoDAF)[8] is currently available as Version 2 and defines several views (i.e. capability, data and information, operations, projects, services, standards and systems), a meta-model and an architecture development methodology.
- E2AF: This framework refers to the E2A[9], "Extended Enterprise Architecture" presented by the IFEAD (Institute For Enterprise Architecture Developments). The corresponding site presents a number of documents (i.e. guides) for enterprise architecture, comprising EA validation, deliverables, tool selection, view points, maturity and assessment.

[6] ADONIS BPMS: http://www.boc-group.com/products/adonis/ (Accessed on 2011-02-05)

[7] BOC Management Office: http://www.boc-group.com/products/ (Accessed on 2011-02-05)

[8] DoDAF: http://cio-nii.defense.gov/sites/dodaf20/index.html (Accessed on 2011-02-05)

[9] Extended Enterprise Architecture (E2A): http://www.enterprise-architecture.info/Images/Extended%20Enterprise/Extended%20Enterprise%20Architecture.htm (Accessed on 2010-11-16)

- FEA: The US Federal Enterprise Architecture[10] provides guidance and templates for architecture development, reference models an assessment and a transition framework.
- MODAF: The MOD Architecture Framework was developed by the Ministry of Defence (MOD)[11], originally based on DODAF.
- OMG: The framework is actually known as MDA (Model Driven Architecture). (Lankhorst 2009) predicts, that the latest developments on the CIM (Computation Independent Model) level will raise MDA's level of relevance for EA to the level it has for software development, i.e. with platform independent and platform specific models (PIM and PSM).
- PROMET: The comparison refers to a method of process development (PROMET BPR) described in (Österle 1995).

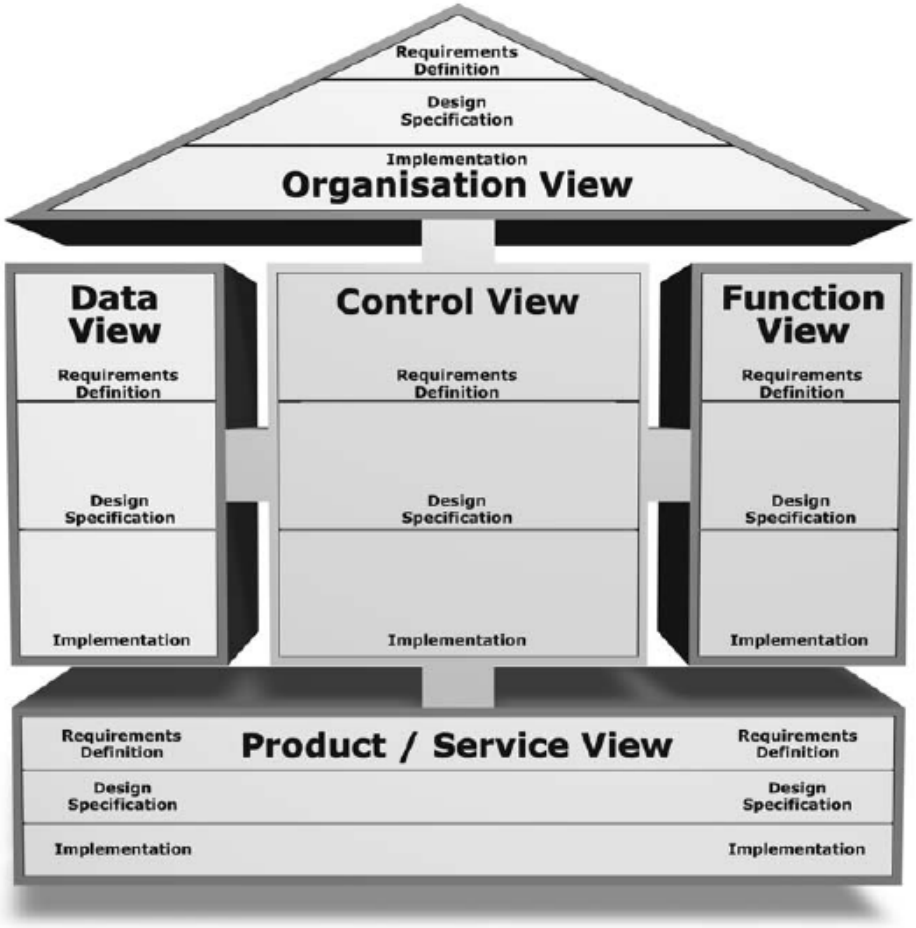

Figure 12: ARIS House (Davis 2008)

Models of Different Aspects and Perspectives

In (Hinkelmann et al. 2010) a framework for organising enterprise models is described. The framework is derived from enterprise architecture frameworks and is presented in Figure 13 in order to give an overview on possible models used for enterprise architecture. (Hinkelmann et

[10] Federal Enterprise Architecture (FEA): http://www.whitehouse.gov/omb/e-gov/fea/ (Accessed on 2011-01-30)

[11] MOD Architecture Framework: http://www.mod.uk/DefenceInternet/AboutDefence/WhatWeDo/InformationManagement/MODAF/ (Accessed on 2010-12-02)

al. 2010) point out that there is not necessarily a model each cell, for example for products at system level. Considering the examples mentioned in the previous section for ARIS, TOGAF and BPMS it also becomes clear that different architecture methods could lead to diverse representations.

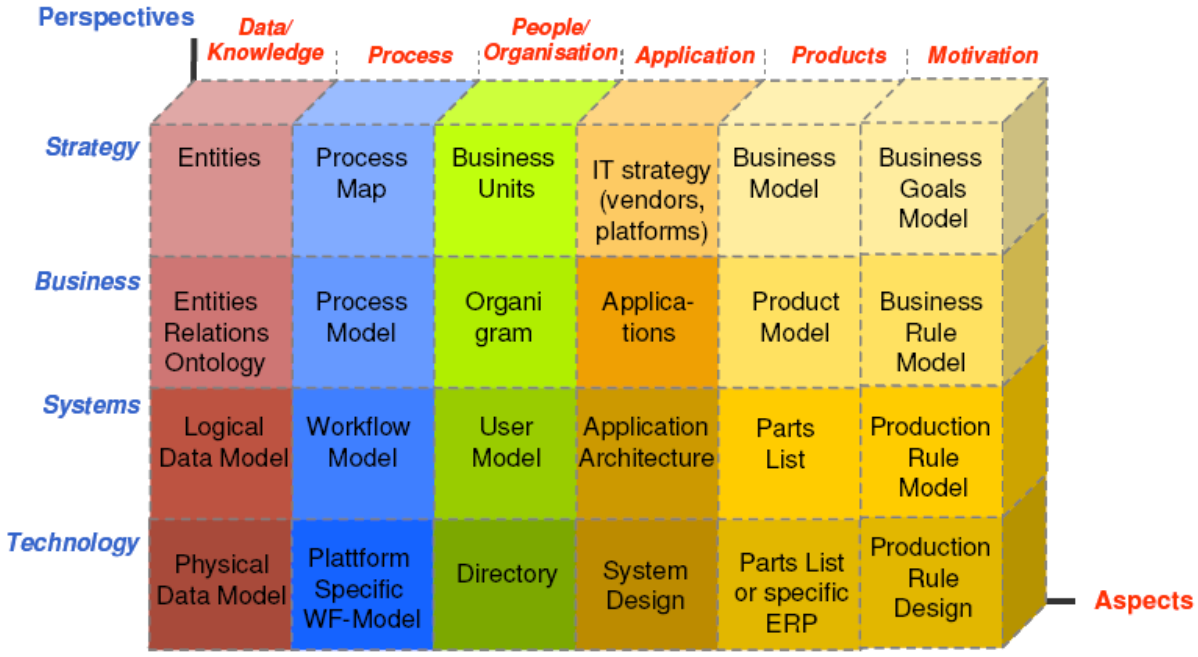

Figure 13: Models associated to perspectives and aspects (Hinkelmann et al. 2010)

3.2 Ontology- and Rule-Languages

Detailed descriptions on the evolution of ontology languages can be found in in (Corcho et al. 2004) and in (Baader, Horrocks, & Sattler, 2009: 23-25) focusing on description logics. This work focuses on specific languages and therefore the subsequent sections will concentrate on those, giving an overview of each and discussing their relations. Some information on further languages is given in the context of ontology based approaches in section 3.3. In order to compare the language features and also to point out some limitations an overview based on expressive means with SBVR[12] is given finally.

3.2.1 Resource Description Framework (RDF)

In its first official specification in 1999, RDF focused on metadata on web resources and was later intended to build the basic representation format for more complex languages (Hitzler et al. 2008). The latest version of RDF is specified as W3C recommendation in six documents as of 10 February 2004 (Manola & Miller 2004) including the RDF Vocabulary Description Language RDFS.

A basic idea behind RDF is to identify things by means of a Uniform Resource Identifier (URI) reference and to describe resources with properties and property values (Manola & Miller 2004). The key concepts of RDF according to (Klyne & J. J. Carroll 2004) are a graph data model, an URI-based vocabulary, datatypes, literals, an XML serialization syntax, expression of simple facts and entailment. A graph is a set of triples, each consisting of a subject (RDF URI Reference or blank node), a predicate (RDF URI Reference) and an object (RDF URI Reference, literal or blank node).

The vocabulary consists of syntax names (RDF, Description, ID, about, parseType, resource, li, nodeID, datatype), class names (Seq, Bag, Alt, Statement, Property, XMLLiteral, List), property

[12] Semantics of Business Vocabulary and Rules (SBVR): http://www.omg.org/spec/SBVR/1.0/

names (subject, predicate, object, type, value, first, rest, _*n*) and resource names (nil) (Becket 2004). These vocabulary elements are described in the following sections according the specification (Hayes 2004; Becket 2004; Manola & Miller 2004).

Syntax Names

RDF is the root node of a document, normally also used to declare the used namespaces. With "Description" nodes are declared, "ID" and "about" declare RDF URI references for nodes (e.g. relative to the base URI - xml:base - rdf:ID="name" is equal to rdf:about="#name") and "ID" can also be set on a property to identify the triple it defines for reification. The "parseType" attribute on a property element may take several values such as "Literal" (parseType="Literal") to embed XML content into RDF (as object of a triple), "Resource" (parseType="Resource") to omit blank nodes or "Collection" (parseType="Collection") allowing the property to contain multiple object nodes. The "resource" attribute declares an URI reference and can be used on a property to refer to the object node instead of defining the node (with rdf:Description & rdf:about) if it has no predicates. "resource" is also used to identify the resource in list elements (rdf:li or rdf:_1 … rdf:_*n*). The "nodeID" attribute is used to declare an identifier for a blank node to refer to it again. Declaring the "nodeID" means to replace the "about" attribute on a node or "resource" on a property. Finally, the attribute "datatype" can be set on a predicate to declare the datatype of the corresponding property value.

Class Names

Seq, Bag and Alt are container classes containing a sequence (Seq), an unordered list (Bag) or a set of alternative values (Alt). "Statement" is the class of RDF statements, Property that of properties, "XMLLiteral" the class of literal values and "List", in contrast to the container classes can specify an exact set of items in a collection.

Property Names

The property names are, together with the class "statement" part of the reification vocabulary. Reification is used to make statements about statements. Thereby a statement is defined by defining a subject, predicate and object for a resource. An example of using reification is given by (Hitzler et al. 2008): if a detective suspects the butler for having murdered the gardener this may require the representation of a "theory" which is not actually a fact, as it is not proven (that the butler was the murderer). Thus a reification statement could define the theory (Hitzler et al. 2008): ex:theory rdf:subject ex:butler, ex:theory rdf:predicate ex:murdered, ex:theory rdf:object ex:gardener. The suspection could then refer to the theory (ex:detective ex:suspects ex:theory), whilst there is no assertion about the validity of the theory.

The "type" property declares a "typed node". This will be further discussed in the context of RDFS. Instead of "rdf:type", also a typed node can be declared (e.g. <ex:lion rdf:about= http://example.ch/animal"/> is equal to <rdf:description rdf:about="http://example.ch/lion"><rdf:type rdf:resource="http://example.ch/animal"> </rdf:description>).

The property "value" is used to define the main value of a structured value. For example the weight of a good could be represented by a structured value consisting of the "rdf:value" represented by a number and a value indicating the unit, such as kilograms.

The properties "first" and "rest" are used in collections to refer to either a collection item node (first) or a further node (by "rest") that will refer to either the next collection item or to the finishing node ("nil"). In containers (which are not terminatory), the "_n" numbering (e.g. rdf:_1, rdf:_2) is used as container membership property.

Resouces Names

As described in the previous section, collections define a fix set of elements. The last element of a collection is therefore a closing node "nil", an instance of rdf:list that represents an empty list.

3.2.2 RDF Schema (RDFS)

RDFS is the RDF vocabulary description language and semantically extends RDF (Brickley & Guha 2004). RDFS consists of the following elements of the http://www.w3.org/2000/01/rdf-schema# namespace, referred to by the prefix "rdfs" (Brickley & Guha 2004):

Classes

The set of instances (members) of a class is called the extension of a class. A class can be an instance of itself and a member of its own class extension.

rdfs:Resource is an instance of rdfs:Class but all other classes are subclasses of it. It is also described as "the class of everything" and all things described by RDF are instances of this class. rdfs:Class declares class resources and is an instance of itself. Also the property class of RDF (rdf:Property) is an instance of rdfs:Class. rdfs:Literal is a subclass of rdfs:Resource and an instance of rdfs:Class and defines the class of literal values and may be typed. The class of datatypes is rdfs:Datatype, an instance and subclass of rdfs:Class. Instances of rdfs:Datatype are subclasses of rdfs:Literal, this is also true for rdf:XMLLiteral, which is an instance of rdfs:Datatype.

Properties

RDFS does not define a super property of all properties but several instances of rdf:Property. These are explained shortly hereinafter, omitting those already mentioned in the RDF section, such as rdf:type.

With rdfs:range the values of a property can be declared to be instances of one or more classes. This does mean to restrict the possible values but stating that the values are instances of the declared class(es). Whatever resource (subject) we may annotate with the predicate rdfs:range is interpreted as instance of rdfs:property and the object of the triple is interpreted as instance of rdfs:class.

Similar to the range property, annotating a resource with rdfs:domain declares the subject as instance of rdf:Property and the triples object as instance of rdfs:Class. The statement then is, that any resource having that property (stated as subject) is an instance of the class given as object.

In a triple with the predicate rdfs:subClassOf, the subject and object are instances of rdfs:Class. All instances of the object are also instances of the subject class.

A triple in the form "subject rdfs:subPropertyOf object" declares both, subject and object as properties (instances of rdf:Property) and states that all resources related with the property given by the triple's object are also related with the property given by the subject.

With rdfs:label and rdfs:comment, resources can be annotated with human-readable names (label) and descriptions (comment).

Additional vocabulary

RDFS defines additional classes and properties related to RDF containers and so called utility properties:

rdfs:Container is the super-class of rdf:Seq, rdf:Bag and rdf:Alt.
rdfs:ContainerMembershipProperty is the class of the instances rdf:_1 … rdf:_n and itself a subclass of rdf:Property. Instances of rdfs:ContainerMembershipProperty are also sub-properties (rdfs:subPropertyOf) of rdfs:member, whereby rdfs:member is an instance of rdf:Property.

rdfs:seeAlso and rdfs:isDefinedBy are instances of rdf:Property and are used to refer to a resource that either provides additional information about a subject (seeAlso) or defines the resource (isDefinedBy).

Syntax used in this work

RDF(S) statements are represented in this work as simple triples of the form:

[prefix]subject [prefix]predicate [prefix]object

Prefixes ([prefix]) are either arbitrary, e.g. "ex:" for an example namespace, or "rdf:" and "rdfs:" according to the specification of the predefined vocabulary.

3.2.3 Web Ontology Language (OWL)

OWL as a recommendation of the World Wide Web Consortium (W3C)[13] was published in 2004 as a revision of DAML+OIL and specified the three increasingly-expressive sub languages OWL Lite, OWL DL and OWL full (McGuinness & van Harmelen 2004). DAML+OIL (Connolly et al. 2001) was a joint initiative that brought European (OIL) and American (DAML) language specification efforts together (Antoniou & van Harmelen 2003). OWL builds on RDF(S) e.g. by reusing certain constructs such as rdfs:domain or rdfs:subClassOf (McGuinness & van Harmelen 2004). The relation between RDF(S) and OWL will be discussed later in more detail. The latest recommendation as of October, 2009 is OWL 2 consists of 13 documents, whereby there are five rather technical core specification documents and four user guides (Anon 2009). Whilst the primary exchange syntax is RDF/XML, OWL 2 also specifies an alternative called Manchester Syntax (Anon 2009). The specification documents predominantly make use of a "Functional-Style Syntax" that is far shorter. An arbitrarily chosen example from (Bao et al. 2009) is the universal quantification which is declared as "ObjectAllValuesFrom(P C)" in the functional style and refers to three RDF triples "_:x rdf:type owl:Restriction", "_:x owl:onProperty P", and "_:x owl:allValuesFrom C" in the RDF Syntax. The well known dialects of OWL 1 (Lite, DL and Full) are not focused anymore in OWL 2. According to (Anon 2009), there are two ways to interpret OWL 2 ontologies: Direct Semantics and RDF-Based semantics whereas informally, "OWL 2 DL" refers to an ontology interpreted with the first (corresponding to SROIQ description logic) and "OWL 2 Full" to an ontology interpreted using the second semantics. However, OWL 2 instead defines three profiles (Motik, Grau, et al. 2009):

- OWL 2 EL - referring to the EL family of description logics that only uses existential quantification - for large ontologies.
- OWL 2 QL - referring to "Query Languages" as queries can be translated into relational query languages - for large instance databases focusing on query answering.
- OWL 2 RL - referring to "Rule Language" as reasoning can be implemented with a standard rule engine - for relatively high expressivity with even though scalable reasoning.

Because OWL 2 is backward compatible, any OWL Lite or OWL DL ontology is a valid OWL 2 ontology, OWL Lite and OWL DL can be seen as OWL 2 Profiles and, actually, any further Profiles can be defined (Hitzler et al. 2009; Motik, Grau, et al. 2009).

The following sections provide an overview of OWL 2 features based on (Hitzler et al. 2009; Bao et al. 2009; Motik, Patel-Schneider, et al. 2009). The description does not aspire completeness but should give an overview on the possibilities, especially in addition to RDF(S).

Class Expressions

OWL allows for expressing classes as intersection, union, complement of classes or enumeration of individuals. Intersection - ObjectIntersectionOf($C_1 \ldots C_n$) - means, that a class consists of the instances of the given classes ($C_1 \ldots C_n$). A union - ObjectUnionOf($C_1 \ldots C_n$) - is a class consisting of all individuals which are at least member of one given class ($C_1 \ldots C_n$). The complement of a class - ObjectComplementOf(C) - is a classes' negation that consequently contains all individuals which are not member of the given class (C). An enumeration defines a class through its members, the individuals $a_1 \ldots a_n$ in ObjectOneOf($a_1 \ldots a_n$).

Further on, we may define a class quite precisely by restricting their properties. Whilst RDF(S) - with this respect - simply allows us to define that a class has a certain property with a certain range of values, OWL has specific quantification possibilities. First of all, a property may be universal or existential: An existential quantification - ObjectSomeValuesFrom(P C) - means, that

[13] http://www.w3.org/

an instance must be connected to at least an individual of class "C" via property "P" to be considered as instance of a specific class. The universal quantification is indicated with ObjectAllValuesFrom(P C) and requires that, if an individual is connected to a class via "P", that individual must be member of "C".

With ObjectHasValue(P a) we may define a class by its relation (P) to a specific individual (a). The expression ObjectHasSelf(P) indicates local reflexivity: a member has itself as value of property "P".

Further on property restrictions allows us to restrict the number of relations. With the qualified expressions ObjectMinCardinality(n P C), ObjectExactCardinality(n P C), ObjectMaxCardinality(n P C) we may specify the number (n) of individuals of class (C) related with property (P) by a minimum, maximum or exact number. The same restrictions are available without specifying the class "C" - thus, restricting the number of individuals connected by "P" independent of their (sub-)class membership. Similar restrictions are possible with respect to datatype properties. Reasonably, local reflexivity is not available. Instead of an object property (P), a datatype property (R) is indicated, instead of a class a datatype and instead of an individual a literal value is given - for example, ObjectHasValue(P a) versus DataHasValue(R v).

Also equivalence (classes containing the same set of individuals) and disjointness (an individual cannot be a member of disjoint classes) can be expressed, even a disjoint union (where all individuals of several classes belong to only one of those but are members of another class).

Property Expressions

There are several expressions to characterise Properties: Inverse properties have another direction with respect to another property, for example if father (x) hasChild child (y), also y hasFather x is true. Symetric properties (such as 'spouse') are similar in both directions. In contrast, an asymmetric property excludes the inversion (e.g. if x hasChild y, y hasChild x is never true). Disjointness with respect to properties means, that an individual can only be connected by one of a set of disjoint. Also equivalence of properties can be stated. Reflexive and irreflexive properties relate to themselves or they may not do so respectively. With a functional property one can indicate, that at most one other individual can be connected with it. Functionality can also be defined with respect to the inverse of a property. Finally, it is possible to define a transitive property - meaning that, if A and B as well as B and C are connected with that property, also A and C are connected with it (for example hasAncestor). With property chains we can even specify, that for example "A hasGrandparent C", if the statements "A hasParent B" and "B hasParent C" are given.

There are several further constructs, for example to specify data ranges, make individual assertions and of course also to define class and property hierarchies like it is possible in RDF(S).

Syntax used in this work

OWL expressions in this work are represented based on the Manchester Syntax as described in (Horridge & Patel-Schneider 2009). To understand the examples contained in this document, the following information should be sufficient:

Expressions are defined as frame, such as classFrame, individualFrame or objectPropertyFrame starting with the corresponding keyword "Class:", "Individual:" or "ObjectProperty:", followed by an IRI and a set of axioms. These axioms are again indicated by rather self explaining expressions such as e.g. "SubClassOf:", "EquivalentTo:" or "SubPropertyOf:". Enumerations are enclosed in "{" and "}" and properties of property chains are separated using "o". The following table lists some operators and keywords and relates them to the functional style syntax expression.

Manchester	Functional-Style Syntax
some	ObjectSomeValuesFrom / DataSomeValuesFrom
only	ObjectAllValuesFrom / DataAllValuesFrom
value	ObjectHasValue / DataHasValue
min	ObjectMinCardinality / DataMinCardinality
exactly	ObjectExactCardinality / DataExactCardinality
max	ObjectMaxCardinality / DataMaxCardinality
<=	minInclusive
>=	maxInclusive
<	minExclusive
>	maxExclusive

Table 1: Manchester Syntax keywords and operators

3.2.4 Semantic Web Rule Language (SWRL)

SWRL (Horrocks et al. 2004) is specified in a member submission to the W3C and enhances OWL by unary and binary rule statements. Basically, a rule consists of an antecedent and a consequent, both of which consist of a set of atoms. If the antecedent is true (i.e. all atoms hold), then also the consequent is true. If the antecedent is empty, the consequent must be satisfied by any interpretation. If the consequent is empty, the antecedent must not be satisfied by any interpretation. The following forms of atoms are possible:

Atom	Description
C(x)	C is a class expression (named class or OWL expression) and x is a named individual or a variable.
D(x)	D is a data range expression and x either a variable or a data value.
P(x, y)	P is a data- or object-property, x is a variable or OWL individual and y is a variable, OWL individual (if P is an object-property) or data value (if P is a data-property).
sameAs(x, y)	x and y are variables or individuals and the atom holds, if both are interpreted as being the same individuals.
differentFrom(x, y)	x and y are variables or individuals and the atom holds, if they are interpreted as being not the same individuals.
builtIn(r, x_1,…x_n)	R is a built-in relation and x_1,…x_n are data values or variables. The SWRL submission defines numerous built-ins such as different kind of value comparisons; therefore they will not be listed or described here. However, some of them will be referred to later in this work.

Table 2: Overview on SWRL language constructs

The abstract syntax and the XML based concrete syntax defined in the SWRL specification are rather difficult to read and lead to long statements. Therefore, the rules presented in this work will be described using a rather informal style (as also exemplified in the specification) that corresponds to the form of atoms described above and can also be used to define rules in Protégé (with slight differences, also depending on the version):

The sign "→" is used between the antecedent and the consequent atoms. "," is used for the conjunction of atoms. Variables are indicated by "?" and built-ins are indicated by the prefix "swrlb:". An example rule taken from (Horrocks et al. 2004) infers that x3 is the uncle of x1, if x2

is a parent of x1 and x3 the brother of x2. In the work at hand, the rule would be represented in the following form:

hasParent(?x1,?x2), hasBrother(?x2,?x3) → hasUncle(?x1,?x3)

3.2.5 Reasoning

By specifying formal semantics for a language it is ensured, that the meaning of knowledge is not open to different interpretations, e.g. by different persons or machines and is thus a prerequisite for reasoning support (Antoniou & van Harmelen 2003). (Antoniou et al. 2005) refer to reasoning as "[…] process of deriving valid deductions from an ontology […]". Web ontology languages in particular allow for deducing (Antoniou et al. 2005):

- Class membership of individuals,
- Classification (subclass relationships),
- Class equivalence (i.e. classes have the same extension),
- Class consistence (i.e. classes do not have an empty extension),
- Ontology consistency - i.e. that there is at least one model that may instantiate an ontology.

The extension of a class is the set of individuals associated with it (Bechhofer et al. 2004). Regarding classification, (Antoniou et al. 2005) refers to subclass membership whilst (Antoniou & van Harmelen 2003) refer to the class membership of individuals based on sufficient conditions. In this work, the term "classification" will refer to both, subclass relationships and individual membership to classes.

There exist numerous description logic reasoners. An overview is provided on the homepage of Uli Sattler from the University of Manchester[14] and on the OWL implementation site of W3C[15]. The work at hand will mainly use Pellet[16], because it is freely available as Java API, is integrated in the latest version of Protégé and supports OWL2 as well as SWRL.

3.2.6 Relations between the Languages

(Antoniou & van Harmelen 2003) explain, that simply extending RDFS with the language primitives available in OWL would lead to uncontrollable computational properties. The trade-off between expressive power and efficient reasoning led to three sublanguages in OWL 1 as mentioned above:

- OWL Full (the entire language, thus not really a sublanguage) allows the combination with any RDF(S) construct, thus even changing the pre-defined primitives (e.g. property restrictions on rdfs:Class) but actually, OWL Full is undecidable.
- OWL DL in particular disallows to apply pre-defined constructs to each other. Whilst efficient reasoning becomes possible, the compatibility with RDF is impaired.
- The third sublanguage, OWL Lite, was just a further restriction on OWL DL with respect to the language constructs.

Among themselves, the three sublanguages are upward compatible (from Lite to Full) as shown in Table 3 in addition to the relations between RDF(S) and OWL.

Compatibility between RDF- and OWL- Documents and conclusions		
A egal RDF Document	is a	Legal OWL Full document
A legal RDF(S) conclusion	is a	legal OWL Full conclusion

[14] List of Description Logic Reasoners: http://www.cs.man.ac.uk/~sattler/reasoners.html

[15] W3C Reasoner Implementation List: http://www.w3.org/2007/OWL/wiki/Implementations

[16] Pellet Reasoner: http://clarkparsia.com/pellet/

A legal RDF Document	is Not necessarily a	Legal OWL DL document
A legal OWL DL document	is a	Legal RDF Document
Upward compatibility between OWL 1 Sublanguages		
A legal OWL Lite Document	is a	Legal OWL DL Document
A legal OWL DL Document	is a	Legal OWL Full Document
A legal OWL Lite Conclusion	is a	Legal OWL DL Conclusion
A legal OWL DL Conclusion	is a	Legal OWL Full Conclusion

Table 3 : Compatibility between RDF(S) and OWL Sublanguages based on Antoniou & van Harmelen (2003)

As already mentioned, any OWL 1 document is a valid OWL 2 document, thus, taken the three dialects as OWL 2 profiles, the relations are still valid. According to (Hitzler et al. 2009) none of the OWL 2 Profiles (EL, QL, RL) is a subset of another. Regarding the syntactical conformance (Smith et al. 2009) make the following statements:

- "An OWL 2 Full ontology document is any RDF/XML document […]".
- "An OWL 2 DL ontology document is an OWL 2 Full ontology document […]" given some conditions being fulfilled according the specification regarding parsing, the RDF mapping and further syntactical restrictions.
- "An OWL 2 EL ontology document is an OWL 2 DL ontology document […]" fulfilling the corresponding profile specification.
- "An OWL 2 QL ontology document is an OWL 2 DL ontology document […]"fulfilling the corresponding profile specification.
- "An OWL 2 RL ontology document is an OWL 2 DL ontology document" fulfilling the corresponding profile specification.

However, the following relations to RDF(S) apply to all sublanguages (Antoniou & van Harmelen 2003):

- RDF Syntax is available
- As it became visible in the language description, the instance declaration bases on the RDF constructs (rdf:Description, rdf:type)
- owl:Class is a specialisation of rdfs:Class and owl:ObjectProperty and owl:DatatypeProperty are specialisations of rdf:Property.

These relations are basically still valid: Given the RDF/XML Syntax which is according to (Hitzler et al. 2009) still mandatory with respect to language conformance (although there is also an optional syntax specification) and according to the specification of the RDF based semantics (J. Carroll et al. 2009) also the property and class relations are given.

Rules in particular add the possibility of using variables (Motik et al. 2004) to OWL. Although SWRL may become undecidable under some circumstances (Horrocks et al. 2004), there exist reasoning support, for example by Pellet with the restriction that rules apply only to named individuals in the ontology[17].

[17] SWRL support of Pellet: http://clarkparsia.com/pellet/faq/rules/

3.2.7 Comparison with Features of SBVR

SBVR (Semantics of Business Vocabulary and Business Rules) "[…] defines the vocabulary and rules for documenting the semantics of business vocabularies, business facts, and business rules […]" (OMG 2008).

By defining the vocabulary, rules and "structured English", SBVR points out several characteristics with respect to language expressions which seem suitable to form a basis for comparing languages with respect to their features and capabilities. SBVR bases on first-order logic with extensions, whilst OWL is a subset of first-order logic. Expectably, SBVR offers more than OWL could represent. The SBVR specification (OMG 2008) provides a table that maps the concepts of SBVR to ISO Common Logic and also to OWL. However, the mapping seems incomplete and (OMG 2008) states that the "[…] cells that are empty will be specified in a future revision […]". Further on, there are several entries which refer to the meaning of vocabulary terms, such as "rule", "integer" or "operand". Therefore, the table is not adequate for the purpose in this work of having a schema to compare or select required language features. For this reasons, the following table does not base directly on the mapping given by SBVR but contains several features and properties occurring in the specification. Further on, the features are compared to RDF(S), SWRL and also to the latest OWL specification.

Category	**Keyword / Label**	**RDF(S)**	**OWL Feature / Functional Syntax**	**SWRL**
World of discourse	Open-world assumption	Open-world assumption	Open-world assumption	Open-world assumption
	Closed-world assumption (SBVR: local / per concept in a domain)	-	-	-
Quantification				
	each (universal quantification)	Universal (rdfs:domain & rdfs:range)	Universal / ObjectAllValuesFrom (P C)	Variables are treated as universal quantified.
	some (existential quantification)	No quantification	Existential / ObjectSomeValuesFrom(P C)	OWL expressions can be used.
	at least one (existential quantification)	No quantification	Existential / ObjectSomeValuesFrom(P C)	-
	at least n (at-least-n quantification)	No quantification	Minimum (qualified) cardinality / ObjectMinCardinality(n P) or ObjectMinCardinality(n P C)	-

	at most one (at-most-one quantification)	No quantification	Maximum (qualified) cardinality / ObjectMaxCardinality(n P) or ObjectMaxCardinality(n P C) with n=1	-
	at most n (at-most-n quantification)	No quantification	Maximum (qualified) cardinality / ObjectMaxCardinality(n P) or ObjectMaxCardinality(n P C)	-
	exactly one (exactly-one quantification)	No quantification	Exact cardinality / ObjectExactCardinality(n P) with n=1	-
	exactly n (exactly-n quantification)	No quantification	(Qualified) exact cardinality / ObjectExactCardinality(n P) or ObjectExactCardinality(n P C)	-
	at least n and at most m (numeric range quantification)	No quantification	Minimum (qualified) cardinality and Maximum (qualified) cardinality / ObjectMaxCardinality(n P) or ObjectMaxCardinality(n P C) and ObjectMinCardinality(n P) or ObjectMinCardinality(n P C)	-
	more than one (at-least-n quantification with n = 2)	No quantification	Minimum (qualified) cardinality / ObjectMinCardinality(n P) or ObjectMinCardinality(n P C) With n=2	-
Logical Operations		-		

	it is not the case that p (logical negation)	-	Complement / ObjectComplementOf(C)	-
	p and q (conjunction)	-	Intersection / ObjectIntersectionOf(C1…Cn)	Conjunction (",")
	p or q (disjunction)	-	Union / ObjectUnionOf(C1…Cn)	-
	p or q but not both (exclusive disjunction)	-	Disjoint union / DisjointUnionOf(CN C1…Cn)	-
	if p then q (implication)	-	-	Antecedent implies consequent.
	q if p (implication)	-	-	(reverse formulation)
	p if and only if q (equivalence)	-	EquivalentObjectProperties(P1 … Pn)	-
	not both p and q (nand formulation)	-	-	-
	neither p nor q (nor formulation)	-	-	-
	p whether or not q (whether-or-not formulation)	-	-	- (p without condition on q)
Modal Operations				
Operative	it is obligatory that p / … must … (obligation formulation)	-	(*) Existential quantification / cardinalities (necessary conditions)	- (*)
Operative	it is prohibited that p / … must not … (obligation formulation embedding a logical negation)	-	(*) Existential quantification with complement	- (*)
Structural	it is necessary that p / … always … (necessity formulation)	-	(*) Existential quantification / cardinalities (necessary	- (*)

			conditions)	
Structural	it is impossible that p / … never… (necessity formulation embedding a logical negation)	-	(*) Existential quantification with complement; Disjoint class definition.	
Structural	it is possible that p (possibility formulation)		(*) Class without restriction on property.	-
Operative	it is permitted that p / … may … (permissibility formulation)		(*) Class without restriction on property.	
Semantic Formulations				
	aggregation formulation	-		-
	antecedent			
	auxiliary variable			
	bag projection			
Time				
	Date1 before Date2 Date1 after Date2 Date1 in the future Date1 in the past Period1 has duration Period1 overlaps period1		- Some aspects may be covered with DatatypeRestrictions (on Time instants or Numbers) with the facets: xsd:minInclusive xsd:maxInclusive xsd:minExclusive xsd:maxExclusive	SWRL built-ins for time and duration and comparison (lessThan, LessThanOrEqual, greatherThan, greaterThanOrEqual). See also (O'Connor & Das n.d.).

Table 4: Comparison of language features based on SBVR

Structural and Operative Rules and Modalities (*)

Entries denoted with (*) in table [table ##] are difficult to relate with the languages. For sure, neither OWL nor SWRL (or RDFS) provides a mean to explicitly declare a modality. In the current OWL specification no statements about modalities were found and in the former specification (Patel-Schneider & Horrocks 2004), the term is only used regarding class equivalence (modality "complete") and subclass axioms (modality "partial"). This brings the denotations "necessary" versus "necessary and sufficient" to mind which was also used in older versions of the Protégé tools. The term "necessary" to close again, is used only in structural rules in SBVR. "Sufficient" on the other hand means that fulfilling a certain condition is sufficient to reason, that something is of a certain type.

3.3 Ontologies in Enterprise Modelling

The idea of representing enterprises by means of (formal) ontologies is not a new one. However, the ideas, approaches and applications are very different. The following sections present a rough overview on the directions and approaches related to enterprise modelling.

3.3.1 Ontologies for Modelling and Metamodelling

Several sources emphasise the importance of meta-modelling capabilities. (Hinkelmann, Nikles, Thönssen, et al. 2007) presented an ontology based modelling tool (ATHENE) that integrates meta-modelling. Their proposed meta-modelling approach consists of three model levels: the meta-meta model which essentially defines an objecttype-, a relation- and an attribute-class. The meta-models are thought of as a set of subclasses of those essential classes. The model level is conceptualised as instances of the meta-model classes. This corresponds well with the meta-modelling hierarchy as depicted in Figure 14

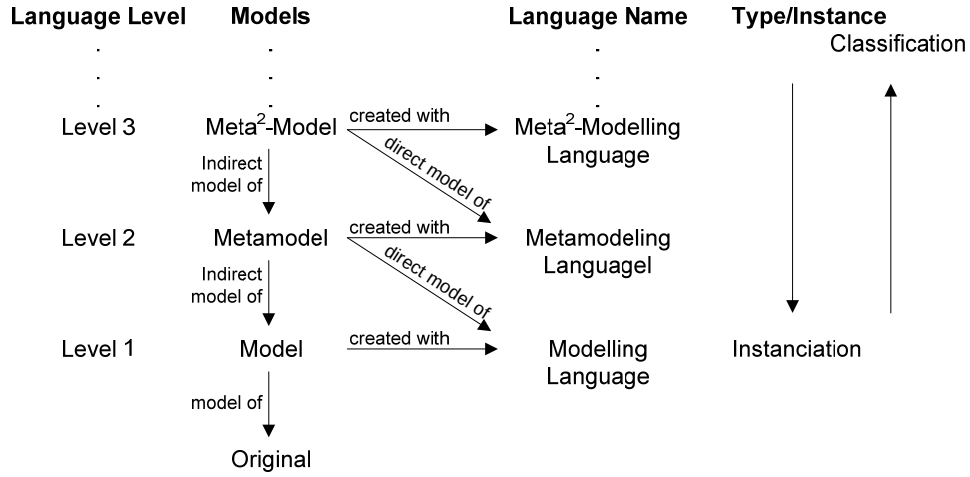

Figure 14: Metamodelling hierarchy according to Karagiannis & Kühn (2002)

Whilst Karagiannis & Kühn (2002) address the importance of finding an adequate level of abstraction to define the upper level (i.e. meta-meta model or above), they also depict the levels as shown in Figure 14 which contains an allusion to be questioned regarding the lower levels: Possibly, not every model represents an instantiation. This topic will be further discussed in the context of required language features in section 7.1.4.

Further approaches to formalise enterprise models often concentrate on specific languages, in particular business processes, thus do not necessarily integrate a meta-meta-model. (Di Francescomarino et al. 2009) for example created a BPMN ontology in OWL to represent business process models whereas models can be annotated with concepts from a domain ontology in order to enable the merging of the process and domain ontologies. (Karastoyanova et al. 2008) also use a BPMN ontology defined in WSML and annotations to relate the process models to organisation and domain ontologies. Such annotations can be used to query a process repository as well as later during the exection. An editor to annotate existing business process models, to create new ones and to transform them into sBPEL is available as extension of the WSMO-Studio[18] and shortly described in (Dimitrov et al. 2007). These approaches will be further discussed in the subsequent sections focusing on different applications.

The ATHENE approach does not attempt to annotate things but rather allows for representing the same concepts with different visualisation in different models and to relate different elements from different models (Nikles & Brander 2009). Thus ATHENE envisions the possibility to

[18] Semantic Web Service and Semantic Business Process modelling environment: http://www.wsmostudio.org/

also graphically model the domain knowledge and to directly (re-)use these concepts in other models (e.g. a business process) rather than merging different ontologies.

3.3.2 Semantic Business Process Management

Hepp et al. (2005) proposed the term *Semantic Business Process Management* (SBPM) as a combination of semantic web technologies, semantic web services and business process management (BPM) in order to support the agile execution and analysis of business processes. Several projects dealt with topics in this area ranging from design to post execution analysis, for instance the European founded Projects SUPER[19] or, in the government sector, FIT[20]. In this thesis, business processes are considered as a very important aspect of enterprise architecture because processes represent the way an enterprise creates value, they implement the strategy, are a linking point to ensure policy and regulatory compliance and their execution found the basis for performance measuring. This section comments on different aspects and outcomes of semantic business process management.

Semantic Web Services

A lot of effort has been put in the direction of semantic web services; therefore the following sections provide an overview of the prominent exponents.

OWL-S (as successor of DAML-S) is a W3C member submission for an ontology of services (D. Martin et al. 2004). OWL-S aims at supporting atomic but especially also composite services. By describing the services, their inputs and outputs as well as prerequisites and consequences in a machine-interpretable form, the automated recovery, invocation and composition should be supported.

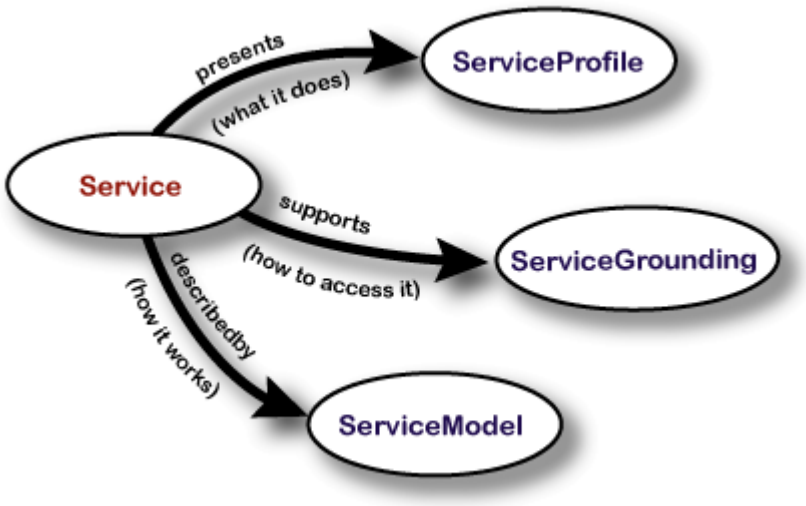

Figure 15: Top Level of OWL-S (D. Martin et al. 2004)

Figure 15 visualises the top level of the ontology: Service profiles can be used to describe the needed (for the requester) and the offered service (by a provider). This consists in particular of information about the organisation (e.g. contacts and responsibilities), functionality (e.g. input and output) and characteristics (e.g. category and quality). The ServiceModel particularly describes the interaction with the service user including a process description with control constructs, pre- and post-conditions and activities to be executed (requiring message exchange with the service consumer). The ServiceGrounding specifies the grounding, i.e. by using WSDL and SOAP.

[19] SUPER (Semantics Utilized for Process management within and between EnteRprises) project: http://www.ip-super.org/

[20] FIT (Fostering self-adaptive e-government service improvement using semantic Technologies): http://www.fit-project.org

Arguing that OWL does not have the expressive power to unambiguously represent the intended meaning of OWL-S, Grüninger et al. (2005) presented FLOWS, a first order logic ontology for webservices. Against the criticism regarding decidability, they argue that there are often simple solutions for practical problems and that intractable problems will have to be solved instead of make the language less expressive. FLOWS is part of the Semantic Web Services Ontology (SWSO) member submission to the W3C (Battle et al. 2005). Some goals of SWSO are to support fluents (terms and predicates changing over time). Although the authors claim that OWL is not capable of representing OWL-S, they not clearly state what the limitations are. However, the presented constructs, such as fluids and rules with n-ary relations give evidence. With this respect, the OWL-S submission at W3C (D. Martin et al. 2004) only refers to the RDF:List vocabulary used in process descriptions.

WSMO, the Web Service Modelling Ontology (Roman et al., 2005) is comparable to OWL-S. According to (Martin et al., 2004) the efforts are rather complementary as the WSMO working group had a strong focus on the language development (i.e. WSML, the Web Service Modelling Language) and the correspondence to OWL. The top level elements of WSMO as shown in Figure 9 are: ontologies that provide a terminology to be used by other WSMO elements. Web service descriptions include functional and non-functional properties as well as interface information. Goals describe what kind of service a service consumer actually is looking for, thus partially the same properties as for the service description can be provided. Mediators should support interoperability purposes, for example resolving mismatches between ontologies, connecting goals or linking goals to services. (Roman et al., 2005) criticizes OWL-S as compared to WSMO in that the OWL-S language specification would not clearly found on MOF-style metamodel layers, had no distinct language basis (e.g. OWL, SWRL and different notations) and open decidability problems. WSMO is supported by the WSML language family that provides different levels of expressiveness and computational properties to support the reasoning tasks for different applications (Roman et al., 2005).

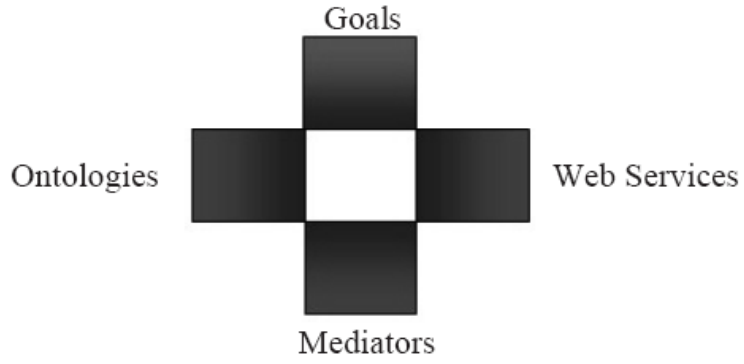

Figure 16: Core Elements of WSMO (Roman et al. 2005)

The WSML language family comprises the following variants (De Bruijn et al. 2005):

The lowest level of expressiveness is provided by WSML-Core, based on the intersection of Description Logics and Horn Logic. WSML-DL extends WSML-Core to SHIQ(D), thus has a lower expressive power than OWL-DL. WSML-Flight also extends WSML-Core with, e.g. non-monotonic negation and meta-modelling (by not distinguishing instances and classes). WSML-Rule extends WSML-Flight for logic programming and WSML-Full combining WSML-DL and WSML-Rule.

Semantic Business Process Execution and Analysis

Karastoyanova et al. (2008) presented a framework for semantic process management including modelling, execution and analysis of business processes. The activities represented in the business process model ontology (BPMO) can specify goals using WSMO. The transformation into an executable format (sBPEL) includes the search for (or the composition of) semantic web services matching these goals, thus, the availability of appropriate services can be checked. For the actual process execution an extended serialisation form of BPEL called BPEL4SWS is used

to support goal-based invocation of services with a corresponding process engine. During the execution, all logged events are stored with their context information as instances of an event ontology. Based on this event ontology, active process monitoring (i.e. showing the execution status) as well as analysis of historic execution data (i.e. process mining) to, for example, discover room for improvement or deviations from the process design are enabled. More details about sBPEL are described in (Nitzsche et al. 2007), what will not be deepened here, though one specific point of interest for this work is mentioned there: the missing ability (in WSML) to express ordering information for activities also used for else-if constructs. Besides sBPMN also an sEPC ontology, formalising Event based Process Chains (EPC), evenly linked to a common process ontology used to transform the models into an executable form (sBPEL) (Pedrinaci & Domingue 2007) exists. These publications relate to the SUPER project which defined a whole stack of ontologies for semantic business process management as depicted in Figure 17.

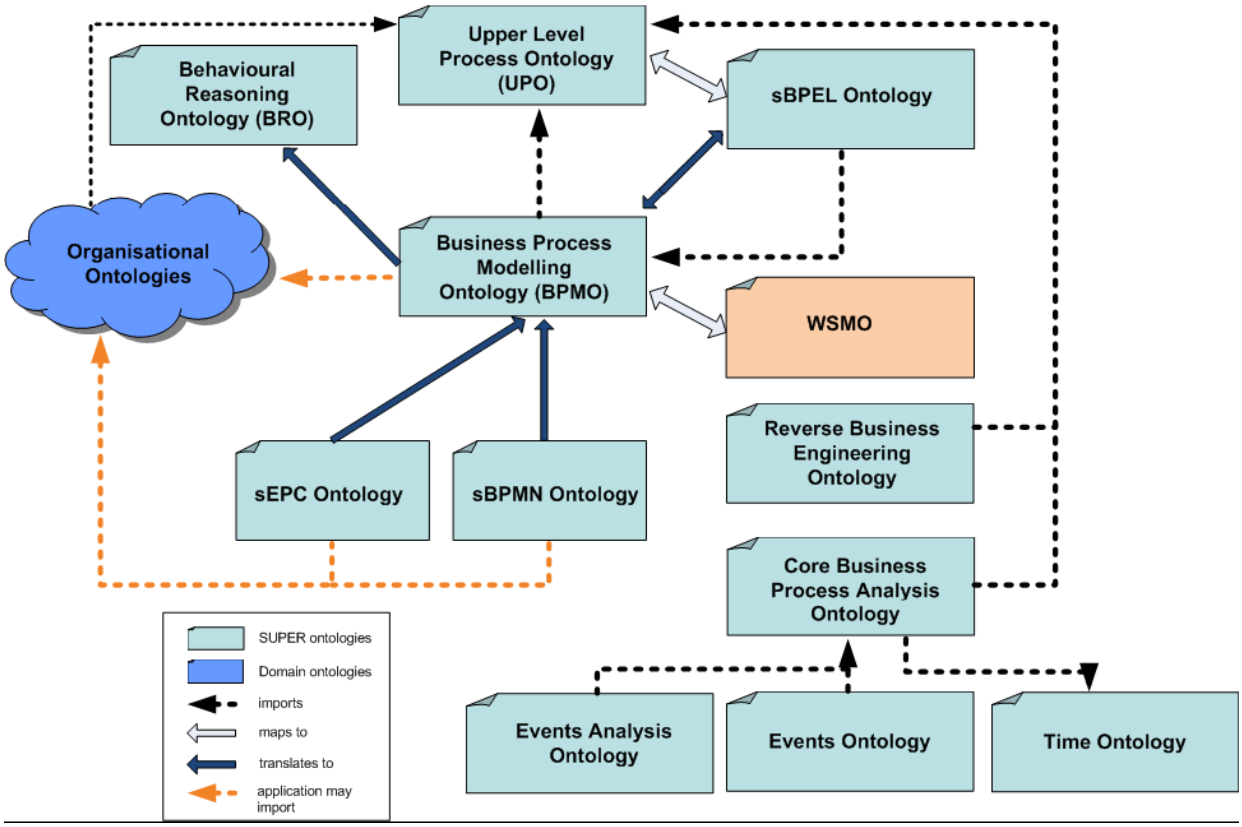

Figure 17: SUPER Process Ontology Stack (Janusch 2009)

Feldkamp et al. (2007) presented an approach called KISS (Knowledge Intensive Service Support) that combines fixed and variable activities which represent knowledge intensive tasks that will be determined at run-time. In contrast to a goal-based approach that tries to find services which fulfil certain goals, this approach determines whether and which activities have to be executed (independent from actual service implementations). The KISS approach distinguishes rules that are concerned about (a) resource allocation, (b) constraints checking, (c) branching and (d) activity and sub-process detection and selection. The approach is divided into three phases whereas the first phase deals with semi formal representations (e.g. process models and business rules); in the second phase the semi-formal representations are transformed into formal ones (e.g. OWL-S and SWRL) and finally, in the third phase, the formal representations are transformed into executable languages (i.e. BPEL). In (Feldkamp et al. 2007) there is rather little information given about technical details of the implementation but two points become apparent: first, the transition from the first to the second phase is described as done manually, thus, the processes and rules described by business experts are reformulated and thus at risk of being wrongly implemented. Second, in the third phase the transformation of OWL-S into

BPEL is mentioned but no information about possible linking of execution information to the formal models, e.g. for analysis.

(Witschel et al. 2010) points out that the KISS approach does neither deal with execution variances nor with the knowledge gained through task handling. (Witschel et al. 2010) describe how an agile execution can be combined with so called task patterns which develop over time through the behaviour of the users (e.g. by copying resources or solution objects from patterns to the task underway) and can be shared with other users and analysed regarding several aspects such as often added resources. However, the approach still relies on a transformation of ontological representations to standard executable formats such as BPEL or XPDL and does not describe, how the execution information is related to the semantic models.

Figure 18 depicts the performance management lifecycle described in (Wetzstein et al. 2008). Considering the goals of enterprise architecture such as design, optimisation, planning and controlling, a consistent view seems to be of importance: the modelling starts on basis of strategies and goals, defines KPI, uses the models for execution (transformation into executable models) and evaluates the execution results in relation to the defined models and measures.

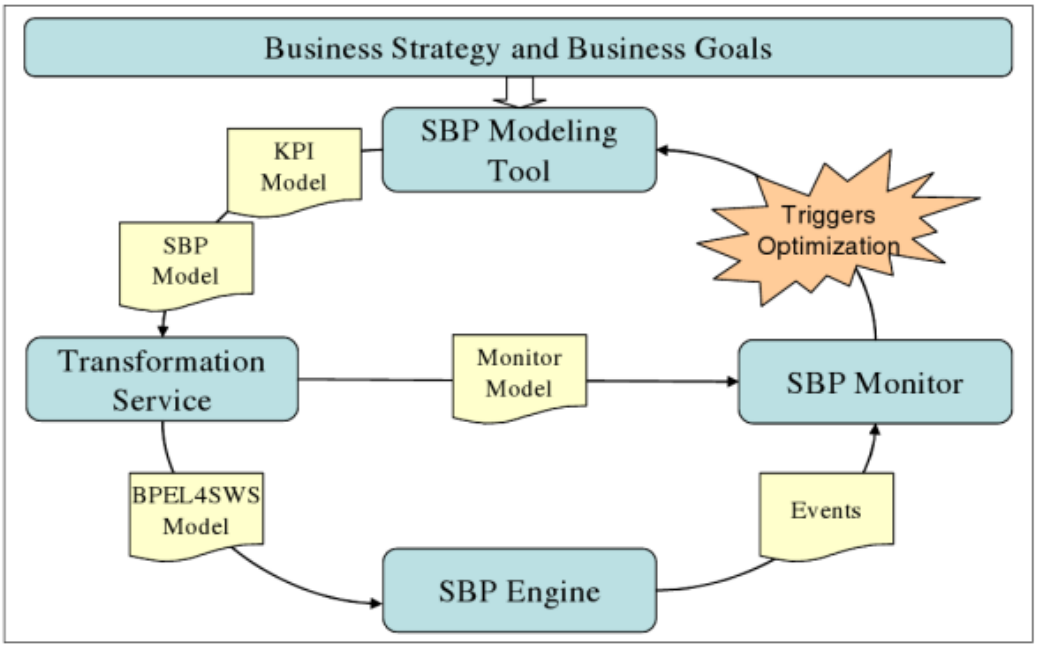

Figure 18: Performance Management Lifecycle (Wetzstein et al. 2008)

3.3.3 Semantic Compliance Management

Kim et al. (2007) and Kim & Fox (2002) described an approach for automated ISO 9000 compliance evaluation using the TOVE ontology. Their models were based on First Order Logic expressions and the compliance evaluation was done using Prolog. They regard compliance as a goal which is achieved, if a corresponding number of constraints are satisfied. The constraints may be for example, that all processes of a certain kind (e.g. inspection and test) must have certain outputs (e.g. some quality evidence and some quality record). The examples given in (Kim & Fox 2002) also show, that tracing models may be required (e.g. ensuring, that the origin of a resource is traceable through a process).

Di Francescomarino et al. (2008) and Di Francescomarino et al. (2009) present an approach to check several kinds of restrictions related to a business process such as general guidelines or security and privacy issues - exemplified in the context of a web shop process. General guidelines may e.g. restrict the number of outgoing sequences from a gateway to keep models simple. Regarding privacy, a constraint may ensure that the organisation policy has to be read before private data is provided. Or with respect to security that an authentication sub-process has to be contained in the model. The approach uses a domain ontology, a BPMN ontology and

instances generated from a concrete process model. Additionally, merging axioms specify, with which domain ontology concepts the corresponding modelling elements may be annotated (by defining subclass relations). Constraints can be formulated based on the BPMN concepts or in combination domain concepts. The approach targets at the reasoning of inconsistencies or especially unsatisfied concepts given by the union of the domain concepts, process ontology, the merging axioms and constraints. Even though results of an experimental test is presented the constraining requirements seem rather simple as compared to some regulatory issues and there are only few hints regarding language limits and, although cardinality restrictions were used, no statement was made about the properties of open world reasoning with e.g. minimum cardinalities.

Ceravolo et al. (2008) described an approach to enforce policies based on OWL (DL). Processes are modelled by instantiating concepts of the BMO Ontology[21] and the process execution is represented on basis of a log ontology, that for example contains transitions of the type "log:ProcessTransition" which are related to the process model through further properties such as "log:workflowEntity" that may point to a BMO relation (e.g. sequence flow) of type "PrivateSplitToTaskRelation". Further, the execution (log) data is related to a domain ontology that contains customer information, for example. Policies are then formulated by annotating relations with ASK queries in SPARQL[22], such as the following example to check, whether a free upgrade is granted (Ceravolo et al. 2008):

> "ASK { ?x rdf:type log:ProcessActor
>
> ?x rdf:type dom:Customer
>
> ?x dom:eligibleFor dom:FreeUpgrade }"

However, in the presented form, the query would actually check whether there exists any "x" being customer, process actor and eligible for an upgrade. Further, the approach exemplifies only a gateway condition based on a few business rules (related to the EU-Rent example of SBVR) and does not fully describe the enforcement (or checking). By using SPARQL queries, the conditions are not embedded in the logical model but rather they have to be interpreted by a program. Some presented queries involve arithmetic aggregations but the actual value (i.e. percentage) is not calculated by the presented queries. The authors regard the actual enforcement as future work.

Kähmer & Gilliot (2009) as well as (Raub & Steinwandt 2006) point out the importance of modalities such as permissions and prohibitions but also highlight the fact, that compliance requirements may have different sources and consequently their concurrence has to be considered (e.g. contradictions and composition of rules). (Kähmer et al. 2008; Kähmer & Gilliot 2009) developed an approach called ExPDT (Extended privacy definition tool) with a corresponding policy language based on OWL-DL. The language defines the elements and their properties used to formulate policies. The language defines a ruling for each rule which combines modalities, obligations and sanctions, for example, a ruling could state, that an action is allowed but a notification is necessary or that something is prohibited and sanctioned with a fine. The work places a special emphasis on the prioritisation and comparison of rules and present corresponding algorithms; also a default ruling is comprised in the ExPDT language for cases, where no rule matches certain conditions. The presented monitoring tool uses OWL queries to find rules matching related elements such as user, action and data and evaluates the corresponding ruling (of the rule with the highest priority). Relating system events to conditions (e.g. that a user consents by clicking on a certain button) is done manually. Actually, no concrete facts are provided on how to relate processes or tool data to the policy ontology. The authors argue that obligations were not enforceable in general, because they describe actions which have to be done in future. The authors even though give examples of obligations, such as the notification of a

[21] BMO (Business Management Ontology): http://www.bpiresearch.com/Resources/RE_OSSOnt/re_ossont.htm

[22] SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/#ask

customer within a given time period whose fulfilment can be evaluated by checking the condition after the corresponding period. Further, Krähmer & Gilliot generalise, that obligations which can be formulated as preconditions are enforceable. However, enforcement of obligations is mentioned as part of their future research.

4 Junisphere Interview

The main purpose of the interview was to gain insight into real-world enterprise modelling and profiting from long experience. The following sections describe the interview situation as well as the resulting findings.

4.1 Company

"Junisphere is a leading software company for end-to-end business service management (BSM) and beyond"[23]. Their products, the ITSIR™ methodology and the eRanger™ software, help organisations to overcome the difficulties of business and IT alignment (Junisphere Systems AG 2010). As business-IT alignment is one of the major objectives of EA and Junisphere has practical experience in applying a methodology to model the business services and their dependencies as well as a tool to monitor and control these services, the company was considered as valuable counterpart for an interview.

4.2 Interview settings

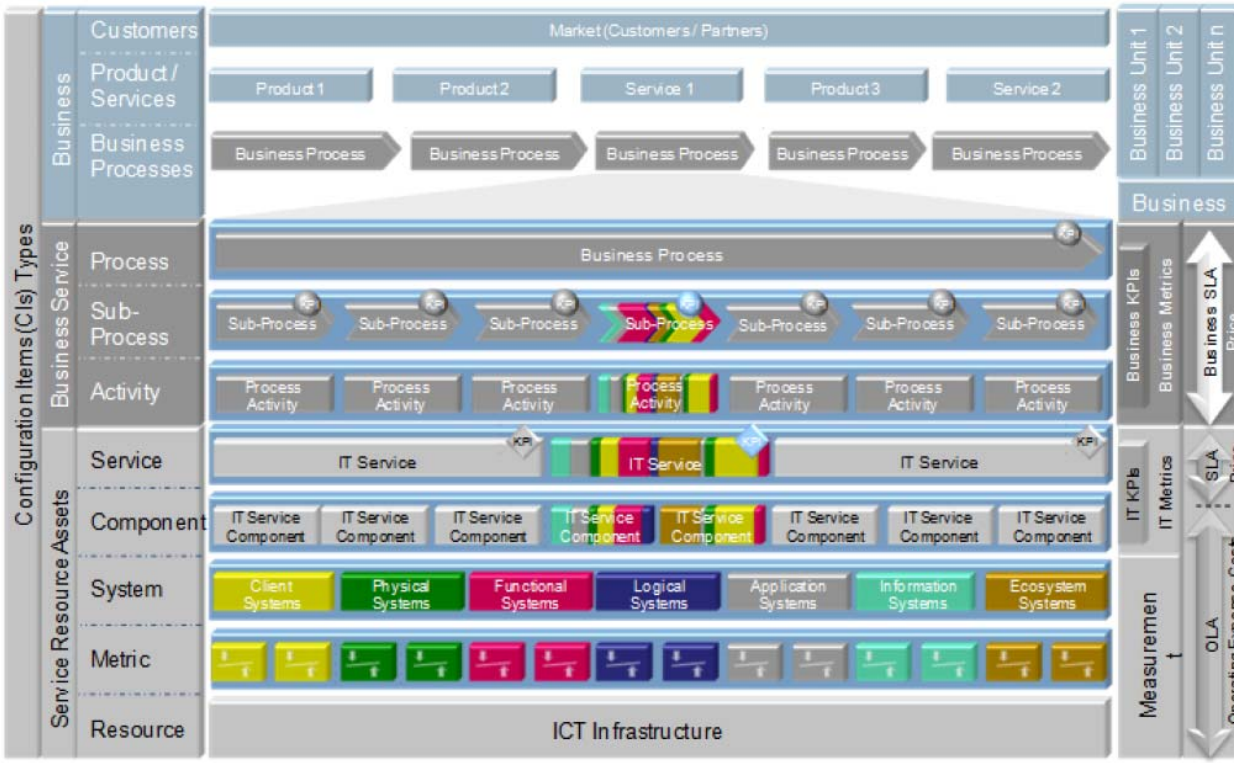

Figure 19: Generic Model of the ITSIR Method (Junisphere Systems AG 2010)

The actual meeting took place with Guido Salvi, the principal consultant of Junisphere on Wednesday, 2010-12-15 for about 2 ½ hours. The interview was not planned or performed in a classical "question-answer style" but rather by a consequent request for explanation. Following the generic model of the ITSIR method (see Figure 19) top-down, it was tried to work out, what is modelled on each layer (e.g. process), what information is captured (e.g.

[23] Junisphere Systems AG – Fact Sheet: http://www.junisphere.net/fact.htm, (Accessed on 2010-11-21)

process attributes) and why the information is relevant in terms of functionality or how it is used respectively.

4.3 Findings

The following paragraphs summarise the findings of the interview session. The top level of the generic model (Figure 19) represents the business with the market and the products offered. This part is not modelled but leads to the processes which reoccur on the next level (Business Service).

Process Level

The processes are represented on three levels in the generic model: Process, Sub-Process and Process Activity. These can be seen as process objects having basically the same attributes: predecessor, successor, child and status. The modelled activities do not represent a workflow in the classical sense and the succession attributes solely represent the order to graphically display the items in the eRanger.

An example was the process "order process" for which there may be the sub-processes "receive order", "prepare goods", "deliver goods" and "invoicing". By further investigating these sub-processes (i.e. asking how a sub-process is actually performed), it may be discovered that there are three entry points for "receive order", e.g. by a web-interface, by e-mail or by telephone considered as three activities. These activities are different because there are not the same processing steps for an electronic order and one received by phone. Most importantly, by questioning these activities, the necessary technological infrastructure and resources (Service Resource Assets) can be discovered. In the case of an order by phone, it becomes obvious, that a phone number (and thus a provider and telephone infrastructure) and probably a tool to enter the order is needed.

Service Level Objects

In the tool (eRanger), the activities are not stored but something called "service object". This could be for example a call-center for receiving orders or a webshop. This means, the processes are broken down until one comes upon a SLO (Service Level Object) which is something needed to perform an activity. However, it is seen as possible improvement to capture more information. Especially, when it comes to changes, it is not always visible, why certain objects are there. SLOs have the same core information like process objects - in particular a status and child objects.

Configuration Items

The systems layer is modelled as clusters of configuration items. The clustering is done to bring the discussion away from technical questions. Actually, it is an open issue, whether these clusters will be modelled in future. The completeness of a model is difficult to measure - actually the modeller never knows if everything has been considered.

Measurement Jobs

The lowest level (leafs) consist of measurement jobs (sensors) which monitor the availability of infrastructure elements such as ventilation, network, CPU or operating systems etc. Examples are a ping on a network device or a TCP request. If a ping replies, the server has flow. If the TCP request gives a respond, the web-server (incl. the operating system) is running.

Dependencies and Impact

All elements (also those on the lower layers) have a status which is visually represented in the software by displaying the respective element in green, yellow or red colour. Each element passes its status to its parent element which decides on its own status based on the status of the child elements. More concretely, there are measurement jobs or KPI information that

change the status of an object. Then, the status of the parent object is determined based on boolean operations. Precisely, the operations are AND, OR, MUST.

Examples are:

- if two underlying objects are red, the parent becomes yellow.
- if all underlying objects become red, the parent becomes red.
- if a certain object becomes red, the parent becomes yellow.

"MUST" means that the status is necessarily passed up: it is mandatory that a certain element has status green otherwise the parent changes the status (to yellow or red). This is actually a tool specific expression to simplify condition statements.

A further aspect is time: for example, the call-centre may be down (status red) for an hour without changing the state of the order process but after four hours, the process becomes the status yellow.

In the current tool version, temporal dependencies are not implemented in an "elegant way" but this is a requirement for the future development. An example from a running project: if the processing of certain data was successful is checked based on whether an e-mail has been received within a certain period that confirms the completion of the respective processing.

For the managers it is actually necessary to know, where they have to manage or workers have to know how they should behave respectively. Therefore, with a status, also textual information can be connected. For example, a text could state for a process, "if the status is yellow, then apply workaround X".

Measures

The calculation of business KPI (Key Performance Indicators) often require scripts which collect data from different sources and calculate the respective value (e.g. number of orders, open accounts etc.). Thus, those KPI are expensive, in particular also for maintenance, because they have possibly to be changed if a process changes. Currently a weakness is also, that there is no information in the model that states why a value is calculated in a certain way and why the value is of importance. How this could be included will be analysed in the recently started research project of Junisphere. In future it should also be differentiated between informative KPI and status relevant indicators. E.g. the daily number of orders can be a useful information but without influence on the process status. On the other hand, the influence may depend on ranges and times. If, for example, the number of orders is outside a certain range, possibly for a longer time period or during a certain time window such as from 10 am to 1 pm, it may change the status. Such time windows may even be further constrained to time: e.g. the normal buying behaviour may differ on certain weekdays or phases such as Christmas time. Thus, fixed and dynamic thresholds are demanded.

KPI are developed in interaction with the client: "how do you appraise, that the order process runs smoothly?" There may be quantitative, qualitative and financial indicators, e.g. time to enter an order, number of orders per day, cost of entering an order. Often the initial KPI and correlation rules do not lead to a satisfactory result. They are then reworked and often, it is difficult to see, why a certain measure or rule had been applied when changing the system later.

The model does not capture single process flows (e.g. one execution of a task). There is one process object (per process) and the current situation is represented by metrics (KPI, Status).

5 Applications and Models of Enterprise Architecture

This chapter corresponds to the problem awareness phase of the design research method in the sense of becoming aware of application requirements, considered as problem to be solved or approached. The actual goals and applications of enterprise architecture have been identified and collected within the literature review. The following sections derive and recapitulate the most important applications and corresponding modelling languages of an enterprise architec-

ture based on the literature review and the interview. With that, this chapter founds the answer to the first research question.

> Research Question 1:
>
> What are the most relevant applications for enterprise architecture and what models (and corresponding languages or subsets of them) are required for those applications?

These applications are considered as high-level requirements from a user perspective. As such, this chapter provides an important input to the second question, which is dealt with in the subsequent chapter.

5.1 Relevant Applications

With respect to the development process of enterprise architectures and the use of the outputs (i.e. models), the following applications are regarded as the most important ones to be further investigated in this work:

- Representation of models and languages

  A basic task within enterprise architecture development is the representation of as-is and to-be models and thus also the modelling languages used for modelling. Modelling is a step towards the goal of gaining transparency about the enterprise structure and the relations between different enterprise domains. Therefore, the representation also has to allow for relating elements of different models to each other. This application includes the capability of defining and extending modelling languages (i.e. define meta-models).

- Verification of Models

  From the viewpoint of tool support it is seen as relevant to ensure, that models are correct with respect to the modelling language specification (e.g. language grammar rules). A tool should guide the user by only allowing the intended use of model elements or at least check the correctness. This partially ensures that models are unambiguous and can be used as input for further applications such as the generation of executable models. Model checking may also be more comprehensive by for example checking modelling guidelines.

- Enterprise Development

  The continuous improvement and change of an enterprise involve intermediate architecture states at different points in time. Especially fundamental transitions may require stepwise planning. Therefore, architecture models should be traceable with respect to version control, validity periods and reasons for change. Also the comparison of different versions is useful.

- Analysis of (alternative) architectures

  As one of the major goals of EA is the improvement and support of change, the analysis of architecture models is seen as important application. For example, structural analysis can indicate the impact of changes; quantitative analysis may be used to estimate runtime performance or workloads and dynamic analysis can be used to simulate and optimise dynamic systems.

- Compliance Management

  As-is and to-be models should consider internal and external regulations (e.g. laws, enterprise goals and corresponding policies). This can be supported at design time by checking models against corresponding constraints but also at run-time (e.g. during process execution) as well as a posteriori by analysing execution data.

- Model Execution

If models can be transformed into executable form, this helps to ensure that the actual implementation is aligned with the business needs and goals. It further allows for faster implementation processes and thus improves business agility. Relating execution data with the architecture models supports monitoring, analysis and compliance management.

- Execution Monitoring and Analysis

  When a new architecture is implemented, it is important to evaluate whether the intended goals were reached. Execution monitoring and analysis allows for detecting problems, determining need for improvement and measure the actual (IT) performance as compared to the business goals.

5.2 Junisphere Application

The findings of the interview results are described in section 4.3. This section briefly recapitulates the application and addresses its relations to the applications of the previous section.

The business service management scenario resulting from the interview is strongly related to the execution monitoring application. The major goal is to enable runtime monitoring of processes and systems involving system statuses, their impact on processes and performance indicators. First, a rather simple structure of elements from the business and the IT perspectives with their dependency relations has to be captured. Though, the impact of problems (related to the structural analysis) is defined fine-grained throughout the dependency tree and in practice the relevant knowledge therefore is gathered within workshops. According to the interviewee, the modelling process and the necessary analysis of the infrastructure and dependencies provide transparency and often support the communication between business and IT what confirms an often stated benefit of enterprise architecture modelling. The interviewee stressed the problem of traceability several times. First the relation between (sub-) processes and service level objects is difficult to comprehend in hindsight as the model does cover the process details (i.e. activities). Second, as the reason why measures and their impacts were specified in a certain way is not documented, changes to the model become difficult.

The company so far does not use a common or standardised graphical modelling language or modelling tool. The information gathered in workshops is defined with forms and depicted with office tools and finally entered as structured data into the system.

5.3 Languages and Language Subsets

The importance of specific languages is varying between the domains. UML is the predominant language for the IT domain; within the other domains a comparable prevalence seems not to be given (Lankhorst 2009). However, the websites of prominent tool vendors such IDS-Scheer (ARIS), Mega, Metastorm and Sparx contained in Gartners magic quadrants (Handler & Wilson 2009; Wilson & Short 2010) show that the support of BPMN is also very widespread. BPMN is, like UML, a rather well specified language. Further, BPMN supports different levels of abstraction: it is capable of capturing high level perspectives (as often required in EA), e.g. process landscapes, but it also specifies execution semantics and a mapping to the executable BPEL standard.

These two languages will therefore been taken as reference with respect to the representation of concrete languages and, if appropriate, to exemplarily testing the applicability of the semantic languages against application requirements.

Even though several applications have been analysed (chapter 78) it was not possible to clearly define a language subset required for enterprise architecture. Some applications, e.g. model checking are not directly related to a specific model type. Others, for example analysis and monitoring (including the interview case) focus on processes, systems and applications/services. The concrete models used for enterprise architecture also vary between different architecture methods. Besides IT components, services and processes, the relation to organ-

isational aspects (e.g. organisation units and roles) are seen as very important, because this refers to responsibilities and relate to compliance issues. Further the representation of (business) rules has turned out to be of importance to represent policies and to govern process execution.

5.4 Chapter Summary

By synthesising goals, applications and concrete tool functionality from the literature review, I argued in response of the first research question that:

a) The most important applications for enterprise architecture are (besides modelling and model verification) planning and analysis of enterprise architecture models, governing and supporting the implementation of changed architectures up to model execution, ensuring compliance of models (and the implementation) with internal and external regulations and to monitor and analyse the execution in order to close the loop from planning over execution to measure and further improve the implemented architecture.

b) The most important models are process models and their relation to application- and system models. Besides the organisation and people aspects which are in particular related to the processes, rules to formulate policies and guidance are of great importance.

c) With respect to their application in enterprise architecture software, UML and BPMN are seen as the most important languages. No definite subset of these or other languages could be identified or derived during the research activities.

6 Representation Requirements

This chapter reflects the suggestion phase of the research method. The outcome in the methodological sense is a tentative design. In this concrete case, the user- and application requirements are mapped to concrete features of the representation language. During the iterations of the research procedure, the mapping was continuously reworked and resulted in a high level overview of the applications (regarded as user requirements) and the corresponding language requirements (denoted as language characteristics).

With the mapping of application requirements to language characteristics this chapter in particular answers the second research question.

> Research Question 2:
>
> What are the relevant requirements originating from the purposes and applications of the models and specifications of the modelling languages?

The chapter is structured as follows: Section 6.1 recapitulates the application requirements based on chapter 5 and discusses the requirements of the Junisphere scenario. Section 6.2 presents the language characteristics considered as representation requirements. 6.3 addresses the fulfilment of these characteristics by RDF(S), OWL and SWRL. Section 6.4 contains the mentioned mapping between the application requirements and language requirements (characteristics). Finally section 6.5 summarises the chapter by formulating an answer to the second research question.

6.1 Application Requirements

The interview conducted with Junisphere directly lead to a rather detailed view on requirements, whilst the applications identified in the literature review are presented on a rather abstract level initially. Both views are presented hereinafter.

6.1.1 Requirements from Literature

The following structure represents the applications regarded as high-level requirements from a user perspective as motivated in section 5.1. The representation of models and languages is subdivided into several aspects which are required for different other applications. The differentiation is further described in chapter 7. Analysis is divided according to the aspects described in section 3.1.2 due to different representation requirements.

- Representation of models and languages
    - Structural Representation
    - Syntactic Representation
    - Semantic Representation
    - Meta-Modelling
- Verification of Models
- Enterprise Development
    - Versioning and Annotation
- Analysis of (alternative) architectures
    - Static Structural Analysis
    - Quantitative Analysis
    - Dynamic Analysis
- Compliance Management
- Model Execution
- Execution Monitoring and Analysis

6.1.2 Requirements for the Junisphere Application

The information gathered in the interview probably not reflects every detail of the methodology and the software of Junisphere's business service management approach. However, combined with the concrete examples of the experienced interviewee, the inputs can be used as a concrete application scenario in a size that can be dealt with as a whole. Of course this does not result in an encompassing real world scenario as compared to a concrete implementation of business service management for a specific company. Consequently, the application requirements addressed hereinafter are represented in more detail divided into elements, relations and functional requirements. The further development of the Junisphere application scenario is described in chapter 8. In the representation requirements mapping of section 6.4 the application is referred to as business service management.

Basic element types and properties have to be represented, roughly:

- Process Objects
- Service Level Objects
- Configuration Items (which can be clustered)
- Jobs

The elements have to be related to each other by:

- Child relation (between objects of the same type)
- Dependency relation (between different element types top down)

From a functional viewpoint, the following requirements are given:

- Each element has to know the dependent elements or the child elements respectively.

- Each element can change its status based on the status of the elements connected with either one or another relation type.
- An element may decide, which status the other elements have to consider ("the element passes its status to the parent elements").
- A job can change its status based on further information (e.g. provided by a monitoring program if a server is not available).
- Correlation rules can be defined very specific. The status of an element is determined based on the status of its child elements with arbitrary Boolean conditions (AND, OR) with respect to:
    - the status of one or more specific child elements
    - the status of a certain number of child elements
- Additionally, time based conditions related to the status (i.e. duration) and numeric comparisons (i.e. KPI values) must be possible.
- The relations between service- and process-objects and the reason for using certain measures must be traceable.

6.2 Language Characteristics

Table 5 lists the major characteristics which turned out to be of importance to represent EA models in order to fulfil certain application purposes (i.e. by reasoning support). Several of the criteria are related to the closed world assumption (e.g. exclusive disjunction, prioritisation and defaults). They have been separated because it seems more intuitive to associate them with certain applications. In addition it is rather inconvenient to express them by using negation, thus, corresponding features may be useful.

Characteristics	**Description**
Open World Assumption	"Open world semantics allows that some knowledge may be incomplete; so if a proposition and its negation are both absent, it is unknown whether the proposition is true" (OMG 2008).
Closed World Assumption	"Adopting closed world semantics basically means that all relevant facts are known […]. So if a proposition cannot be proved true, it is assumed to be false" (OMG 2008).
Alethic Modality	Interpreted according to the descriptions of the SBVR specification (OMG 2008): Alethic modality refers to a formulation of necessity, whereas the stated fact is taken for true in all possible worlds. It is thus not possible to violate such a necessity, e.g. it is impossible that a person was born at more than one place.
Deontic Modality	Interpreted according to the descriptions of the SBVR specification (OMG 2008): Deontic modality refers to a formulation of obligation, which is taken for true in all acceptable worlds. Facts which violate such an obligation may exist but could have consequences (e.g. a penalty for parking violation).
Quantification	Quantification refers to property restrictions - i.e. propositions about the existence of relations. This includes existential and universal as well as concrete cardinalities (exact, min, max).
Negation	Expressing negated propositions such as "car movement that is not round-trip" defining a one-way car movement in SBVR (OMG 2008).
Disjunction	Disjunction is a "logical operation that formulates that the meaning of at

	least one of its logical operands is true" (OMG 2008).
Exclusive Disjunction	A disjunction where only one of the operands may be true but not both.
Conjunction	Conjunction is a "logical operation that formulates that the meaning of each of its logical operands is true" (OMG 2008).
Dynamic constraints	Dynamic constraints refer to a constraint regarding transitions or comparing states at different points in time.
Element Aggregation	An aggregation in the sense represented by the given criteria refers to compositions (in different flavours). In particular, the part-of relation (parts that make up the whole) is considered. Several discussions can be found on the topic of aggregation and composition and - different applications will cover different meanings (what is probably the reason, that OWL still has no specific language features to cover it).
Taxonomic Relations	Taxonomic relations refer to specialisation and generalisation of terms. E.g. A fast processor is a processor that has more specific characteristics (specialisation) than a processor in general (e.g. it has a high clock frequency).
Non taxonomic relations (Properties)	The characteristic of supporting non taxonomic relations refers to any kind of relation that does not indicate a specialisation or generalisation but allows for relating things with other things or data.
Arithmetic Aggregation	Arithmetic aggregation refers to computational features, such as counting individuals or make calculations over the values of a specific set of elements. This can be compared with database operations (e.g. found in SQL) such as the calculation of sums, averages, minimums, maximums etc. over a set of database entries restricted by query conditions (including grouping etc.).
Arithmetic Operations	Calculations and comparison of numeric values not covered by quantification.
Meta-modelling	Meta-modelling refers to the question, whether classes, instances and also properties are strictly divided. A set of classes with its constraints could be considered as meta-model for possible constellations represented by individuals. Meta-modelling (as reflected by this criteria) further requires, that these individuals could also represent classes having further individuals as their instances.
Relational Comparison	It is difficult to name and describe this characteristic appropriately. To describe some limitations of OWL, Ghidini, Rospocher, & Serafini (2008) referred to the point, where more than two variables were needed to represent something in FOL. This is not precise enough to state what is thought of with "specific comparison". I we think of a fast computer which is defined something which has a fast processor (CPU) and a fast hard-disk (HD), we would need three variables in FOL: $\forall x, y : hasCPU(x, y) \wedge FastCPU(y) \wedge hasHD(x, z) \wedge FastHD(z) \rightarrow FastComputer(x)$ But we could still state this in (OWL) DL: $FastComputer \equiv (\exists hasCPU.FastCPU \cup \exists hasHD.FastHD)$ The characteristic is used to refer to situations, where we want to make statements about two things which are related to another thing and have to be compared (e.g. equal or different). An example was the condition, that an element in a process model may

	only have message flows to elements of another pool. To state such a condition, we need to express, that the target of any "message flow" relation is not in the same pool as the source. Thus, the pool related to a target has to be compared to the pool related with the source.
Defaults	In traditional programming language, one often finds "if…then…else" constructs. If a condition is not satisfied, the "else" case should apply. Defaults (or initial) values refer to consequences if none of certain set of rule conditions hold. Defaults can be expressed by negation under closed world assumption.
Prioritise	If more than one condition is fulfilled (e.g. multiple rules), it was desirable to apply only one of the consequences - the one whose condition is considered most important. This is closely related to defaults. With a closed world assumption, priorities can be formulated by negation of other conditions.
Property Characteristics	The specialisation of properties according a predefined semantics such as inverse, functional, symmetric etc.
Time operations	Time operations can be date comparison (e.g. before, after), calculation of date differences and durations.
Annotation	In particular versioning information but also explanations can be annotated.

Table 5: Language characteristics that turned out to be of importance

In (de Bruijn et al. 2005) restrictions are seen as part of the logical theory of an ontology which therefore restrict the number of possible models or increase the set of consequences respectively. In contrast to that, they consider constraints not as part of the logical theory but rather as integrity constraints which allow for checking a knowledge base for violations but without influencing the set of logical consequences of the ontology.

This differentiation is not reflected in the table. Although there are two criteria regarding constraints, they refer to what the subject of an expression can be, rather than the logical embedding of the expressions in the language.

From an application point of view, it seems important, that the constraints can be checked, not how the formalism is embedded in the language. In this work, checking (integrity) constraints is seen as application. The characteristics refer to different parts which are necessary to support the application. For example, there are characteristics which refer to the means to express conditions (e.g. quantification and conjunction). Other characteristics are related to the way things may be expressed to reach the intended behaviour (e.g. modality). In the case of constraints checking, the world of discourse is considered as the most important factor: Based on the conditions (e.g. cardinality restrictions), a reasoner can classify things and detect inconsistencies as well. But because of the open world assumption, some conditions cannot be proofed.

6.3 Language Support

Table 6 indicates which of the characteristics described in the previous section are supported by which language. The table does not distinguish between profiles of OWL because the differences between the profiles are on a rather detailed level, whilst for a primary selection between languages rather rough criteria seem useful. For example, the OWL specification (Bao et al. 2009) lists ten features to restrict object properties and another nine for data property restrictions. To select between OWL, RDF(S) and SWRL, it is first of interest, whether any kind of quantification needs to be expressed at all. In a second step, a more detailed selection can be made.

Some of the characteristics such as the modalities are indicated to be expressible in more than one language but the selection of the language (OWL or SWRL in this case) has an impact on how consequences (e.g. of a deontic rule) can be expressed. An exemplary deontic (operative)

rule such as "It is obligatory that the rental duration of each rental is at most 90 rental days." (OMG 2008) could be reformulated in OWL to classify a "Rental" as "ContrarianRental" given the following sufficient condition for the class "ContrarianRental":

Rental and (rentalDuration min 1 int[> 90])

In SWRL it is also possible to add property values instead of having additional classes which indicate violations. Notice that it has also to be considered, how the facts - e.g. rentalDuration - are determined. If "rentalDuration" was a system input when a car had been rent out, the given example could apply. If, otherwise the duration had to be calculated from two dates, only SWRL was possible, whilst, if the date had to be determined based on a running rental with respect to the current date, this would correspond to dynamic constraints which could not be formulated by any of the given languages (without programmatic control).

Chapter 7 discusses such considerations in more detail and in the context of concrete applications.

Characteristics	**RDF(S)**	**OWL (DL)**	**OWL (FULL)**	**SWRL**
Open World Assumption	Y	Y	Y	Y
Closed World Assumption	N	N	N	N
Alethic Modality	Y	Y	Y	Y
Deontic Modality	N	Indirect*	Indirect*	Indirect*
Quantification	N	Y	Y	N
Negation	N	Y	Y	N
Disjunction	N	Y (Union)	Y (Union)	Y (Multiple Rules)
Exclusive Disjunction	N	N	N	N
Conjunction	N	Y (Intersection)	Y (Intersection)	Y
Dynamic constraints	N	N	N	N
Element Aggregation	N	N	N	N
Taxonomic Relations	Y	Y	Y	Y
Non taxonomic relations (Properties)	Y	Y	Y	Y
Arithmetic Aggregation	N	N	N	N
Arithmetic Operations	N	N	N	Y
Metamodelling (instance of an instance)	Y	N	Y	N*
Relational Comparison	N	N	N	Y

Defaults	N	N	N	N
Prioritise	N	N	N	N
Property Characteristics	N	Y	Y	Y**
Time operations	N	N	N	Y
Annotation	N	Y	Y	Y

Table 6: Support of language characteristics

Indirect*: The given languages are rather close to alethic interpretations: Statements made about a class are actually taken for true. If an individual is asserted as member of the class, the statements of the class are taken for true and lead to an inconsistent knowledge base, if facts proven to be false. However, it is possible to define classes or properties and to classify individuals (or infer property values via SWRL) indicating the violation of deontic constraints.

(**) Property characteristics are not part of the SWRL vocabulary, but some logical consequences (such as inversion, property chains or transitivity) can be expressed.

6.4 Requirements Mapping

With Table 7 this section provides an overview of the representation requirements (columns) identified for each of the applications or the corresponding application requirements respectively (rows). Explanations about the selection of representation requirements can be found in chapter 7 which is structured according the applications.

It is important to consider that the mapping does not claim to completeness. It can not be stated that there were no further representation requirements. It is also possible, that parts of an application can be fulfilled with only some of the representation requirements.

The representation requirements stated for structural, syntactic and semantic representation of models were not repeated in general but mainly those requirements which appeared to be of particular importance for each application.

The most representation requirements are selected for the Business Service Management application what is a result from the concrete implementation of a scenario. It is noticeable that several applications are rather related to a closed world assumption (CWA) or to representation requirements which are related to CWA (e.g. supportable by negation). No application was identified to be explicitly related to an open world assumption. Several applications are also marked as requiring meta-modelling. Regarding meta-modelling, chapter 7 provides more detailed considerations. Execution monitoring and analysis for example is conceivable without meta-modelling but seem to profit from it.

	Open World	Closed World	Alethic Modality	Deontic Modality	Quantification	Negation	Disjunction	Exclusive Disjunction	Conjunction	Arithmetic Operations	Dynamic constraints	Element Aggregation	Taxonomic Relations	Non taxonomic relations	Arithmetic Aggregation	Meta-Modelling	Relational Comparison	Defaults	Prioritise	Property Characteristics	Time operations	Annotation
Structural representation of models and modelling languages	-	-	-	-	-	-	-	-	-	-	-	-	X	X	-	-	-	-	-	-	-	-
Syntactic representation of models and modelling languages	-	-	-	-	X	X	X	-	-	-	-	-	-	-	-	-	X	X	-	-	-	-
Semantic representation of models and modelling languages	-	-	-	-	-	-	-	X	X	-	X	X	-	-	-	-	X	-	-	X	-	-
Meta-Modelling	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-
Verification of models (includes syntactic representation)	-	X	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Support for enterprise development	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X
Support for static structural analysis	-	-	-	-	-	-	-	-	-	-	-	-	X	X	-	X	-	-	-	-	-	-
Support for quantitative analysis	-	-	-	-	-	-	-	-	-	X	-	-	-	-	X	-	-	-	-	-	-	-
Support for dynamic analysis	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-	-	-
Support for compliance management	-	X	X	X	X	-	-	-	-	-	-	-	-	-	-	X	X	-	X	-	-	-
Support for model execution	-	X	-	-	X	-	-	-	-	-	-	-	-	-	X	X	-	-	X	-	-	-
Support for process monitoring and analysis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	X	X	-	-	-	-	X	-
Business Service Management	-	-	-	-	X	-	-	-	-	X	X	-	X	X	-	-	X	-	X	-	X	X

Table 7: Representation Requirements (mapping between applications and language requirements)

6.5 Chapter Summary

This chapter presented the applications regarded as relevant considered as high-level requirements (in terms of tasks to be supported) and the requirements of the Junisphere application in a more detailed fashion.

Further, the language characteristics (in terms of representation requirements) which were initially developed in the suggestion phase and continuously reworked during the subsequent phases have been described and related to a) the languages of discourse, RDF(S), OWL and SWRL and b) to the applications.

The answer to the second research question (the relevant requirements originating from the purposes and applications of the models and specifications of the modelling languages are) is answered in two directions: first in form of the characteristics described in section 6.2 considered as representation requirements and second by the requirements from an application point of view described in section 6.1.

The set of characteristics is seen as an important contribution of this work, because it can be used for selecting a representation language also apart from the languages considered here, by extending the language mapping of 6.3. The requirements mapping table in section 6.4 allows for selecting the application-requirements which should be supported in a certain case (e.g. tool development or enterprise architecture management in a company) whereof the corresponding representation requirements (characteristics) become clear. The mapping between the characteristics and languages of 6.3 can then be used to select an appropriate language.

7 Enterprise Architecture Representation Analysis

The design phase of the design research procedure followed in this work is represented in two chapters. This chapter deals with the more general applications whilst the subsequent chapter (8) addresses the concrete application scenario based on the interview with Junisphere.

The previous chapter 6 already provided an overview of the language characteristics required to support applications of enterprise architecture. Following an iterative approach, the findings of the design phase presented in this chapter are also considered in the previous chapter. Consequently, this chapter provides the reasoning for the representation requirements presented in 6.4.

> Research Question 3:
>
> How can these requirements been fulfilled by one of the languages RDF(S), OWL and SWRL or a combination?

Following the third research question, this chapter analyses several applications and addresses possibilities to support them by means of RDF(S), OWL and SWRL. Approaches found in literature are considered and compared and alternative representation forms are described where possible. The result as a whole thus answers the third sub-question (supplemented by the scenario dealt with in chapter 8) and leads to observations and findings regarding the fourth research question (shortcomings and strengths).

The first section hereinafter addresses the basic representation of models (and modelling languages). In the subsequent sections the applications - verification, enterprise development, model analysis, compliance management, model execution and execution monitoring and analysis - are examined.

Each section summarises the identified representation requirements and the most suitable language(s). The summaries of the first section (7.1) are slightly different because the model/language representation founds the basis of the applications dealt with in the other sections. The representation requirements of the applications build upon on the model/language representation and are not repeated (e.g. it is not mentioned for each application that non-taxonomic relations are needed).

In some cases RDF(S) is indicated as in combination with rules. In these cases the existence of a rule language for RDF(S) with features similar to those of SWRL is assumed (syntactic constructs and built-ins, not the features based on OWL).

7.1 Representation of Models and Languages

As introductorily mentioned this section treats the most basic requirements - the actual ontological representation of models and their languages - before more functional requirements (applications that make use of the models) are covered.

The representation is divided into four sub-sections, each representing different aspects of (meta-) model representation which were identified to require different expressive means and support different applications: Structural representation, syntactic representation, semantic representation and meta-modelling.

According to the explanations in section 1.5.2 the syntax includes the elements and the syntactic rules of a language. Therefore the separation between "structural" and "syntactical" representation is contrasting. However, the separation has two reasons: First, it allows structuring the discussion according to different focuses, which is the main reason. Second, some basic goals of enterprise modelling can be covered with the notion of structural representation.

The four focuses are briefly introduced according to a small example model depicted in Figure 20. The simple graphical process at the bottom of Figure 20 simply consists of two activities

connected with an arrow. On the ontology level, each of the three parts is represented as individual of either "#SequenceFlow" or "#Activity". The sequence flow relates the two activities with the properties "#flowSource" and "#flowTarget".

This example represents the notion of the structural representation as addressed in sub-section 7.1.1: there exist some concepts and properties representing a metamodel (the meta-meta model is omitted for reasons of simplicity) and some elements related by the properties representing a model. Thus, in this sense, the notion of "structural" does not refer to the difference between structural and behavioural aspects.

In such a structural representation, instances of activity and sequence flow could be arbitrarily connected. To restrict the use of the elements and properties, grammar rules have to be introduced. A simple grammar rule may for example state, that the two properties may only be used for relating sequence flows to with other elements. This level of representation is considered in sub-section 7.1.2.

Restricting the allowed or possible combinations of model elements still does not cover the intended meaning. The sequence flow for example would indicate that the activity "prepareOrder" is executed first and "deliverGoods" afterwards. Sub-section 7.1.3 refers to this kind of model representation.

With meta-modelling, sub-section 7.1.4 discusses an aspect of the model representation which does not directly depend on those representation levels (structural, syntactic and semantic). Meta-modelling refers to the upper level of Figure 20 and the meta-metamodel (not depicted) but in particular 7.1.4 is about the instantiation level: is "#prepareOrder" actually an individual?

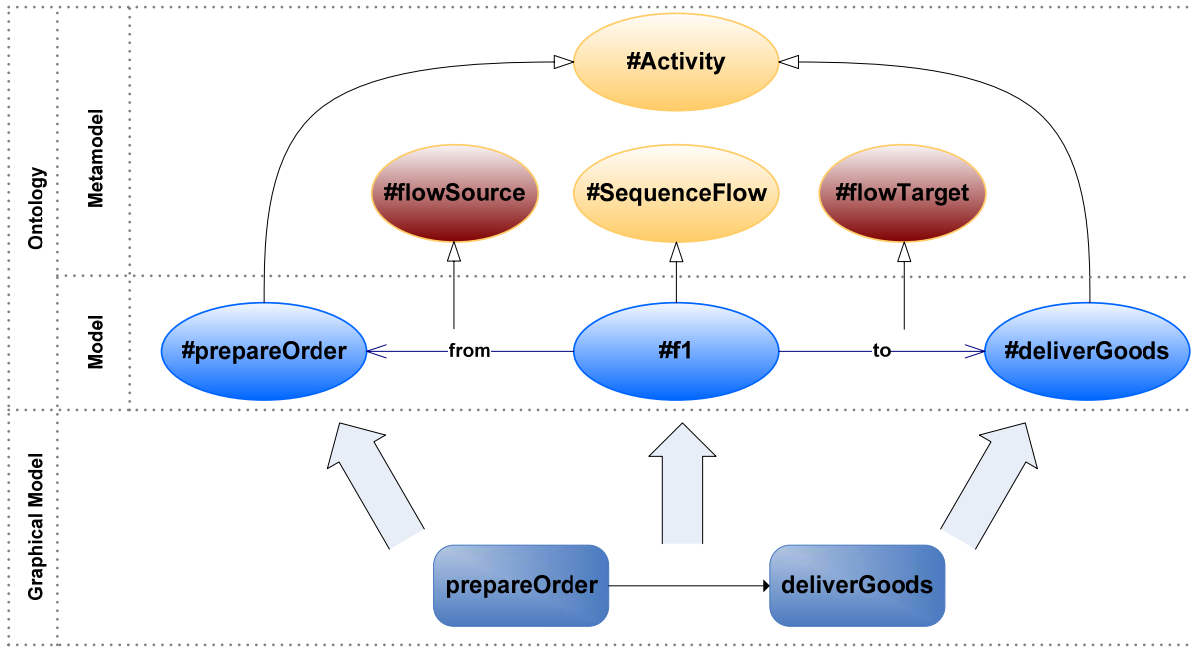

Figure 20: Exemplary Model Representation

7.1.1 Structural Representation

The most basic requirement of model representation is the ability to represent elements and their relations - independent of the syntactic constraints and without focusing on semantics of a model beyond taxonomic and non taxonomic relations. The representation of models using RDF(S), OWL and SWRL is of course a semantic representation as the languages define a formal semantics (i.e. interpretations and inferences). But, as exemplified before, the meaning of a modelling construct (e.g. the connection of two elements) is not necessarily covered by simply using a non-taxonomic relation.

In the context of the open source tool ATHENE[24] (Hinkelmann, Nikles, Thönssen, et al. 2007) it has been proved[25] that - at least - models which form a graph (i.e. nodes and connecting edges) can be represented by means of RDF(S): On the meta-model level, a meta-model class can be defined and related to a set of possibly contained elements. Each of these elements can be defined as a class with a set of properties (being data or objects) and on the model level, instantiations of these classes constitute concrete models with corresponding relations between elements.

Based on the properties defined on the meta-model, in ATHENE it is further possible to relate any element with other elements of the same or of a different model. With that, a fundamental requirement is fulfilled: relating models of different domains.

Summary: Structural Representation of Models

Work in the context of ATHENE has shown that meta-models and models can be represented with basic constructs such as classes, subclasses and properties. Different models can be represented and related. The goal of a transparent view of an enterprise as a whole seems to be sufficiently supported (in particular complemented with a visual representation, which is not treated in the work at hand). RDF(S) is seen as adequate language for structural representation.

Structural Representation of Models	
Supported goals and applications	Representation of entities and their relations. Transparency of relations between different domains.
Language Requirements	Taxonomic Relations Non-Taxonomic Relations
Supporting Language	RDF(S) is adequate for this purpose. All language profiles (or sub-languages) of OWL basically support this requirement.
Requirements Fulfilment	Completely (ignoring tool support)

Table 8: Analysis Summary - Structural Representation of Models

7.1.2 Syntactic Representation

The following paragraphs concentrate on BPMN to analyse, how grammar rules can be added to a structural representation.

Cardinality Restrictions and Property Characteristics

Basically the syntactic rules require (local) domain and range restrictions. In the BPMN ontology (Ghidini et al. 2008) we find in particular universal, existential and quantified property restrictions combined with union and intersection but also inverse properties for connecting elements (e.g. sequence flow).

The Need for Rules

Ghidini et al. (2008) mentioned two restrictions of OWL in the context of their BPMN ontology. First, that some properties would require more than two variables, if translated to first order logic. One example thereof was with respect to the uniqueness of the object identifier. This ex-

[24] The ATHENE tool: http://147.86.7.23/athenewiki

[25] The author of this work was involved in the development of the ATHENE tool and the within the scope of his master thesis, Roman Brun developed several model types (i.e. metamodels) including processes, business motivation and organisation (Brun 2010).

ample, however seems to be coverable with an inverseFunctional property - e.g. "id" - we could state, that an object may only be connected to one individual by "id". A new feature of OWL 2 are keys which state, that a classes' individuals are uniquely identified by the value of property (or properties) declared as key (Golbreich & Wallace 2009). Thus, this was also an approach to represent the object identifiers.

The current BPMN specification contains statements affirming Ghidini et al. (2008). An example thereof is that "Message Flows cannot connect to objects that are within the same Pool" (OMG 2010). To represent such a condition, we must refer to both, the source and the (requested) target of a sequence flow as well as to the pool(s) they are contained in.

An OWL property restriction could refer to an object that has a property with a certain range, e.g.:

```
Class: MessageFlow
        SubClassOf:
                (flowSource only (FlowNode and (inPool some Pool)))
                and (flowTarget only (FlowNode and (inPool some Pool)))
```

It is even possible to place further restrictions on "Pool" based on its property ranges, but it is not possible to state anything about the Pool which contains the FlowNode related by flowSource with respect to the other Pool related by flowTarget.

With a SWRL rule variables can be used and we can make statements about their relations. The following example states that, if the source and target of a MessageFlow are each contained in a pool, then these pools must not be the same (using the SWRL expression "differentFrom").

```
MessageFlow(?mflow), flowSource(?mflow, ?source), flowTarget(?mflow, ?target),
inPool(?source, ?pool1), inPool(?target, ?pool2) → differentFrom(?pool1, ?pool2)
```

The question, how to usefully state syntactic constraints is discussed in the context of model checking.

Dealing with Default Values

The second limitation of OWL's expressiveness mentioned by Ghidini et al. (2008) is the ability to state default values. The latest BPMN specification (OMG 2010) includes the following occurrences of default values (incomplete example list):

BPMN Element	Attribute and default value	Usage
Documentation	textFormat (String), default: "text/plain"	Mime type of the contained documentation text.
Association	associationDirection ({None | One | Both}), default: None	Determines, whether there is an arrowhead and in which directions it points.
Gateway	gatewayDirection (GatewayDirection), default: Unspecified	Basis for constraining the number of incoming and outgoing sequence flows.
MultiInstanceLoopCharacteristics	isSequential (boolean), default: false	Determines whether activity instances will be executed sequentially or in parallel and controls the visual representation of the "loop" symbol of the activity.
Collaboration	IsClosed (boolean),	Determines, whether

	default: false	message flows which are not modelled could occur.
Activity	IsForCompensation (boolean): false	True would indicate, that the activity is only activated, if a compensation event was initiated.

Table 9: Examples of Default Values in BPMN

These examples show different impacts of attributes with default values: some may change the visual appearance, some influence the execution behaviour, some also change the grammar rules, and others decide on how to interpret content.

Simply declare no default value would pass the responsibility to programs which have to deal with the representation: e.g. a modelling tool has to decide on the default mime type to display documentation (or guess the type, if it is from a foreign source) or the default to be set when documentation is written. In (Nikles & Brander 2009) an attribute mapping is proposed for changing the visualisation of model elements dependent from attribute values. In the case that there is no value, the decision for using the default visualisation would require a close world assumption. In case of the gateway, as the default value is not constraining, there was no specific impact. The execution related attributes would, of course have to be interpreted by the execution- (i.e. workflow engine) or a transformation-program (e.g. BPMN to BPEL). In the context of semantic systems, the execution has not only to consider traditional workflow engines but strive for controlling the execution based on the semantic representation. These examples show that declaring no value is not a good solution, because an application would have to know the language in order to interpret and select the default (instead of having the information from the metamodel).

Defining additional attributes (declaring the default value) is probably not more satisfying, because these properties would somehow represent the "else" case that has to be considered, if no other condition was applicable (because no value was set and thus, the default should be considered) - what in turn (due to OWA) would require the proof, that there is no value.

What remains is to specify the attributes as mandatory to force the user to specify a corresponding value or to specify different classes for the different values (one having the default value) and forcing the user to select one of those during modelling. Both of these options result in the necessity of programmatic control. The Association element of BPMN could for example be represented by three classes with defined values for the "associationDirection" property. This would reduce the flexibility of modelling because the user would be forced to select one of these elements rather than choosing the default association and changing its direction attribute to "Both" or "One". Further, by specifying non-standard properties or classes/elements, the model interchange becomes potentially more difficult (e.g. the non-standard subclasses have no direct mapping to a target schema).

Summary: Syntactic Representation of Models

Based on existing work this section pointed out that in particular cardinality restrictions are required to represent grammar rules. It has also been shown that some constraints require the comparison of related elements, thus a rule language is needed. Additionally the handling of default values has been discussed, which is a limitation which could be solved with a close world assumption. Syntactic representation is considered as most important for checking models against language specifications (which is important for model exchange and transformation into executable formats) and also to define modelling guidance restrictions.

Syntactic Representation of Models	
Supported applications	Model checking (and execution).
Language Requirements	Quantification,

	Disjunction, Relational Comparison, Defaults
Supporting Language	OWL-DL plus SWRL.
Requirements Fulfilment	Partial: No Default Values

Table 10: Analysis Summary - Syntactic Representation of Models

7.1.3 Semantic Representation of Enterprise Models

This section discusses the semantics of modelling languages focusing on BPMN and UML. The semantics in these languages is textually described in the corresponding specifications (OMG 2010; OMG 2009a).

Sequence Flows

"A Sequence Flow is used to show the order of Flow Elements […]" (OMG 2010). The BPMN ontology (Ghidini et al. 2008) contains a class for sequence flows and properties indicating the source and target references to flow elements (cp. Figure 20). Further, there are some restrictions related to them. A human reader may infer at least a direction of the relation (because of the terms "source" and "target"), but, as indicated earlier, for an OWL reasoner, these terms have no further meaning. There is no actual knowledge about the order of the flow elements, being it related to time, space or even just a lexical or numeric order. In OWL, there was no possibility to state such a correspondence. In SWRL we could at least define a rule to infer a property value such as "before" or "after":

SequenceFlow(?sflow), sourceRef(?sflow, ?from), targetRef(?sflow, ?to) → before(?from, ?to), after(?to, ?from)

To make any use of this information, these properties ("before" and "after) have to be either used programmatically or further rules have to be defined in order to validate or analyse a model.

Exclusive Gateways

Another arbitrary example from BPMN is the exclusive gateway: there is actually no way to express an exclusive decision in RDF(S), OWL or SWRL. OWL can express unions (objectUnionOf) which are inclusive and SWRL has only conjunctions, therefore, only separate rules can express an "OR" but, as being independent of each other, they are not exclusive. Exclusivity could actually be expressed through negation but, with an open world assumption this can only partially solve the problem.

Aggregation

The representation of UML by means of OWL has been studied by several parties. A mapping between OWL and UML is described as informative (not normative) part of the Ontology Definition Metamodel (ODM) of OMG (OMG 2009b). Na et al. (2006a, 2006b) concentrated on the structural models (in contrast to behavioural models) and point out OWL's limitation regarding part-of relations (aggregation and composition). The representation of part-of/part-whole relations in OWL was investigated by different parties. In (Golbreich & Wallace 2009) several examples related to part-whole/part-of relations using OWL 2 features such as reflexivity and property chains are presented and it is stated, that most but not all common cases of part-of/part-whole relations can be expressed in OWL. Veres (2005) rather focuses on different meanings than aggregations as compared to UML, e.g. that a knife can be used as weapon but also as cutlery and thus their relation should not be taxonomic (but an aggregation of things which can be used as weapon). Veres (2005) presented three approaches and concluded that the grouping purpose was not covered completely by either one approach. Na et al. (2006a, 2006b) actually proposed properties named "is_part_of" and "is_whole_of" for the representation of aggregations.

The use of such properties, possibly with an existential quantification (or property characteristics) seems actually adequate for some basic UML aggregations, including the aspect that a part of a composition can only be part of one whole, a meaning stressed in the Ontology Definition Metamodel (OMG 2009b). However, shows a more complex example taken from the UML specification (OMG 2009a) depicting two views on the same class "Car". In the composition shown on the left side (i) an unrestricted number of wheels can be related to an unrestricted number of engines. The right side (ii) shows the same classes with further restrictions: only two wheels can be related to one engine, what only applies, if they play the role "rear" or "e" respectively. Further, in (ii), wheel and engine can only be connected if they play the corresponding role in the context of the same car. To express this, not only local restrictions are required but also variables are to state that the related car is the same for "Wheel" and "Engine". Thus a rule language is needed.

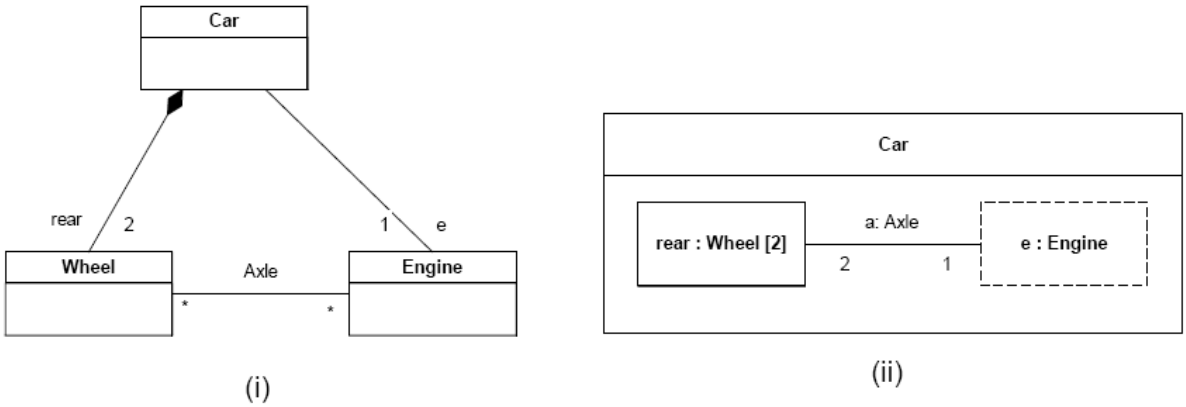

Figure 21: Context dependent cardinalities in UML (OMG 2009a)

Behavioural Aspects

Processes have a dynamic characteristic as they represent sequences of activities and often also data objects which are changed by the activities. Lankhorst (2009) divides the concepts of the ArchiMate language into three aspects: the passive structure, the behaviour and the active structure. Examples thereof are a business actor (part of the active structure) who is executing a process (behaviour) what changes some business objects (part of the static structure). Such behavioural characteristics can also be found in UML (OMG 2009a). UML contains structural diagrams such as class diagrams and component diagrams and behaviour diagrams such as activity diagram, use case diagram and sequence diagram.

There have been efforts to represent behavioural semantics in ontologies. In the SUPER project for example, the Behavioural Reasoning Ontology (BRO) was developed that formalises process algebra (Janusch 2009). However, the work also defined an operational semantic over the syntactic constructs (i.e. a transition system) - thus goes beyond the semantics of the languages of discourse, which consider the state of a world at a certain point in time. Behavioural equivalence is formalised in (Janusch 2009) by WSML-Flight rules, i.e. axioms defining behavioural equivalences. Besides the need of enhancing the reasoner with certain functions (e.g. the introduction of integer indexes), Janusch (2009) also mentions that reaching certain goals with the intended approach were limited by the need of closed world reasoning (i.e. in the context of removing parallelisms).

The ODM, besides aggregation and composition points out access control (e.g. visibility of properties such as public, private, and read-only) and especially behavioural features such as operations and active classes which specify different threads regarding their execution as features not contained in OWL (OMG 2009b).

Summary: Semantic Representation of Models

This section showed that there are several features in UML and BPMN which have no counterpart in the representation languages of discourse.

It has been argued, that some of the meanings can be represented with rules, for example the order of sequence. Whether the behavioural semantics of BPMN could be completely covered with these means could not be decided on in the scope of this work. Instead, it has been referred to existing work in this context which showed several limitations using comparable languages, in particular transitions (considered as dynamic constraints) and open world assumption.

With respect to UML it is questionable, how important some missing features are in the context of enterprise architecture. This cannot be answered coherently. UML strongly refers to software specification and how software should behave. Although it was desirable to express any kind of knowledge, it is not considered as aim to represent the fixed functional behaviour of software applications but rather at controlling the behaviour based on logical conclusions. Therefore, limitations such as visibility are not considered as being relevant for this work.

The relevance of the semantics discussed in this section is seen as relevant for execution and analysis of models.

Semantic Representation of Models	
Supported Applications	Execution, Analysis
Language Requirements	Property characteristics, Element aggregation, Dynamic constraints, Exclusive disjunction, Relational comparison
Supporting Language	OWL DL (ev. SWRL)
Requirements Fulfilment	Partially: No dynamic constraints (e.g. transitions) and no exclusive disjunction.

Table 11: Analysis Summary - Semantic Representation of Models

7.1.4 Meta-Modelling

The literature review has shown that meta-modelling in terms of defining or changing elements of a modelling language is a tool requirement. Work in the context of the ATHENE tool has dealt with this concern. This section focuses on meta-modelling in the sense reflected in the respective representation requirement (cp. characteristics in section 6.2).

If we think of a visualised element in a graphical model, there may probably be no other element with the same properties (such as shape, colour, label, position etc.). Even if so, each element can be distinguished (e.g. selected, moved and deleted). Thus in terms of a graphical model object, an instantiation was justified (i.e. representing elements in a model as individuals). If we instead think of the abstracted thing represented by a model, the differentiation between classes and instances becomes rather indistinct and dependent from the use of the model.

Different Aspects of Individuals

In a model that describes the IT infrastructure of a company, each element may correspond to a physical entity (i.e. any computer which is somewhere located in the company). In this case, representing these elements as individuals was probably appropriate. Taking for instance a certain computer in a company, it may be adequately described by its constant properties (e.g. size, colour, manufacturer and purchase date). Some properties even though may change over time: of course its age (which could be inferred from the purchase date) but also the location or configuration or even its hardware (e.g. a faster CPU, or another hard-disk). Taking shorter time

periods (or specific moments), its load factor or availability may differ. Going one step ahead, a human would not only reason that a computer with a new faster CPU was the same computer but also would have an increased performance and thus, probably a lower load factor.

This imaginary example basically justifies the representation of some model elements as individuals (e.g. concrete systems). It also exemplifies the dynamic of individuals over time whereas. Whilst some parts (e.g. detailed configuration or CPU) may be outside the scope of enterprise architecture, the state at different points in time becomes apparent in applications such as execution and monitoring and the quantitative analysis (e.g. load factors).

Individuals and Classes in a Model

For a UML class diagram it seems obvious, that each model represents classes. Another situation is given for business processes for instance. On a high level we may consider a process as a rough sequence of steps that create a service for a customer. On this level it may still be adequate to represent each step as individual - i.e. there is only one process step that takes orders and only one step that delivers the goods or invoices the service. Going into more detail, one may for instance find documents which are used in the process: a product catalogue, a guideline, an invoice or a bill of materials. Are these example documents individuals? A product catalogue or a guideline in a process model could represent specific documents, whilst an invoice would stand for all invoices created when the process is executed (e.g. related to a specific customer order). From this viewpoint, the same type of model element (document) may be used to represent an individual or a class and both could occur in one model.

Instances of Instances

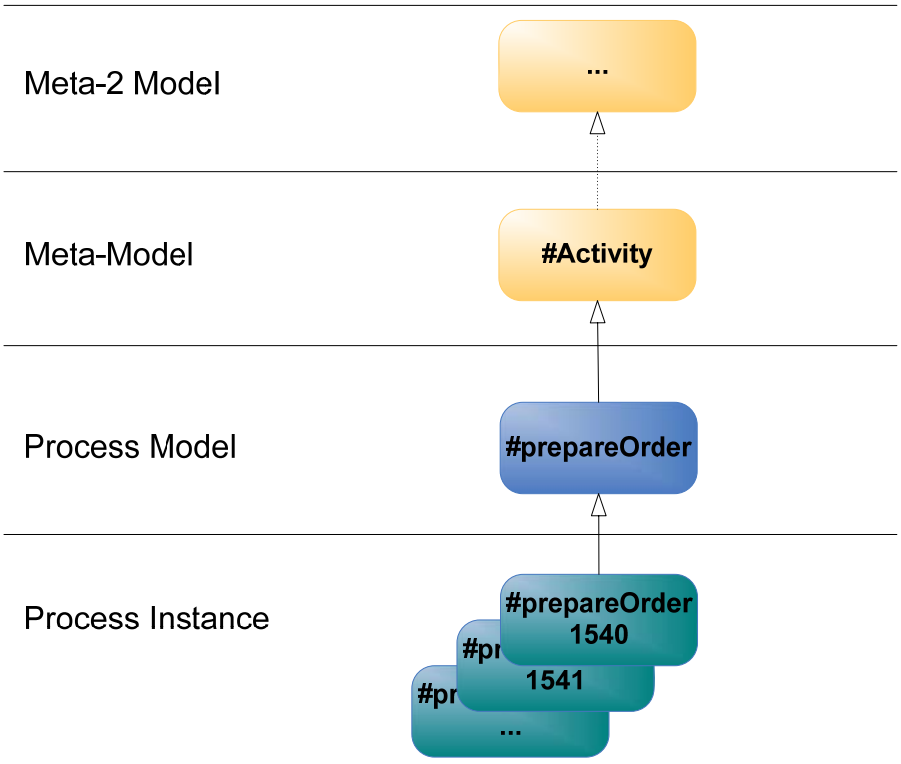

Figure 22: Process Model Levels with Execution

In the previous paragraph, the process elements are described as individuals, i.e. "prepareOrder" as instance of the class "#Activity" (cp. Figure 20 and Figure 22). The next level is the actual execution of a process. If the model is not only used as guidance for the actual implementation of process execution systems but should be executable, the whole process with all its activities would be instantiated. Figure 22 visualises this additional execution level. As compared to Figure 20 the "#prepareOrder" activity in Figure 22 has several actual executions instances (i.e. one for each order).

In BPMN (OMG 2010) this fact becomes apparent through the existence of so called "instance attributes" which refer to values which are not set at model time, e.g. the state of a process or the actual owner of a user task. BPMN further specifies a mapping to WS-BPEL to create executable models. By simply transforming a process model (represented as ontology) into BPEL one would offer up the possibility to use the knowledge contained in the ontology (the process model and its relations to other enterprise models) during runtime (e.g. for flow decisions and monitoring) and for later analysis (process mining). A consequence thereof is that the process instances and activities should be represented as instance of the modelled activities.

Representation Requirements

The examples discussed so far indicated, that the semantic representation of models must provide flexibility regarding specialisation in terms of classes, subclasses and instances. One model may contain elements represented as classes as well as instances. In case of executable models, the actual instantiation of a model may happen at runtime. Thus, for full meta-modelling support, instance of instances have to be supported by the language.

OWL DL strictly separates the terminology (classes) and assertions (individuals). OWL 2 (DL) has a new feature called "punning" that relaxes the strict separation between names of classes and individuals (Golbreich & Wallace 2009). With punning it is possible to use the same term, e.g. "activity" for a class and an individual (or property) in the same ontology. But though, an individual and a class with the same name are "interpreted semantically as if they were distinct" (Hitzler et al. 2009). Thus, it is still not possible to define an instance of an instance: having an individual "deliverGoods" and an individual "deliverGoods_1" of type "deliverGoods" would mean that there is a class "deliverGoods" which has basically nothing in common with the individual (i.e. super-class or properties).

Summary Meta-Modelling

This section showed that the separation of individuals and classes is not always clear on different levels (Meta-Model, Model) and motivated the need for meta-modelling features in terms of allowing the use of an element as individual and class.

It has been argued, that punning (in OWL 2) is not adequate to fully represent the relation between the model- and execution-instances. Full meta-modelling capabilities are provided by RDF(S) and OWL-Full), thus these languages are proposed as being adequate. The representations including the execution level without full meta-modelling capabilities are further discussed in subsequent sections, in particular in section 7.6 regarding execution.

Although meta-modelling is not an application but a language requirement, it may influence the language selection and imposes an often discussed and relevant limitation of OWL-DL, a language often selected because it offers expressive power whilst being decidable (and in particular supported by reasoners). All applications which are (partially) related to model execution would be supported by meta-modelling capabilities.

Meta-Modelling	
Supported applications	Model execution, execution monitoring and analysis and compliance management.
Language Requirements	Meta-Modelling (i.e. no strict distinction between individuals and classes).
Supporting Language	RDF(S) OWL-Full
Requirements Fulfilment	Yes

Table 12: Analysis Summary - Meta-Modelling

7.2 Verification of Models

This section discusses how syntactic rules or modelling guidance can be represented in order to check whether models conform to them.

7.2.1 Basic Problems

Two basic problems become apparent regarding model verification: The open world assumption and the question on how to deal with detected deviations from specifications.

Open World Constraint Checking

The BPMN ontology[26] does not generally ensure the satisfaction of the structural constraints it defines. As one of many examples, a business process diagram in BPMN is required to have one or more pools. The BPMN ontology reflects this with the property restriction (class name simplified):

```
SubClassOf: has_pools min 1 Thing
```

Due to the open world assumption (OWA) an ontology is not necessarily inconsistent, if the asserted facts do not correspond to cardinality restrictions. Thus, when adding the following individual assertion, the ontology was still consistent, because there is no fact telling that "example_diagram" does not have a pool:

```
Individual: example_diagram.
Types:
    business_process_diagram
```

Maximum cardinalities more likely lead to inconsistency when exceeding the restricted number of relations, but only, if the values (e.g. two individuals) are explicitly distinct.

Dealing with Non-Conform Representations

Restrictions which lead to inconsistent ontologies are possibly not most desirable way to verify models anyhow. In the context of enterprise architecture, models and knowledge in general should be related to each other. When a process model is only a part of a large enterprise ontology such a strategy would affect the whole ontology. Thus, it was a problem to store unfinished models (e.g. reasoners such as Pellet do not complete the reasoning, if an inconsistency was detected).

7.2.2 Alternative Representations

The following paragraphs discuss how constraints could be represented in order to deal with the problems mentioned in the previous sub-section, especially representations which do not lead to inconsistencies.

Classifying Non-conform Elements

One approach to check the model's structural conformance is to define a "warning" or "error" class and strive for classifying individuals which do not satisfy given constraints. The modeller could then be notified if such a class contains members. Defining such constraints in OWL would mean to invert or negate the corresponding conditions. Assuming distinct individuals, a maximum cardinality (e.g. <=1) could be expressed as >=2. The following example states that an individual which is a inclusive gateway and has at least two default gates would belong to the class "Error":

```
Class: Error
```

[26] ontoBPMN.owl, available from: https://dkm.fbk.eu/images/2/23/OntoBPMN.owl

```
        EquivalentTo:
            inclusive_gateway
             and (has_default_gate min 2 Thing)
```

The requirement, that a diagram has at least one pool was not fulfilled, if no pool was contained:

```
Class: Error
        EquivalentTo:
            business_process_diagram
             and has_pools exactly 0 Thing
```

However, the initial problem with the open world assumption stands: There is no fact which states that there is no pool, but just no fact is stating that there was one. Even though one of both statements will (due to OWA) not infer any member, the following expression shows how they were combined:

```
        EquivalentTo:
            (business_process_diagram
             and (has_pools exactly 0 Thing))
             or (inclusive_gateway
             and (has_inclusive_gateway_default_gate min 2 Thing))
```

Considering that this union only represents two simple restrictions and that the BPMN Ontology contains several hundred axioms, the disadvantage of the approach becomes apparent: Defining all restrictions in one class to represent deviations from structural rules would result in an enormous expression which was possibly hard to handle.

A further noteworthy point is that it was not possible to express both, the structural representation as given in the BPMN Ontology as well as classes to contain deviating individuals exemplified above as the conditions of the class "Error" could not be satisfied: If the class "business_process_diagram" states, that its members have at least one pool (this is taken for true), then consequently the intersection of things being a "business_process_diagram" and having no pool is empty.

Classifying conform Elements

There is also the possibility to define the constraints not as simple subclass expressions (necessary conditions) as done in the BPMN Ontology but as equivalent class expressions (necessary and sufficient conditions). This means, in contrast to the example of an "Error" class above, only individuals which meet the conditions will be classified. Consequently, the individuals in a model must not have an asserted type.

When ignoring further conditions (e.g. that the diagram may have a version, but not more than one), a diagram could be declared as follows:

```
Class: business_process_diagram
        EquivalentTo:
        SubClassOf: has_pools min 1 Pool
```

A foreseeable characteristic of this approach is the influence of the modelling process from a tool perspective. If the type of an inserted element is not asserted according to the intention of the modeller (i.e. by selecting a certain element to draw) but has to be classified, this would affect the tool performance. In addition handling the visual representation could be affected, even if the graphical representation is separated from the conceptual model by mapping graphical specifications to the conceptual elements as for example described in (Nikles & Brander 2009).

When an incomplete model is closed and reopened later, a tool has to somehow represent the elements: if for example an activity was inserted but could not be classified as such, the tool could not represent it according the visualisation class mapped to the concept "activity".

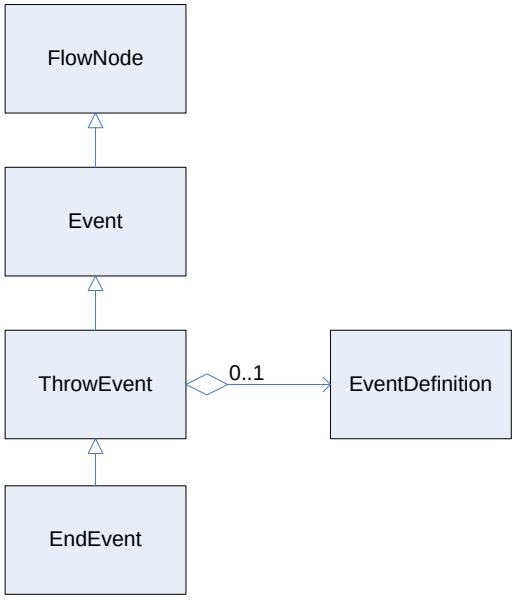

Figure 23: BPMN EndEvent classification (according to OMG 2010)

Independent from tooling, an important restriction of this approach is that no mutual constraints could be defined: A classification was not possible for an element *E1*, if it requires a relation to *E2*, whilst *E2* requires a relation to *E1*. Moreover it would require, that any class can be characterised sufficiently by its properties. Figure 23 depicts a excerpt of the classification of events as specified in current BPMN specification: None of the displayed elements (except for subclasses of EventDefinition) has mandatory attributes (cardinality > 0). Thus, an EndEvent cannot be classified, if it does not have a specific event type (defined by EventDefinition), what is not necessary.

The problem, that some elements may not have sufficient conditions to be classified could possibly be circumvented. In the case of "EndEvent", an additional "EventDefinition" could be introduced and asserted. But when new things are introduced, the model may at some point not conform to the specification anymore. When models are exchanged with other tools, such additional information probably would have to be excluded. In the given case, this was only a bypass to enable a solution that has several weaknesses and disadvantages as described above. Actually, a technology should be suited to solve a problem or to support certain tasks. If one has to alter the actualities to support a task with a given technology, there are potentially more suitable technologies.

Define Models at Class Level

Finally, there is the possibility to define models on class level only, e.g. each task of a process model as a subclass of the class "task", and check the concepts for satisfiability. The following example defines "ConcreteSequenceFlow " as subclass of "SequenceFlow" that exceeds the exact cardinality of 1:

```
Class: SequenceFlow
        SubClassOf:
            FlowElement,
            sequenceFlowSourceRef exactly 1 FlowNode
Class: ConcreteSequenceFlow
        SubClassOf:
```

SequenceFlow, sequenceFlowSourceRef exactly 2 FlowNode

A reasoner would classify "ConcreteSequenceFlow" as subclass of "owl:Nothing", because it is not satisfiable. However, this approach again does not work for a minimum cardinality (except if one defines explicitly a cardinality restriction of 0 for unset properties).

7.2.3 Summary: Verification of Models

For the purpose of verifying models against the BPMN specification the following conclusions are drawn: Verification has to be considered as a closed world problem, even if models may represent things of an open world (for example, knowledge about customers or business partners). The implication of OWA in terms of checking whether cardinality restrictions are fulfilled (i.e. structural conformance) has been shown with several examples.

The last example of staying at class level may be relevant in terms of meta-modelling aspects. However, whilst for example querying on class level is possible, one has to consider, that this restrains the usable features to reasoning about on the level of models. If we define models and their elements as individuals and for example define a property "modelHasElement", its inverse property "elementBelongsToModel" and assert the triple "process1 modelHasElement activity1", a query for "elementBelongsToModel some Thing" would return "activity1". Defining "process1" and "activity1" as classes and the relation as restriction "process1 some activity1" (or as exact cardinality), the same query provides no result. Also SWRL was not useful, as the variables in axioms relate to individuals (or data values).

Although not the complete semantics of all models find an adequate meaning in the languages of discourse, syntactic restrictions are a part of the meaning. For example, a status transition must lead to a new status - because "nothing" may be considered being not a status. Verifying models is thus a step towards ensuring, that models conform to their meaning and being executable.

Verification of Models	
Supported goals	Model execution, modelling guidance
Requirements	Closed-World assumption & Requirements from syntactic representation / modelling guidance
Supporting Languages	OWL-DL, SWRL
Requirements Fulfilment	Partial: No closed-world assumption
Application support	Low: No approach has been found to use the language features without disadvantages.

Table 13: Analysis Summary - Verification of Models

7.3 Enterprise Development

Due to the goal of EA to support change and improvement, versioning and tracking changes are often mentioned issues. Basically, this is seen as technical requirement for the tools rather than for the representation language. Even though, there are some noteworthy properties of the languages:

RDFS has a couple of properties to provide further information about a resource, such as to provide a human readable label and comment or to refer to other resources (seeAlso and isDefinedBy).

In addition to the RDFS annotations, OWL has specific properties related to versioning: First of all, there is an ontology IRI and a version IRI (by which prior versions could be accessed). Second, the ontology can be annotated by its previous version and with compatible or incompatible

versions. Further, an IRI can be annotated as deprecated to indicate, that it should not be used anymore.

These properties may give a minor advantage to OWL, but the management in terms of making versions available, comparing ontologies, displaying changes, declaring periods of validity etc. will sparsely be simplified.

Enterprise Development	
Supported Goals	Improvement and Documentation
Requirements	Annotations / Version declaration.
Supporting Languages	OWL (DL/FULL)
Requirements Fulfilment	Yes
Application support	Medium: Simple versioning and commenting.

Table 14: Analysis Summary - Enterprise Development

7.4 Model Analysis

This section discusses different types of model analysis. First, static structural analysis (e.g. dependencies), second quantitative analysis (e.g. workload) and third, dynamic analysis (e.g. simulation) is addressed.

7.4.1 Static structural analysis

In literature, dependency analysis is often remarked. In the example derived from the interview with Junisphere, it has already been shown, that relations can be discovered and used for highlighting dependent elements by using specific rules. Also according to (Lankhorst 2009) description logics are useful for structural analysis whereas an example tool for impact-of-change analysis is presented that is based on logical rules and XML transformation.

In its most rudimentary form, static analysis such as dependency or impact analysis is about querying the static structure of a model. For simple queries for related elements a simple structural representation (cp. section 7.1.1) could be a starting point. However, the Junisphere application scenario and examples from Lankhorst (2009) show that more detailed rules and propagation of relations are required.

Propagation of Dependencies with OWL

In OWL, the propagation of dependencies can be implemented very general with transitive properties and more specific with property chains. An interesting approach (in the background of automated abstraction of models) was described by van Buuren et al. (2004) where different relations are weighted, such that for example an aggregation is stronger than an association and indirect relations are propagated according to the weakest relation in the path. Based on such criteria, the propagation could be refined to have weighted dependencies (i.e. stronger or weaker impact) as depicted in Figure 24, where the "weaker" of both relations between "A" and "C" represents the relation from "A" to "C".

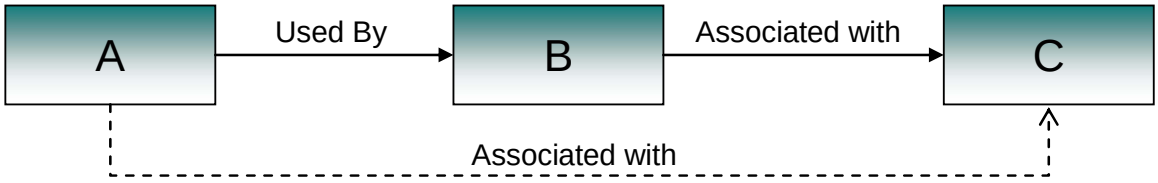

Figure 24: Impact by Weakest Relation

The construct becomes a bit complicated, if the relations (as e.g. defined in BPMN and UML) are classes. If an instance of Association connects two elements with the properties "AssociationSource" and "AssociationTarget", a propagation of dependencies (by property

chains or transitive properties) is not possible directly, because the dependency should discover elements and not relations. Thus, we could first infer a property relation between the source and the target element. This can be done by defining the inverse property "SourceOfAssociation" for "AssociationSource". This inverse still relate the element with the relation-object, thus a property is needed that relates the elements directly. We can infer this relation "AssociatedWith" using a property chain.

```
ObjectProperty: AssociatedWith
        SubPropertyChain: SourceOfAssociation o TargetOfAssociation
```

This is needed for all relation elements - e.g. a "UsedBy" relation:

```
ObjectProperty: ElementUsedBy
        SubPropertyChain: SourceOfUsedBy o UsedByTarget
```

If "UsedBy" is assumed to be stronger than the simple "Association", an indirect relation from "A" to "C" the following property chain infers the weaker relation:

```
ObjectProperty: AssociatedWith
        SubPropertyChain: ElementUsedBy o AssociatedWith.
```

This way the relation from "A" to "C" as shown in Figure 24 can be inferred. Additionally, the properties ("AssociatedWith" and "ElementUsedBy" etc.) have to be defined transitive, otherwise only the indirect relations over one intermediate element would be discovered (e.g. from "A" to "C" but not to a further element "D").

Such an approach, of course requires that corresponding property chains have to be defined for each property with respect to the other properties and also has to consider the directions. Figure 25 shows an example implemented in Protégé to test the described approach (the graph does not show the inferred relations). By connecting elements with relations weighted differently ("element3" to "element4"), an undesired characteristic of the approach was provoked: Whilst of course for "element3" a "Realises" and a "AssociatedWith" relation to "element4" is inferred, this effect spreads to "element2" which has a "ElementUsedBy" relation to "element4" (because used-by was declared as weaker than realisation) but also a "AssociatedWith" relation, because this is weaker than the association represented by "association2". If such cases occur in practice, an application had to decide on which impact relation to select: for "element3" to "element4" the stronger was probably correct, whilst for "element2" to "element4" probably the weaker would have to be chosen.

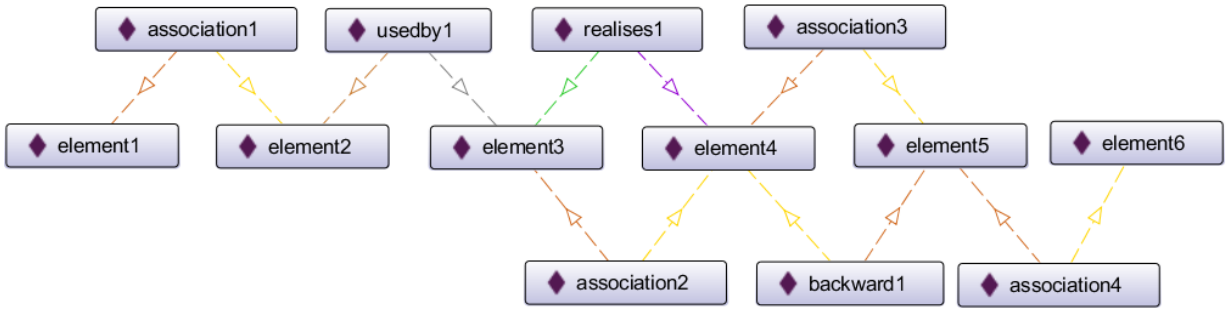

Figure 25: Ontology instance graph with relations of different weight

Limitations

Property characteristics in OWL apply to property relations between individuals. Also variables in SWRL rules refer to individuals. Thus, the presented expressions are only applicable, if the model level is restricted to individuals.

Summary: Static Structural Analysis

Despite the potential problems which were constructed in the presented example but given the capability of ontologies to represent graph structures, to define structural rules and to query the structure, the ontological representation of models for (static) structural analysis is considered as adequate. But, the limitation regarding individuals on model level is seen as critical.

RDF(S) seems to be sufficient, if combined with a rule language to state dependencies and relations more precisely between and over chains of individuals. With OWL property characteristics, some relations can be inferred, for more fine grained rules even though a rule language was useful.

Model analysis is seen mainly as support for continuous improvement in enterprise architecture. The dependencies are in particular interesting to recognise impacts of change.

Static Structural Analysis	
Supported Goals	Change and Improvement of enterprise architectures.
Requirements	Taxonomic relations, Non-taxonomic relations, Property characteristics, Meta-modelling, Relational Comparison.
Supporting Languages	RDF(S) and Rules, OWL-DL and Rules, OWL-Full, if classes are used on model level.
Requirements Fulfilment	Yes
Application support	High

Table 15: Analysis Summary - Static Structural Analysis

7.4.2 Quantitative Analysis

Although there are mathematical operators in SWRL, quantitative analysis cannot be performed by reasoning only. Within the scope of this work, no general statement can be made (e.g. whether there are some quantitative methods which could be performed). Actually it seems natural, that quantitative methods involve counting elements or sum up values of elements. A simple example was just to count the number of times a process was executed (e.g. the process instances). An approach presented in (Lankhorst 2009) makes use of the model relations to calculate workloads. For example, the arrival rate of elements is calculated based on the execution frequency of the services which depend on them. Taking an SWRL rule like the following (that tries to sum up "times" of the elements it depends on) would end in a loop, because the consequent would add the fact "hasTime", whereas this was a new fact which would lead to a new antecedent binding of "?own".

```
dependsOn(?x, ?y), hasTime(?x, ?own), hasTime(?y, ?child), add(?sum, ?own, ?child) → hasTime(?x, ?sum)
```

Without the addition, of course only the child values were added as facts without a numeric result of the intended formula.

Summary: Quantitative Analysis

This section briefly discussed quantitative model analysis. Because arithmetic operations are basically built-ins of SWRL, this is seen as major supporting language. It has been exemplified,

that quantitative analysis may aggregate figures and thus, also SWRL is not able to incorporate corresponding inferences into a model. However, the application support is even though seen as partially given, because the structural knowledge may be exploited by queries. As with all analysis techniques, the main support is perceived to be the improvement of enterprise architectures or models respectively.

Quantitative Analysis	
Supported Goals	Improvement of enterprise architecture.
Requirements	Arithmetic operations, Arithmetic aggregation
Supporting Languages	RDF(S) and Rules OWL and SWRL
Requirements Fulfilment	Partially
Application support	Medium: At least the structural knowledge can be used, even if some knowledge (i.e. based on calculations) cannot be inferred.

Table 16: Analysis Summary - Quantitative Analysis

7.4.3 Dynamic analysis

The dynamics of models cannot be represented in ontologies in terms of transitions. With programmatic control it is of course possible to simulate different states of a knowledge base (by changing facts) but a large amount of logic is then represented by software. For example, all possible states of each element have to be determined and set. Sufficient conditions in OWL or rules in SWRL can define under which condition a certain state of the ontology elements is reached. With rules, a state can also depend on the state of another element. With further programmatic effort, as shown in the interview case, even previous states can be referred to. However, because such behaviour is not part of the reasoning intended for those languages and the relation between elements in different states cannot be sufficiently represented (e.g. an individual is not logically bound to another state of itself but rather bound to another individual or state by relations or additional properties) dynamic analysis is not further considered as possible application.

Summary: Dynamic Analysis

Within the scope of this work no detailed analysis of algorithms has been performed, therefore the stated requirements (Table 17) are reduced to "dynamic constraints". However, the view on a certain state of a model given in the languages of discourse compared to the purpose of analysing dynamics (i.e. changes) seems contrasting, the requirements fulfilment is denied and the application support is interpreted as low.

Dynamic Analysis	
Supported Goals	Improvement of enterprise architecture.
Requirements	Dynamic Constraints
Supporting Languages	None (for the core application), OWL and SWRL (support)
Requirements Fulfilment	No
Application support	Low (a program, for instance an application software can possibly take advantage of the taxonomic relations and evaluate logical conditions)

Table 17: Analysis Summary - Dynamic Analysis

7.5 Compliance Management

The notion of compliance management referred to in this section is rather broad: it may refer to checking and ensuring conformance with any kind of regulation (e.g. based on standards, laws or business goals). Compliance can be checked at design time based on models, enforced at run-time or detected after the execution.

7.5.1 Representation Requirements

In (Kähmer et al. 2008; Kähmer & Gilliot 2009) it is highlighted, that different regulatory constraints may apply to one case and they therefore have to be prioritised. Prioritisation is not supported by any language: a rule either matches given conditions and leads to the corresponding consequences or it does not.

Even though Kähmer & Gilliot (2009) outline the capability of their approach to deal with incomplete context information, in this work, the task of checking or ensuring compliance is considered as rather related to a closed world assumption for the following reason: Compliance with ISO standards as dealt with in (Kim & Fox 2002) or (Schmidt et al. 2007) may for example require, that a responsible role is assigned to each activity or that each process has a certain output. Thus, if there is no assignment in a process definition, the definition is not compliant. With a close world assumption such conditions could only check for compliance, not for non-compliance: each process which has an output is compliant. The weakness is seen in the fact, that this form of ruling does not directly discover problems (e.g. the answer, which process does not comply).

The examples addressed so far imply that existential quantification, prioritisation and closed world assumption are required. In different examples from the literature review deontic as well as alethic formulations can be found. It is seen as important, that different modalities (including consequences) can be expressed.

Business rules (i.e. based on internal policies) are difficult to assess as there is basically no restriction on what enterprises may want to approach or restrict. SBVR provides several examples of business rules. On one hand, these are fictive and thus indicate limited practical relevance, on the other hand, many companies provided inputs for the specification, and hence it also reflects real world requirements. Several criteria have been shown in section 3.2.7 which are not supported by any of the languages (as SBVR actually founds on an extension of first order logic (FOL), whilst all languages considered in this work can be regarded as subsets of FOL). Some examples are related to temporal conditions - e.g. periods - others to open world reasoning, which can be defined for certain fact types in SBVR. An example referring to SBVR and the compliance issue given by (Ceravolo et al. 2008) and reflects the need to count elements and calculate percentages. Conditions which refer to numbers or amounts are regarded as highly probable if we think of inventories (e.g. safety stock) for example.

Even though several approaches use a log ontology (Ceravolo et al. 2008) or an event ontology (Pedrinaci & Domingue 2007) to represent process instances, meta-modelling capability is even though seen as valuable for compliance (by detection) because, rules or constraints that apply to the process level may also apply to the execution level, if direct inheritance is given.

7.5.2 Summary: Compliance Management

No approach has been found in literature that proves a complete or even broad coverage of practical relevant compliance issues based only on semantic technologies. However, all works described in the literature review relied on algorithms to evaluate compliance rather than depend on reasoning only. From this fact we may generally conclude, that ontology (and rule) languages are not adequate to deduce the corresponding facts.

Nonetheless, the proposed approaches indicate that semantic web languages are suitable to partially support compliance management in order to query, classify and formally describe poli-

cies and rules. Depending on the approach, either OWL DL as proposed e.g. by (Kähmer et al. 2008; Kähmer & Gilliot 2009) or a rule language as proposed in (Kim & Fox 2002; Kim et al. 2007) would be useful. I argue for a rule language, especially because deontic rules should not lead to inconsistency and thus an inferred property value may be helpful. Classification of non-compliant element (e.g. with OWL) is comparable to the example of classifying non-conform elements in section 7.2 (verification) and would have the same disadvantage: large sets of constraints in one class.

Compliance Management	
Supported Goals	Ensuring compliance (regulatory and internal).
Requirements	Prioritisation, Closed world assumption, Deontic and alethic modality, Quantification, Meta-modelling, Relational comparison.
Supporting Languages	OWL and SWRL
Requirements Fulfilment	Partial: No closed-world assumption and prioritisation.
Application support	Medium (the features of the languages of discourse allow for supporting compliance issues with limitations). No approach has been found that supports checking and enforcing compliance by design, during execution as well as a posteriori.

Table 18: Analysis Summary - Compliance Management

7.6 Model Execution

Model execution has multiple facets and dependencies. An ontology model is not executable in the way a software program is, that controls user interaction, has control flows, processes and stores data. In this section some forms of model execution are briefly recapitulated followed by a deeper look at process execution and corresponding requirements.

7.6.1 Different Forms of Execution Support

A semantic model is executable in the sense, that it may draw conclusions from user inputs. Such conclusions may include steps (i.e. tasks) to be performed next. As discussed earlier, this requires that there are individuals (the instances and data of a running process) and constraints which refer to the actual types of these individuals. For example, an activity may be chosen as adequate for fulfilling a certain goal (post-conditions) and by being executable because the pre-conditions are fulfilled (e.g. the needed data is available and corresponds to certain conditions). Several approaches related to semantic web-services (i.e. OWL-S and WSMO) have shown, that such functions can be supported by ontology languages, although, of course, software applications are needed which control such selection and composition processes. Taking into account the extensive studies on business process execution, the languages of discourse are regarded as potential mean to support the execution and composition of processes.

On the other hand, if we refer to the general application of executing models, we find the MDA approaches reflecting the generation of software code. Generating executable programs from semantic models is a totally other view. In section 7.1.3 it has been shown, that the semantics of UML, which is the most widespread approach for model driven software development, cannot

be represented completely. Basically, it is assumed, that UML models can somehow be represented (despite the semantics) as it is possible in XML and thus likewise a transformation into programming languages. However, the generation of software is not analysed in depth within this work, as in the context of logics based formalisms, it seems more interesting whether and how knowledge can be stated declaratively and support problem solving by inferences rather than using problem specific algorithms.

7.6.2 Representation Requirements for Process Execution

An intelligent process model would not just interpret a sequence of activities represented by a fix process model (e.g. that an activity has a predecessor and becomes active if that predecessor is finished and, possibly, an additional condition is fulfilled) and also not only searching for an adequate service (what of course is valuable) but rather decide on which activity has to be performed next, based on a given situation, such as in the variable process parts as described in (Feldkamp et al. 2007). In order to determine representation requirements, some approaches to support such forms of process execution are discussed hereinafter. The described approaches relate to the principle ideas presented in the literature review, basically the selection of activities based on pre- and post-conditions.

OWL Classification for Process Execution

Besides using rules, also classification based on OWL reasoning is a conceivable mean for an "intelligent" process. One way standing to reason is to define a class such as "ActivateTask" with sufficient conditions under which an activity should be performed. The first drawback is that the number of conditions to become a member of such a class would become enormous in large processes. Defining an "activation class" for each task to be executed, would blow up the ontology. Further, this approach would require that all the possible instances of a process were available. Therefore, when starting a process all process elements would have to be instantiated. This is not desirable, because it results in unused data (assuming, that not all process instances require all activities). Besides the data overhead itself, an impact on later process analysis is foreseeable, because it cannot be presumed that an activity had been performed based on the existence of a corresponding individual (i.e. an execution status had to be checked).

Alternatively, a concept comparable to tokens in BPMN (indicating the process flow or the active tasks respectively) could be imagined: A program could instantiate process steps based on the classification of a token. However, this would not free us from the necessity to define additional classes to classify the token, because, even if the process is constituted of classes, a class that defines "Task1" does not necessarily have the same sufficient conditions as the one defining that a token reaches "Task1" (meaning that an instance of the task should be created), because the former class comprises possible future instances and active or completed individuals of "Task1". In addition, a process may have more than one token at a time, e.g. during the execution of parallel activities, thus it is not clear at the beginning, how many tokens should be instantiated. Finally, such an approach would require that all conditions were based on the corresponding element (e.g. task) or elements related to it (e.g. that a document of a preceding activity is available). Within this work, it cannot be evaluated, whether this last restriction of approaches based on OWL (without rules) are of practical relevance, but given the mentioned disadvantages of defining class conditions and the instantiation problem, an approach that fully bases on OWL seems not to be advisable.

Process Execution based on Rules

Approaching the problem with rules basically faces similar problems but may relax some of them. The idea of an activation class for example would not lead to a huge union of conditions but to a set of rules and, alternatively, also a property could be set instead of using a class. However, handling rules in the latest version of Protégé (4.1 beta) for example is almost evenly confusing as the rules are represented in an ungrouped list. Also the instantiation problem remains. At least in SWRL no individuals can be created. Thus, also the idea of using a kind of

token has the same limitation as when using OWL only (the number of concurrently active tokens is not certain at process start).

The most promising solution is probably to define rules which indicate which individuals have to be created. This could for example be done by adding property values to a process instance indicating which activities have to be performed. A workflow engine could then create corresponding individuals and invoke the corresponding user interactions or services. If a process is defined using classes, the property value could refer to the URI of the corresponding activity class. If the process is described using individuals, these could be used as property values. This distinction refers to the problem of meta-modelling already discussed in section 7.1.4. This approach corresponds to the rules presented in (Feldkamp et al. 2007) where for example the rule "IF documentComplete == false DO RequestDocuments" is presented. In SWRL, this rule could be applied for a certain process "?p", a related document ?d with a completeness state "isComplete" and a concept with the (example) IRI "ex:requestDocuments" for the "RequestDocuments" activity:

Process(?p), hasDocument(?p, ?d), isComplete(?d, false)

→ invoke(?p, "ex:requestDocuments")

Possible Limitations

The decision presented in the previous paragraph bases on the given fact "documentComplete". Inferring such facts could be difficult. Apart from its correctness, a form could probably be considered as complete, if all fields have a value and as incomplete, if at least one of the fields has no value. A positive value could only be inferred by checking each field (value1 … value*n*), e.g. Document(?d), value1(?d, ?v1), …, value*n*(?d, ?v*n*). A negative value could be inferred under closed world assumption (i.e. negation, if one of the fields has no value) or if each field has a blank value (e.g. "") by default (by comparing the value with a blank string). In addition, an OR-condition was required for the negation, thus, in SWRL a rule for each value would be necessary, e.g.: Document(?d), value*n*(?d, ?v*n*), equals(?v*n*, "") → complete(?d, false). This seems quite inconvenient, as a large number of rules (and long rules for documents with many fields) were necessary. For a certain form, the expected values are known in advantage.

Taking a customer order as exemplified in (Hallmark 2008), the number of order items is not known in advance whilst the process flow may depend on the availability of the ordered items. Assuming, that the stock is known for each item, we could find whether there is an order item, that can be satisfied by the stock and if there is one that cannot. If there is an item that cannot be satisfied, we can infer that the order cannot be completely shipped. But there is no fact to conclude, that the order can be shipped completely: Due to the open-world assumption, SWRL can neither check, whether all items of an order are in stock nor that there is no item without sufficient stock. Universal quantification of OWL would face the same problem. The following statements exemplify the problem.

orderItem(?order, ?oitem), amount(?oitem, ?x), product(?oitem, ?p), stock(?p, ?y), greaterThan(?x, ?y) → itemOnStock(?oitem, false)

orderItem(?order, ?oitem), amount(?oitem, ?x), product(?oitem, ?p), stock(?p, ?y), ?x <= ?y → itemOnStock(?oitem, true)

orderItem(?order, ?oitem), itemOnStock(?oitem, false) → orderOnStock(?order, false)

"orderOnStock(?order, true)" cannot be inferred because the absence of a fact ("itemOnStock(?oitem, false)") is not sufficient assuming incomplete knowledge.

Features of the Oracle Rule Language

With the idea in mind that the offered products of large vendors of enterprise software represent functionality resulting from real world requirements, the user guide for Oracle Business Rules[27] and the white paper of Hallmark (2008) has been consulted where the following (incomplete list of) features are presented: Decision tables based on the boolean value of rules including a default ruling, if no rule applies. The definition of calculation formulas (e.g. costs of several items) including arithmetic aggregations within rules. Even the definition of non-monotonic consequents, e.g. to modify values or retract facts, is possible.

7.6.3 Summary: Model Execution

Within this work no extensive study on real-world business processes (and necessary conditions within them) has been made. Therefore, it is difficult to generalise requirements. It has been shown, that a rule language is advantageous for process control. Further it has been exemplified, that possible control conditions cannot be expressed due to the open world assumption. Considering rule implementations of Oracle, it seems not unrealistic, that negation and also (numeric) aggregation and further features are of practical importance.

With respect to the representation of executable process models using sBPEL the requirements include in particular quantification (cardinality restrictions), ordering and prioritisation (Nitzsche et al. 2007).

Finally meta-modelling is regarded as rather important requirement, because, although workarounds are possible, they all lead to disadvantages: Defining the process elements as classes restricts possible expressions, e.g. rules and property characteristics and thus analysis of models. Further it hinders individuals to point at those classes - otherwise the ontology falls into OWL full and thus, meta-modelling could be used. On the other hand, representing models as individuals would (without meta-modelling) require having separate classes to which the execution instances belong and thus would partially duplicate contents. Even though these individuals could be related to the model-individuals by any property, execution analysis was less convenient.

Whether and how far the execution support belongs to the topic of enterprise architecture is discussable. However, controlling the execution based on business models seems to be highly supportive for the alignment of business and IT. Further relating execution data with enterprise models provides a consistent relation as input for execution monitoring, analysis and compliance management.

Model Execution	
Supported Goals	Business and IT alignment (by supporting consistency from design to execution). Enabler for monitoring, analysis and compliance support (i.e. detection and enforcement).
Requirements	Prioritisation, Quantification, Meta-modelling. Probably also closed world assumption and arithmetic aggregation.
Supporting Languages	OWL-DL/Full and SWRL RDF(S) and Rules
Requirements Fulfilment	Partial: Prioritisation of rules is not provided.

[27] User's Guide for Oracle Business Rules: http://download.oracle.com/docs/cd/E17904_01/integration.1111/e10228/rules_start.htm (retrieved on 2010-02-13)

	Further requirements seem plausible (i.e. closed world negation, numeric aggregation).
Application support	Medium: Several works have shown that agile processes and service discovery can be supported. Farther rule support shall go beyond control flow and service selection (i.e. inferring facts based on which the flow is controlled) the more examples are imaginable which require more than the expressive means of the languages of discourse. This is also seen as reflected by rule systems provided by large enterprise system providers such as Oracle.

Table 19: Analysis Summary - Model Execution

7.7 Execution Monitoring and Analysis

Celino et al. (2007) refer to Business process Analysis (BPA) as "the study and evaluation of the current running processes with the aim to measure performance, find problems and solve them". According to (Pedrinaci et al. 2008) the area of BPA comprises Business Activity Monitoring (BAM), Business Intelligence, Business Process Mining and Reverse Business Engineering. This section does not fully reflect this broad topic but will point out a few examples.

The monitoring of business processes is to a certain extent dealt with in the interview case, however, the calculation of measures (i.e. performance indicators) has not been considered in the implemented test scenario.

7.7.1 Monitoring Measures

Pedrinaci et al. (2008) give several examples of measures of interest in business monitoring, for example the number of running or failed processes or (statistical) information about execution times. They also differentiate these (general) from user-defined (or domain specific) measures such as "process cost". Further, in (Pedrinaci et al. 2008) a time ontology is used to determine temporal relations between elements. To define metrics and calculate metrics, (Pedrinaci et al. 2008) use a language called OCML (Operational Conceptual Modelling Language). This reflects the limitation of ontology or rule languages to calculate aggregated values (e.g. averages or percentages) over a set of individuals. A further area is the comparison (or detection of deviations) of planned and effectively executed processes. However, this is not seen as application to be formally described but as task for comparison or algorithms, for example. (Wetzstein et al. 2008) also present a KPI ontology and a couple of example metrics (related to efficiency and effectiveness). The ontology is defined in WSML and allows, for specifying metrics with conditions (i.e. rule like expressions), aggregation operators (e.g. average or sum) and targeting values, for example. Their approach includes the real-time calculation of the metrics and the evaluation of deviation thresholds based on which events (e.g. messages to the user) can be triggered. However, the presented examples of (Wetzstein et al. 2008) raise similar requirements, in particular numeric aggregation and date comparison (i.e. start and end date). The authors point out the advantage of using implicit knowledge.

7.7.2 Process Mining

Regarding process mining, Celino et al. (2007) refer to the open source framework proM[28] which offers a broad set of mining algorithms, based on e.g. linear temporal logic or statistics to verify process models or to check model versus execution conformance. Whilst Celino et al. (2007) argue that analysis techniques may profit from semantics, no implemented example has been found in literature that would fully base on ontology or rule languages.

[28] ProM process mining toolkit: http://prom.win.tue.nl/tools/prom/

7.7.3 Summary: Execution Monitoring and Analysis

In the context of process instances meta-modelling is seen as important feature of ontology languages. However, given the fact that a lot of the actual functionality (e.g. metrics calculation) is not possible through reasoning and not directly depends on the process model (i.e. the relation to the model can be declared and queried through any user defined property), less importance is attached to meta-modelling in this area. Nevertheless some potential benefits are seen regarding the deviance from process models: if the process instances inherit restrictions and properties from the process definition, conflicting constellations could possibly be detected by reasoning functionality.

Execution monitoring is regarded as important input for architecture improvement and controlling instrument. RDF(S) provides meta-modelling and can be used for querying data in order to calculate measures.

Execution Monitoring and Analysis	
Supported Goals	Improvement of the architecture design.
Requirements	Arithmetic aggregation, Time operations, Meta-modelling.
Supporting Languages	RDF(S): facts and meta-modelling* OWL Full: facts and meta-modelling SWRL: time operations
Requirements Fulfilment	Partially: no arithmetic aggregation.
Application support	Medium: basis for querying data considering inferred facts.

Table 20: Analysis Summary - Execution Monitoring and Analysis

*None of the entailment rules of RDF(S) transfer instance values to subclasses.

7.8 Chapter Summary

This chapter analysed several applications regarding the requirements to be supported and represented using RDF(S), OWL and SWRL. The analysis has been performed with the expectation that the languages do not only provide (and infer) facts to be queried and evaluated but actually be able to deduce the answers often deduced by algorithms of tools (e.g. performance measures). Even though several relationships between the applications have been identified and indicated, their number, complexity and diversity did not allow for a complete and equally deep investigation of all topics within the scope of this work.

For all applications a summary is provided containing considerations of the best suitable representation languages, the requirements fulfilment and the potential application support. Several limitations have been identified which will be further discussed in the evaluation chapter 9.

The third research question on how the requirements can be fulfilled by the languages RDF(S), OWL and SWRL cannot be answered with a few sentences. The answer is rather provided by the considerations and example representations provided throughout the chapter. What can be clearly stated is that not all requirements identified for the analysed applications can be fulfilled by those languages, thus no combination seems adequate.

8 Junisphere Application Scenario

This chapter documents the design, implementation and evaluation of the Business Service Management (BSM) scenario based on the interview described in chapter 4. The scenario has

been approached with a design research procedure in parallel to the more general applications approached in a more analytical manner (i.e. with less implementation) in chapter 7.

8.1 Design of the Junisphere Use Case

Based on the examples and facts gathered in the interview, first a rough model has been sketched that consolidates the elements and relations which are necessary to represent a case based on the ITSM methodology. The following sections describe these components as depicted in Figure 26 and intentionally map them to potential features of the languages of discourse.

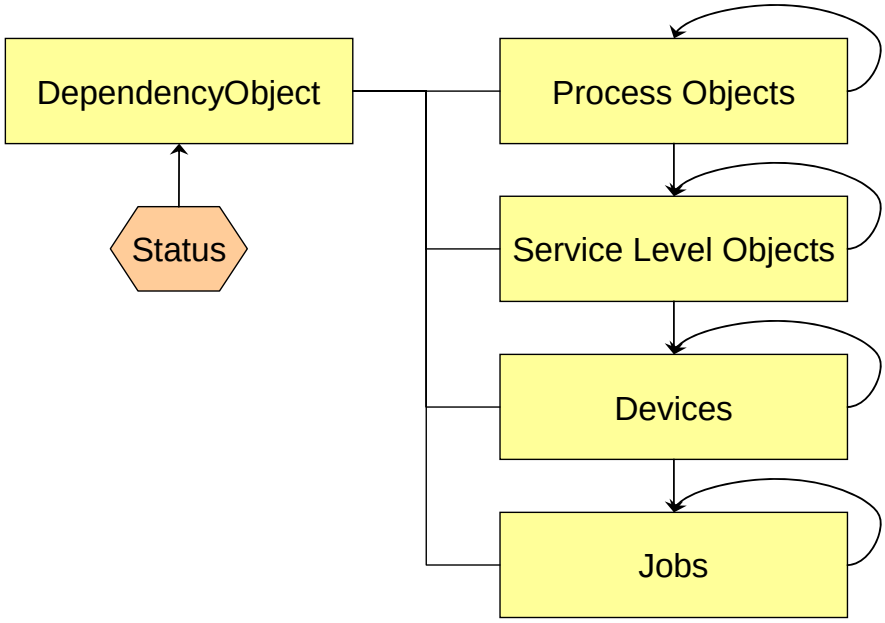

Figure 26: Rough overview of the model elements

8.1.1 Model-Objects

Four basic model object types were identified: Process objects, Service level objects, Devices and Jobs. All of them basically have similar properties: they have a status indicating a need for action and they have dependencies to other items, being more granular ones of the same kind (e.g. processes, sub processes and activities) or items from another perspective (e.g. processes depend on IT services). Figure 26 is not intended as a formal representation but should rather give an impression of the structure: The elements can be seen as classes such as "ProcessObject", "ServiceLevelObject", "Device" or "Job" with individual members such as "Order Process", "Online Shop" or "Web Server". Because all these classes have the same functional mechanism, they should share the same super class which may be called "DependencyObject", as its members are part of a dependency tree. The figure is not meant to interpret whether the relations are defined separately (e.g. "Process Object depends on Service Level Object" and "Service Level Object depends on Device") or whether "Status" is a class or property.

The "Job" concept is probably the most discussable one. The generic ITSM model as well as the interview brought up terms like "Job", "Metrics" and "KPI". A job will return a certain value (e.g. whether a server is reachable), a metric represents (normally) a figure and so does a KPI. The difference may be, whether a job just returns a value to be considered or whether there are several facts for a value to be calculated. As soon as a value has to be calculated, a rule language is needed (because there are no mathematical operations in OWL, much less in RDF). However, as described later, a rule language is necessary and advantageous anyhow.

Because a status for a certain element may be related to a hint (i.e. a workaround), the statuses are likely to be individuals rather than properties (as individuals may have properties themselves). To model workarounds seems not to be of interest for the given goals of proving the

adequateness of the languages to cover the model's expressiveness. According to the interview, such workaround hints are only comments and even if there were more properties, any information could be potentially attached to an individual in RDF as well as in OWL.

Jobs in general represent a programmatic implementation (also for business KPIs, the corresponding data has to be collected). Functionality such as sending a ping message to a network device is out of scope of semantic technologies. Therefore, the result of such a programmatic operation has to change the ontology during runtime, comparable to a user adding a new fact.

The concrete facts represented by a job is variable: If a program has to be executed to send a ping and change the knowledge base according the operation result, it is questionable, whether and to what degree the program should interpret the result. In the case of a ping, it could directly pass on the response or response time or simply whether the target device responded at all. For example, if we execute a ping command in a Windows prompt (e.g. „ping 147.86.7.23“), the command will return several lines of text, telling for example how many packages were sent and how long the response took (or if the command timed out). How the result is passed to the ontology as facts determines what the ontology can reason about it: A textual information can potentially only be evaluated using string built-ins of SWRL, for example swrlb:contains, swrlb:startsWith or swrlb:endsWith. Such a text lookup would not consider whether all or only one response timed out or what the mean response time was. The advantage was that the definition of the job class in the ontology may be very generic (i.e. having just a result-string as property). Structured information (e.g. responseTime 1, responseTime 2, responseTime 3; succeed1, succeed2, succeed3) would allow for more specific rules, using math built-ins to evaluate the mean response time or whether more than one ping failed. However, first, there was more interpretation by the respective processing job and, second, the ontology has to be very specific as it has to know about the specific types of jobs and ist properties.

A job could also be regarded as a general KPI: beside the jobs described above, also economic key figures could influence the state of elements, i.e. of processes. For example, the number of orders in a certain period indicates the status of a process. Their representation states similar questions: i.e. whether the figure is calculated by a program or whether the program simply puts the facts into the ontology.

Further helpful or necessary classes will be discovered within the design phase. As the interview showed several facets of the application, possibly some classes with respect to time or time periods, calculations (KPI) would not be unexpected.

8.1.2 Dependencies

Dependency relations can basically be represented by an object property, such as "dependentOn". Whether or not to globally restrict domain and range cannot be decided yet. This decision seems not to influence the expressivity of the language in terms of OWL DL (i.e. Protégé still declares such an ontology as "AL", which includes atomic negation, intersection, universal restriction and existential quantification with 'thing').

Child relations have a common functional meaning (children as well as other dependent objects may pass their status to the parent) and thus do not need to be distinguished. For human interpretation it may be helpful to have another label, therefore, a super-property for "depends on" and "consists of" could be defined.

8.1.3 Language considerations

RDF(S) could represent the (rather simple) class model with its class, subclass and properties but will surely not allow for formulating even simple conditions (e.g. that a status is red, if a job's return value is false).

OWL could express far more than RDF(S) and would basically allow for classifying statuses (or elements according to their status). However, OWL has limitations when it comes to calculations, date operations and restrictions that refer to properties of related elements.

Calculations could be helpful to evaluate figures returned by a job, i.e. to represent business KPIs. With OWL, we could classify an individual according to a value range (returned by a job):

```
Class: CautionStatusElement
        EquivalentTo:
        KPIJob and hasValue exactly 1 [>500]
```

But it is not possible not calculate a property value like the number of outstanding invoices relative to the number of charged sales.

The interview also showed, that some decisions relate to time, for example, that a job has to be finished until a certain time each day. This could be expressed in a comparable manner as described above for the KPI range. Of course, the value to be compared (e.g. the finishing time) must be accessible by the ontology, thus set as property value by a program.

Finally, the status of an element may be defined depending on the status of the elements it depends on. It is not possible to directly express, that for example the status of activity X is "red", if X is depending on two services which have the status "red.

It could be expressed, that an individual (X) belongs to the class of things which have the status "red", if X depends on at least two other elements which also belong to the class of things having the status "red".

```
Class: RedStatusElement
        EquivalentTo:
        DependencyElement and DependsOn min 2 RedStatusElement
```

This however was true for any element of the same type (Activity).

This is not intended, because, for example "receiveOrder" should be "red" if two services are off, "prepareGoods" already, if one is off and "deliverGoods" if a service is off for more than 4 hours. Referring to certain individuals could be done by using enumerations (ObjectOneOf) containing only one member:

```
Class: RedStatusElement
        EquivalentTo:
        ( {receiveOrder} and DependsOn min 2 RedStatusElement )
        or
        ( {prepareGoods} and DependsOn min 1 RedStatusElement )
```

The exemplary condition for "deliverGoods" cannot be expressed, because it is not possible to further restrict properties of the member connected by "DependsOn". What OWL 2 offers as a new feature are property chains. To use this feature seems not useful in the given case, because artificial properties and elements would have to be introduced, e.g. a property which is only set for "prepareGoods", if an element it depends on has a certain value (i.e. service off for more than 4 hours).

The limitations of OWL as explained above lead to the expectation, that a proper solution will additionally require a rule language. First of all, with SWRL the properties of related elements can be evaluated.

For example, if an element (x) has a status (sx) and a dependency to another element (y) which has a status (sy) that is a "CautionStatus" (meaning that the element requires attention) and has a transition time (t) of the last transition as well as the current time (now), the following rule could classify the status of x (sx) as "CautionStatus", if the time difference (diff) between "t" and "now" is greater than 2 hours (dur) - the current time aspect will be discussed later:

CurrentTime(?now), dependentOn(deliverGoods, ?y), hasStatus(deliverGoods, ?sx), hasStatus(?y, ?sy), CautionStatus(?sy), hasTransitionTime(?sy, ?t), swrlb:subtractTimes(?diff, ?now, ?t), swrl:dayTimeDuration(?dur, 0, 2, 0, 0), swrlb:greaterThan(?diff, ?dur) → CautionStatus(?sx)

A calculation of a KPI is also possible but would require, that the corresponding figures are somehow modelled. The following example assumes an order information object (OrderInfo) which is set as a return value (y) of a Job (x), which collects order information from a certain period. The rule simply calculates the ratio between the number of outstanding (o) and the total number of charged orders (c) and classifies the status of x (sx) as CautionStatus, if the ratio is above 0.5 (meaning less than 50% of the invoices of the given period have been already paid). This was possibly not a relevant key figure, but the example shows the additional possibilities of SWRL compared to OWL which seems useful for the Junisphere approach.

Job(?x), hasStatus(?x, ?sx), jobReturnValue(?x, ?y), numOutstanding(?y, ?o), numCharged(?y, ?c), swrlb:divide(?ratio, ?o, ?c), swrlb:greaterThan(?ratio, 0.5) → CautionStatus(?sx)

Using a rule language would offer another advantage apart from the expressiveness: the methodology (ITSIR) and its corresponding base model (classes and relations to be modelled) are rather static, whilst the individual elements (processes, services etc.) and the status conditions are different for each company. Anyhow, if the base model is changed or extended and the conditions are formulated in the ontology, the maintenance is potentially more difficult to manage. If the individual parts of each company are represented with OWL individuals and rules only, this can be easily distinguished from the base model. An expected problem is the "else case" - due to the open world assumption.

8.1.4 Further Design Considerations

Reasoning over an ontology is just a snapshot with respect to its state at a certain point in time. There is no mechanism to access the current time in OWL, thus, if rules refer to the current time this has to be set as property of a time instance before the reasoning process is started. As mentioned above, this is also true for jobs. If for example a program regularly checks whether a server can be reached over the network, the result has to change the corresponding ontology instances.

SWRL is Monotonic, thus, the inferred facts will not change the ontology but only add facts. Therefore, the inferred information has to be discarded each time reasoning is performed. Otherwise, contradictory or even inconsistent facts could result. For example multiple statuses could emerge for the same object what was not actually true for a given state of the ontology.

8.2 Implementation of the Junisphere Use Case

8.2.1 Test Scenario of the Junisphere Case

Because the correlation rules are specifically related to the concrete model elements, a concrete test case based on the examples and explanations of the interviewee has been defined first. The corresponding dependency model is depicted in Figure 27 and the conditions to test the element status transitions are listed afterwards. The consequent sections describe the implemented ontology and rules.

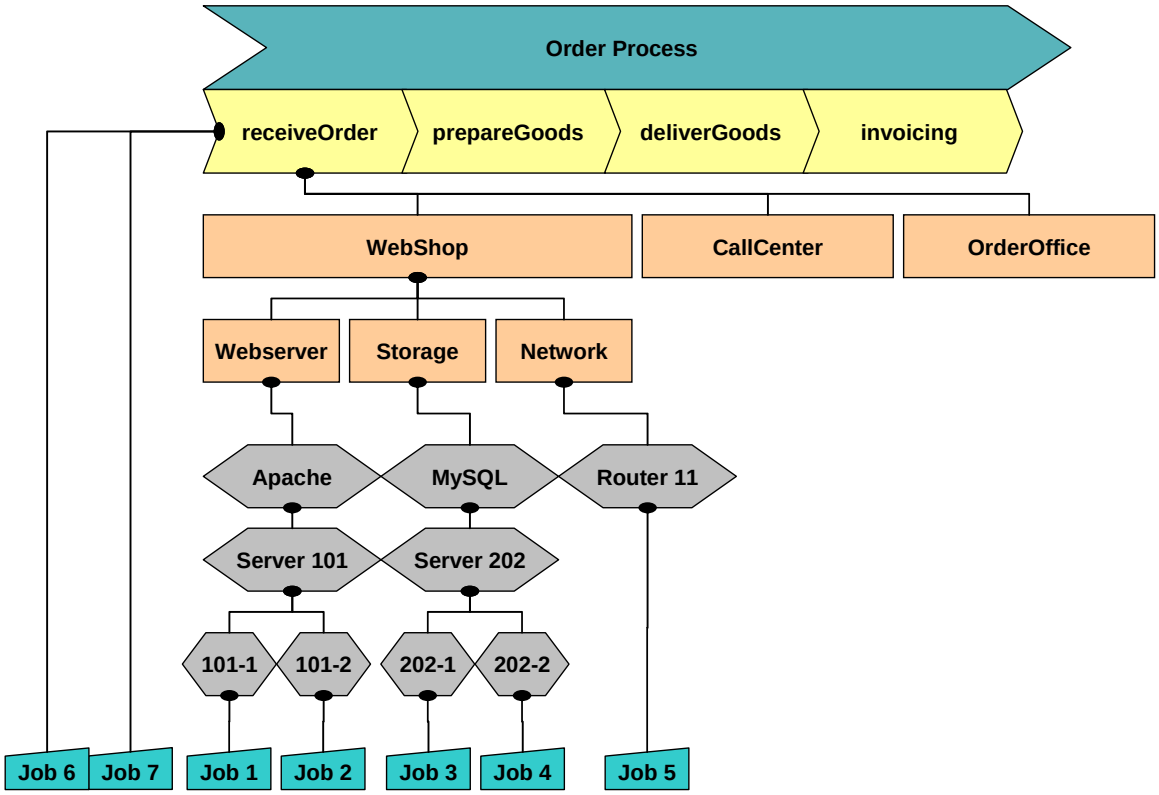

Figure 27: Test scenario model

The model and the test conditions are not complete (e.g. there are no dependencies for several elements). The purpose was to have a comprehensible example and a set of rules which represent the types of conditions mentioned by the interviewee.

The conditions are presented hereinafter in natural language. The statuses are referred to as the colours as they were symbolised during the interview. In the concrete implementation, the colour "red" will refer to "emergency" and "yellow" to "caution".

Receive Order Status Red:

- if 2 channels are off (red) or, if 1 channel is off (red) for more than 2 hours.

Receive Order Status Orange:

- if 1 channel is off or
- if Job 6 don't return a positive message until 07:00 am of the current day or
- if Job 7 has a value outside the range of 300 and 500 and it is not Christmas time or
- if Job 7 has a value outside the range of 500 and 800 and it is Christmas time

WebShop Status Red:

- if one child is off (red) for more than 1 hour or
- if Webserver and Storage is off (red)

WebShop Status Orange:

- if one child is off (red)

Server 101 Status Red:

- if all children are off (red)

Server 202 Status Red:

- if one child is off (red)

Router, D101-1, D101-2, D202-1, D202-1 Status Red:

- if the corresponding measure job (1-5) has a negative success value (false)

8.2.2 OWL Model

The core class hierarchy is rather simple and could be extended on demand (e.g. with further subclasses of "ProcessObject" like "SubProcess" or "Activity"):

- DependencyObject
    - ProcessObject
    - ServiceLevelObject
    - Device
    - Job

A member of the class "DependencyObject" may have an unrestricted number of other members of the same class connected by the property "dependentOn". Additionally, the two properties "dependencyObjectStatus " and "dependencyObjectTransferStatus" can relate an object to its status. The status set with the first property is the actual status of the object and the one set with the second property represents the status considered by the dependent element (parent). This separation reflects the fact that each element may decide on the status passed to the parent element.

Further, the test implementation contains a Status class with two subclasses:

- Status
    - CautionStatus
    - EmergencyStatus

Statuses were considered as class (or individuals respectively) because in this way, further information can be related to a status - for example a workaround or status explanation. This seems more adequate than relating information that depends on the status (or to the cause that led to the status) to the object itself.

The Status "Green" is not represented but seen as "default" which is always true for a status which is not classified as "CautionStatus" or "EmergencyStatus".

Both subclasses of Status have a value restriction on the data property "statusValue":

```
Class: EmergencyStatus
        SubClassOf: statusValue value 'Emergency'
        and
        CautionStatus SubClassOf "statusValue value 'Caution'
```

If a status is classified as member of either one (CautionStatus or EmergencyStatus), the corresponding property value is infered. Thus, a program would not have to query the class membership but could directly access the string value of the property which could for example be used as name of a style class of a graphical implementation. Thus, there is no condition such as *"if class = <class> then represent the element in <colour>"* or *"if value = 'Caution' then represent the element in 'yellow'"* needed. The class membership could be omitted by only using the property string. Classes have the advantage that their members are directly visible in ontology editors, thus this construction, is at least helpful for test and development.

Finally, there is a class "TimeObject" with two subclasses:

- TimeObject
    - ChristmasTime

- UnnamedPeriod

The TimeObject class has one member in the test implementation, which is "CurrentDateTime" and is used for time comparisons. The value of this individual has to be set by a program (or manually) before the reasoning starts.

"ChristmasTime" is an example period, because the threshold of rules could depend on temporal facts. In the test model, it is defined as:

```
Class: ChristmasTime
    Equivalent to:
    TimeObject and (timerDay some integer[>= 15]) and (timerMonth some integer[>= 12])
```

"UnnamedPeriod" was the actual complement. To use OWL's "ObjectComplementOf" a time object would have to be classified as member of a disjoint class because of the open world assumption. Therefore, the class is defined as:

```
Class: UnnamedPeriod
    Equivalent to:
    TimeObject and ((timerDay some integer[< 15]) or (timerMonth some integer[< 12]))
```

Howver, the definition of Periods (and especially of its complement) could be very cumbersome. For the test example a simple period had been chosen by taking the second half of the last month in the year.

Dealing with date and time turned out to be one of the main difficulties: on one hand, OWL offers only property value restrictions on xsd DateTime values. SWRL has some built-ins, but durations include either the year (Duration and YearMonthDuration) or the Day (DayTimeDuration) but not month and day. Thus periods within every year cannot be described. Further on, the reasoning support for date and time operations is limited. Pellet[29] for example only supports five built-ins, FaCT++[30] and HermiT[31] none at all.

Some workarounds have been implemented to test even rules which were actually expressible with SWRL.

To enable the period definition above, a couple of rules were applied to determine the day and year of the date of the "CurrentDateTime" instance. When it comes to weekdays, the information has to be provided too (i.e. programmatic maintenance of the ABox).

Figure 28 visualises the dependency tree created for the example model. Because the rules for the given application are not general but depending on the enterprise to be modelled, the rules in general refer to specific individuals.

[29] Pellet Reasoner: http://clarkparsia.com/pellet/

[30] FaCT++ Description Logic reasoner: http://code.google.com/p/factplusplus/

[31] Hermit OWL Reasoner: http://hermit-reasoner.com/

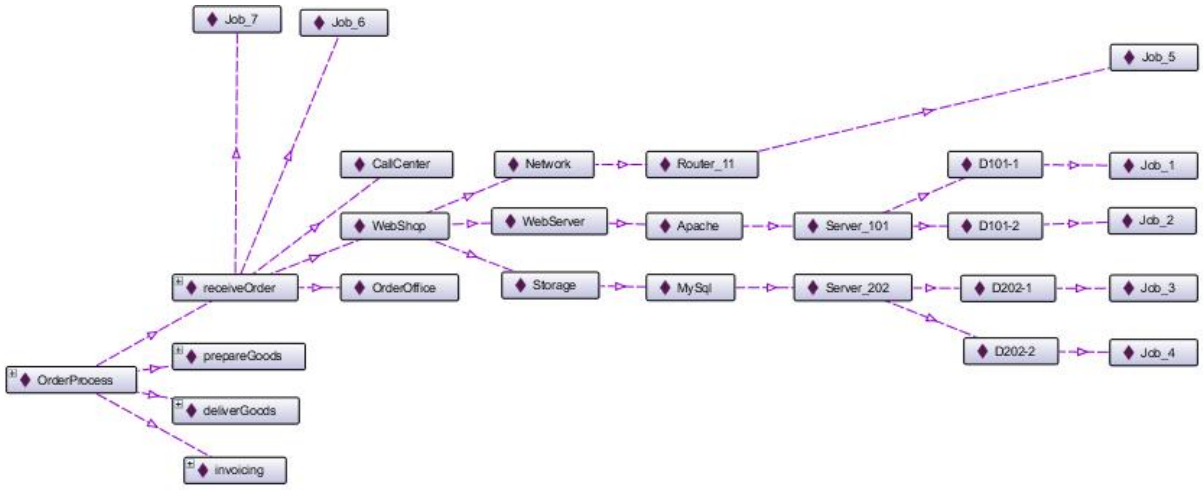

Figure 28: Dependency tree individuals visualised in Protégé

8.2.3 Rules

The rules are explained hereinafter and represented in the Manchester Syntax supported by Protégé.

receiveOrder

Testing whether a status is set for more than a certain number of hours was planned to implement by comparing dayTimeDurations. In contrast to the other functions (e.g. add, subtract) Pellet did not set unbound variable values such as "swrlb:dayTimeDuration(?unbound,0,2,0,0)". In addition, the comparison would require subtracting the duration from a date, what is not supported.

Thus, for test purposes, the difference was compared based on time stamps, where 7200000 milliseconds represent 2 hours.

```
EmergencyStatus(?statusx), dependencyObjectTransferStatus(?x, ?statusx),
dependentOn(receiveOrder, ?x), objectStatusTimeStamp(?statusx, ?stime),
timerTimeStamp(CurrentDateTime, ?now), swrlb:greaterThanOrEqual(?diff, 7200000),
swrlb:subtract(?diff, ?now, ?stime) → EmergencyStatus(receiveOrderStatus)
```

This rule checks, whether two subordinated elements have the status "Red":

```
EmergencyStatus(?statusx), EmergencyStatus(?statusy), dependencyObjectTransferStatus(?x,
?statusx), dependencyObjectTransferStatus(?y, ?statusy), dependentOn(receiveOrder, ?x),
dependentOn(receiveOrder, ?y), DifferentFrom (?x, ?y) → EmergencyStatus(receiveOrderStatus)
```

All jobs on which the process depends lead to a "Yellow" status if they have a "Yellow" status too:

```
CautionStatus(?jobstatus), Job(?job), dependencyObjectTransferStatus(?job, ?jobstatus),
dependentOn(receiveOrder, ?job) → CautionStatus(receiveOrderStatus)
```

WebShop

This rule again compares two timestamps to evaluate, whether the status "Red" of an object on which the "WebShop" depends is set since an hour or more (>= 3600000 milliseconds).

```
EmergencyStatus(?statusx), dependencyObjectStatus(WebShop, ?webshopstatus),
dependencyObjectTransferStatus(?x, ?statusx), dependentOn(WebShop, ?x),
objectStatusTimeStamp(?statusx, ?stime), timerTimeStamp(CurrentDateTime, ?now),
```

swrlb:greaterThanOrEqual(?diff, 3600000), swrlb:subtract(?diff, ?now, ?stime) → EmergencyStatus(?webshopstatus)

The caution status is set, if one element on which the "WebShop" depends has status "Red":

EmergencyStatus(?stat), dependencyObjectTransferStatus(?child, ?stat), dependentOn(WebShop, ?child) → CautionStatus(WebShopStatus)

If the Storage and WebServer objects have the status "Red", the WebShop also gets status "Red":

EmergencyStatus(?storagestatus), EmergencyStatus(?wsstatus), dependencyObjectStatus(Storage, ?storagestatus), dependencyObjectStatus(WebServer, ?wsstatus) → EmergencyStatus(WebShopStatus)

Intermediate Elements

The Service Level Elements **Webserver, Storage and Network** as well as the **Apache- and MySql Server** take the status of the element they depend on. The rules are stated differently: the former three directly refer to specific statuses (which are named individuals) and the latter apply variables to check for dependency objects. Thus, the latter would also set the status, if any additional child would be connected and had a "Red" status.

EmergencyStatus(ApacheStatus) → EmergencyStatus(WebServerStatus)

EmergencyStatus(MySqlStatus) → EmergencyStatus(StorageStatus)

EmergencyStatus(Router_11Status) → EmergencyStatus(NetworkStatus)

EmergencyStatus(?status), dependencyObjectTransferStatus(?x, ?status), dependentOn(MySql, ?x) → EmergencyStatus(MySqlStatus)

EmergencyStatus(?status), dependencyObjectTransferStatus(?x, ?status), dependentOn(Apache, ?x) → EmergencyStatus(ApacheStatus)

Server 101 and 202

The following rules simply transfer the status to the dependent objects. The "Server_202" emergency status was intended to set if ALL elements it depends on are "Red". This can not be formulated. Therefore, both statuses of the subordinated elements are explicitly stated:

EmergencyStatus(D101-1Status), EmergencyStatus(D101-2Status) → EmergencyStatus(Server_101Status)

EmergencyStatus(?status), dependencyObjectTransferStatus(?y, ?status), dependentOn(Server_202, ?y) → EmergencyStatus(Server_202Status)

Router, 101-1, 101-2, 202-1, 202-2

The next rules also simply transfer the "EmergencyStatus" from the Jobs to the dependent elements. The last rule again would also work for any additional dependency element.

EmergencyStatus(?status), dependencyObjectTransferStatus(Job_2, ?status) → EmergencyStatus(D101-2Status)

EmergencyStatus(?status), dependencyObjectTransferStatus(Job_3, ?status) → EmergencyStatus(D202-1Status)

EmergencyStatus(?status), dependencyObjectTransferStatus(Job_1, ?status) → EmergencyStatus(D101-1Status)

EmergencyStatus(?status), dependencyObjectTransferStatus(Job_4, ?status) → EmergencyStatus(D202-2Status)

EmergencyStatus(?status), dependencyObjectTransferStatus(?x, ?status), dependentOn(Router_11,

?x) → EmergencyStatus(Router_11Status)

Jobs

A general rule sets the emergency status, if a job has a negative "pingJobSuccess".

dependencyObjectStatus(?job, ?status), pingJobSuccess(?job, false) → EmergencyStatus(?status)

All jobs except number 6 get an emergency status if they have a negative Boolean return value. As the jobs are defined as different individuals, "Job_6" can be excluded from the rule (DifferentFrom). "Job_7" was not excluded as a numeric return value is expected.

dependencyObjectStatus(?x, ?status), jobReturnValueBoolean(?x, false), DifferentFrom (?x, Job_6) → EmergencyStatus(?status)

Consequently, a further rule sets the "CautionStatus" for "Job_6":

dependencyObjectStatus(Job_6, ?status), jobReturnValue(Job_6, false) → CautionStatus(?status)

The more interesting and (for reasons of functionality) challenging rules are the following.

Job 6 has the status "Yellow", if the job return value is false. But also, if there is no return value until 7 a.m. of the current day:

dependencyObjectStatus(Job_6, ?status), jobDateValue(Job_6, ?exec), timerDateValue(CurrentDateTime, ?d), timerHour(CurrentDateTime, ?hour), swrlb:greaterThanOrEqual(?hour, 7), swrlb:lessThan(?exec, ?d) → CautionStatus(?status)

If it is not Christmas time, Job 7 should return a figure in the range of 300 to 500:

UnnamedPeriod(CurrentDateTime), dependencyObjectStatus(Job_7, ?status), jobReturnValueNumeric(Job_7, ?value), swrlb:greaterThan(?value, 500) → CautionStatus(?status)

UnnamedPeriod(CurrentDateTime), dependencyObjectStatus(Job_7, ?status), jobReturnValueNumeric(Job_7, ?value), swrlb:lessThan(?value, 300) → CautionStatus(?status)

During Christmas time, the expected value should be in the range of 500 to 800:

ChristmasTime(CurrentDateTime), dependencyObjectStatus(Job_7, ?status), jobReturnValueNumeric(Job_7, ?value), swrlb:lessThan(?value, 500) → CautionStatus(?status)

ChristmasTime(CurrentDateTime), dependencyObjectStatus(Job_7, ?status), jobReturnValueNumeric(Job_7, ?value), swrlb:greaterThan(?value, 800) → CautionStatus(?status)

Status and Time

The separation between the object's own status and the one to be considered by the dependent object is handled with two general exemplary rules. The test case shows, that two statuses can be handled. The rules could however be specialised similar to the dependency rules.

All members of "Job" get the object's status as "transfer status":

Job(?x) dependencyObjectStatus(?x, ?y) → dependencyObjectTransferStatus(?x, ?y)

The alternative rule sets the values of the object's status for a separate transfer status individual:

dependencyObjectStatus(?x, ?y), dependencyObjectTransferStatus(?x, ?z), objectStatusTime(?y, ?time), objectStatusTimeStamp(?y, ?stamp) → objectStatusTime(?z, ?time), objectStatusTimeStamp(?z, ?stamp)

The classification then also has to be done in accordance to the object's own classification:

CautionStatus(?s), dependencyObjectStatus(?x, ?s), dependencyObjectTransferStatus(?x, ?ts) → CautionStatus(?ts)

EmergencyStatus(?s), dependencyObjectStatus(?x, ?s), dependencyObjectTransferStatus(?x, ?ts) → EmergencyStatus(?ts)

Some of the rules refer to how long a certain status is active. The status itself is inferred knowledge and has to be discarded (to avoid adding multiple or even contradictory facts). Thus, the knowledge, when a status changed will be lost too. A programmatic workaround could work as follows: After the reasoning process, a program could add assertions such as "objectRecentStatus", and "objectRecentStatusTime" based on the inferred knowledge. When the model changes and the reasoning is performed again, the following rules compare the new and the "recent" information and set the status time accordingly.

If the "recent" equals the "new" status, the time will be set from the "objectRecentStatusTime".

objectRecentStatus(?x, ?rstat), objectRecentStatusTime(?x, ?rtime), objectRecentStatusTimeStamp(?x, ?rstamp), statusValue(?x, ?val), swrlb:equal(?rstat, ?val) → objectStatusTime(?x, ?rtime), objectStatusTimeStamp(?x, ?rstamp)

If the "recent" status is different from the "new" status, the current time will be set.

objectRecentStatus(?x, ?rstat), statusValue(?x, ?val), timerDateTime(CurrentDateTime, ?now), timerTimeStamp(CurrentDateTime, ?stampnow), swrlb:notEqual(?rstat, ?val) → objectStatusTime(?x, ?now), objectStatusTimeStamp(?x, ?stampnow)

Obviously the classification rule of a status may not depend on its own status time (but on the recent time).

The following rules perform some date operations and calculations supported by the Pellet Reasoner - not all of the inferred values of each are used.

With "swrlb:time", the components of the "CurrentDateTime" can be accessed. The "timerHour" property is then used to check whether a job returned a value until a certain time.

timerDateTime(CurrentDateTime, ?dt), swrlb:time(?dt, ?h, ?m, ?s, ?z) → timerHour(CurrentDateTime, ?h), timerMinute(CurrentDateTime, ?m), timerSecond(CurrentDateTime, ?s)

The same can be done for year, month and day:

TimeObject(?x), timerDateTime(?x, ?dt), swrlb:date(?dt, ?y, ?m, ?d, ?z) → timerDay(?x, ?d), timerMonth(?x, ?m), timerYear(?x, ?y)

The following two rules generate a numeric date value (e.g. "2010-11-01" as 20101101). This is a workaround to work with the Pellet reasoner, which does not support "swrlb:subtractDates" for example. The use of this approach is very limited: it allows for comparing dates on a day basis with a single number.

TimeObject(?x), timerDate(?x, ?dt), swrlb:add(?dv, ?yv, ?mv, ?d), swrlb:date(?dt, ?y, ?m, ?d, ?z), swrlb:multiply(?mv, ?m, 100), swrlb:multiply(?yv, ?y, 10000) → timerDateValue(?x, ?dv)

Job(?x), jobLatestExecution(?x, ?dt), swrlb:add(?dv, ?yv, ?mv, ?d), swrlb:date(?dt, ?y, ?m, ?d, ?z), swrlb:multiply(?mv, ?m, 100), swrlb:multiply(?yv, ?y, 10000) → jobDateValue(?x, ?dv)

The suspicion that such a calculated value is not sufficient turned out to be true. Thus, also a timestamp (i.e. milliseconds) was used in addition to the dateTime properties.

8.3 Junisphere Use Case Implementation Tests

A concrete test scenario has been described in the previous section in terms of dependency rules. To actually test and demonstrate the functionality, a couple of test cases (constellations according the expected rule execution) have been defined and a simple user interface has been implemented. The following sections briefly present the program and test results.

8.3.1 Test GUI

The user interface is depicted in Figure 29: it allows for setting time- and job return values (on the right side). The boolean return values are checkboxes, whereas checked means success. Job 7 is expected to return a certain figure. The first date field allows for "simulating" the "current time" (of reasoning) and the other time fields indicate when a job returned its latest value. The program also implements the ontology maintenance operations assumed in the design (i.e. set "recent values" and reset inferred knowledge). After performing the reasoning, the resulting statuses are displayed.

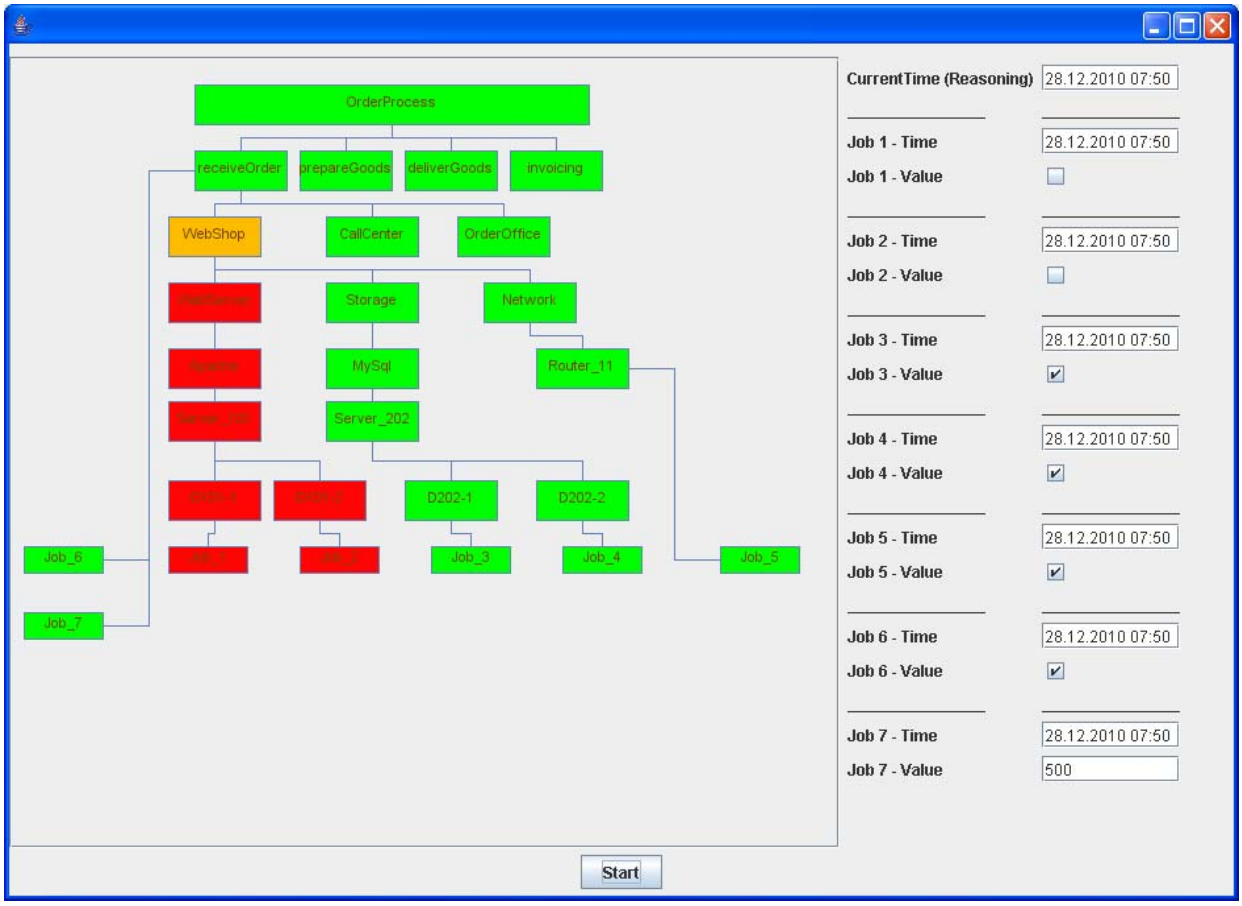

Figure 29: Test GUI - Junisphere Use Case

8.3.2 Test Cases

The following concrete tests have been performed. The values were set according to "Description" and all other values were set "positive", so that the remaining Jobs were "Green":

Description	Expected result	Success
negative value for Job 1	Job 1: Red, D101-1: Red, **Server 101: Green**	Y
negative value for Job 2	Job 2: Red, D101-2: Red, **Server 101: Green**	Y
negative value for Job 1 & 2	Job 1: Red, D101-1 & D101-2: Red, Job 2: Red, Server 101: Red, Apache: Red, **Webshop: Yellow**	Y
negative value for Job 3	Job 3: Red, D202-1: Red, Server	Y

	202: Red, MySql: Red, Storage: Red, **WebShop: Yellow**	
negative value for Job 3 - repeat with time +1.5h	Job 3: Red, D202-1: Red, Server 202: Red, MySql: Red, Storage: Red, **WebShop: Red**	Y
negative value for Job 4	Job 4: Red, D202-1: Red, Server 202: Red, MySql: Red, Storage: Red, **WebShop: Yellow**	Y
negative value for Job 4, repeated with current time +1.5h	Job 4: Red, D202-1: Red, Server 202: Red, MySql: Red, Storage: Red, **WebShop: Red**	Y
negative value for Job 4, repeated with current time +2.5h	Job 4: Red, D202-1: Red, Server 202: Red, MySql: Red, Storage: Red, WebShop: Red **receiveOrder: Red**	Y
negative value for Job 3 & 4	Job 4: Red, Job 4: Red, D202-1: Red, Server 202: Red, MySql: Red, Storage: Red, **WebShop: Yellow**	Y
negative value for Job 5	Job_5: Red, Router 11: Red, **Network: Red**	Y
negative value for Job 1, 2 & 3	Job 3, 2 & 3: Red, D101-1, D101-2 & D202-1: Red, Server 101 & Server 202: Red, Apache: Red, MySql: Red, Webserver, Storage: Red, **WebShop: Red**	Y
negative value for Job 6	Job 6: Yellow, **receiveOrder: Yellow**	Y
positive value for Job 6 & repeated reasoning with Current Time before 7:00 am and Job 6 Time previous day.	Job 6: Green, **receiveOrder: Green**	Y
positive value for Job 6 repeated with Current Time after 7:00 am and Job 6 Time previous day.	Job 6: Yellow, **receiveOrder: Yellow**	Y
value "500" for Job 7, current date outside of December 15-31:	Job 7: Green, **receiveOrder: Green**	Y
value "200" for Job 7, current date outside of December 15-31:	Job 7: Yellow, **receiveOrder: Yellow**	Y
value "600" for Job 7, current date outside of December 15-31:	Job 7: Yellow, **receiveOrder: Yellow**	Y
value "600" for Job 7, current date between December 15 and 31:	Job 7: Green, **receiveOrder: Green**	Y
value "450" for Job 7, current date between December 15 and 31:	Job 7: Yellow, **receiveOrder: Yellow**	Y
value "900" for Job 7, current date between December 15 and 31:	Job 7: Yellow, **receiveOrder: Yellow**	Y

Table 21: Test Results of the Scenario Implementation

A problem which is not shown by the test cases: the combination of a yellow status of Job 6 or Job 7 with constellations that lead to a red status of receiveOrder leads to multiple statuses for receiveOrder.

8.4 Chapter Summary

This chapter has presented an example scenario based on the interview conducted with Junisphere. The scenario has been developed, implemented and tested. Some limitations of the representation languages have been faced but partially bypassed by programmatic control. In particular these were the dynamic constraints and missing prioritisation (thus indirectly closed world negation).

The answer to the third research question (how the application requirements can be fulfilled by the selected semantic languages) is, as in chapter 7 not directly answerable but provided through the developed constructs. Because there were also limitations, it can be stated, that none of the languages is adequate. But the scenario could be broadly covered by OWL-DL and SWRL.

9 Evaluation of the Representation Languages

This chapter provides the final results of testing the developed concepts. The following sections describe the test cases and the corresponding results. The chapter finalizes the design research by summarising the findings regarding to the fourth research question.

> Research Question 4:
>
> What are the shortcomings and strengths of either one representation (or combination)?

Most of the applications investigated required features which were not fully covered by the languages of discourse. The development of the Junisphere scenario resulted in a comparable finding. On the other hand, this does not mean that the applications could not benefit from representations in ontologies. Further, it has to be questioned, what the alternatives exist and how important those features are. These questions are out of scope for this work, even though, some languages were briefly described throughout the work at hand.

Language(s)	**Evaluation**
RDF(S)	Only the structural representation of models was possible. Some limited reasoning can be exploited by querying the model. Given the analysed applications, most of the semantics was not incorporated in the model but in the queries and the programs processing and evaluating the queries.
RDF(S) and Rules	Model analysis, execution and monitoring/execution analysis was partially possible. The possibilities of annotating models would reduce the version/life-cycle management support. Due to missing quantification, the support of syntactic representation, verification and compliance management was further limited as compared to OWL. On the other hand, the meta-modelling facilities allow for a more adequate representation of different levels of abstraction (e.g. models representing classes and/or individuals) and layers (e.g. model and execution data).
OWL-DL	In addition to the structural representation, OWL-DL was capable of representing many important syntactic and semantic features of models and provides built-in support for versioning annotations. Due to open-world semantics the validation of the syntactic part are limited. Even though, it is seen as better alternative to use the built-

	in features (i.e. property restrictions) to represent syntactic rules than to use classes with properties representing them, because defining an object (or relational) model to represent restrictions can also be done in any format (e.g. database or XML) but has no corresponding semantics. Even if an application still had to do some checks by querying.
OWL-DL and SWRL	SWRL enhances the possibilities of OWL in particular by allowing for variable based conditions (e.g. distinct indirect relations), built-ins for calculation and comparisons and setting property values. These features complement the capabilities of syntactic representation, (model and execution) analysis and compliance. Further they reduce the need of artificial constructs, e.g. for compliance- and execution-support.
OWL-Full	Regarding the identified requirements, OWL-Full only adds the meta-modelling facilities from RDF(S). This becomes important for execution and execution analysis (and thus in a posteriori compliance checks).
OWL-Full and Rules	Consequently, in combination with a rule language, OWL-Full would fulfil the largest set of requirements and thus provide the highest level of application support. Unfortunately, at the price of decidability. From a practical point of view even worse theoretical decidability was of questionable importance (if a strategy to handle respective problems that occur could be found). But actually, the support by reasoners is restricted (during this research, I found no provider claiming to have a reasoner supporting OWL-Full).

Table 22: Evaluation Table of Language Combinations

The answer to the fourth research question (shortcomings and strengths of either one representation or combination) is represented by the comparison in Table 22. Given these results, it seems impossible to make a distinct choice (i.e. proposition for either one language or combination). The language combination with the highest application support lacks of reliable reasoning (i.e. implementations) and seems therefore not to be an advisable choice. The OWL-DL based combinations offer useful additions to RDF(S) but still not cover all requirements while having a disadvantage regarding abstraction-/representation-levels. RDF(S) alone leaves most logic decisions outside the model, but with a rule language it provides at least partial support for several applications.

Actually, this reflects a decision between variants whereof none is fully satisfactory. For a modelling tool, OWL-DL with SWRL seems to be the best choice because the level of syntactic and semantic representation that also partially allows for checking model constraints and compliance (by design).

To support the actual execution and execution analysis, RDF(S) seems to be the better choice, because meta-modelling allows for a consistent representation from models to execution data and still allows for some analysis.

10 Conclusion and Contribution

This section provides a short recapitulation of the results presented in the document at hand. Based on the findings and results of the work, the main research question is answered. Further on, the possible contribution and consequences with respect to ontology based enterprise modelling is discussed.

10.1 Recapitulation and Reflection

Chapter 5 identified relevant applications and modelling languages for enterprise architecture models. Chapter 6 presented the mapping between applications and representation requirements which were developed iteratively during the project. Further, the representation requirements were mapped to the languages. Chapter 7 discussed the applications (from chapter 5) in more depth, analysed necessary and possible representations, derived representation requirements and appraised the best suitable languages and limitations. In chapter 8 a concrete scenario has been developed and implemented. This lead to similar results as chapter in a similar conclusion as 7: not all requirements can be fulfilled by the selected representation languages. 9 provided a brief evaluation of the representation languages and combinations by comparing the strengths and shortcomings of either one.

A lot of research publications can be found which deal with the support of applications related to enterprise architecture using semantic technologies (i.e. ontologies and rule languages). It seems that the motivation behind most of these works was to demonstrate the power and advantages of ontologies rather than highlighting limitations or objective requirements. Whilst some publications mention or describe some limitations or missing features, others rather scope their work to parts of an application that potentially can be covered.

The work at hand analysed several applications and approaches in a rather critical manner. Although the work predominantly based on examples from literature, a concrete application has been examined on the basis of an interview that showed several similarities to the other applications. The work illustrated that a few but basic characteristics of the languages of discourse hinders them to completely fulfil the application requirements.

The open-world assumption adopted by the semantic web languages has been identified as one of the most limiting characteristics. Whilst even quantified restrictions and conditions can be formulated, their adherence cannot always be inferred when assuming incomplete knowledge. An open world assumption seems plausible in many cases. When we search for a document or a company in the internet, we would probably not infer, that no document (or company) exists which corresponds to the search terms, if the search engine provides no matching result. In business, information about third parties (e.g. other companies or customers) could also be considered as incomplete knowledge. But the data which is maintained (e.g. in databases) is normally considered as complete, even if it may be erroneous. Enterprise models may be incomplete in that not everything in a company is modelled and not every property of the real world is represented. But it is of interest, whether the part which is modelled corresponds to what is intended. If a model object has no name, I would infer that it has not been labeled, not that I just don't know the name. All representation languages considered in this work have a semantic web background, thus the languages are probably more adequate for applications related to the internet.

Another limitation is related to quantitative measures. Business key figures and performance metrics often base on aggregated figures such as e.g. the average processing time, relative number of returns etc. Whilst SWRL defines arithmetic built-ins, aggregation of individual properties is not possible.

A further issue is meta-modelling. The analysis resulted in the belief that several of the considered applications would at least profit from meta-modelling and that, even without reaching the execution level of models, the combination of classes and individuals at model level could be

justified. The supporting languages, however, lack either of expressive power or of reasoning support.

10.2 Main Research Question and Thesis Statement

The thesis statement assumed that one of the representation languages would adequately support the model representation to adequately support the major goals and applications of enterprise architecture.

> Thesis Statement:
>
> The expressiveness of one or a combination of RDF(S), OWL dialects and SWRL is adequate to formally represent the most relevant models used for enterprise architecture modelling up to the level that is required for the major application and purpose of enterprise architecture and the models themselves considering the necessary features of the corresponding language specifications.

The corresponding main research question asked for the adequate formalism out of combinations of RDF(S), OWL and SWRL.

> Main Research Question:
>
> Which language or combination out of RDF(S), OWL and SWRL is adequate to represent the relevant aspects of the most relevant enterprise architecture models?

Based on the results at hand and the comparison of chapter 9, the answer to the main research question is that none of the combinations is adequate, because any language or combination has disadvantages and none covers all identified representation requirements.

Consequently, the thesis statement has basically to be negated: The expressiveness of the languages is not adequate to represent the most relevant models of enterprise architecture up to the level which is required for the major applications and purposes.

10.3 Interpretation of the Results

So far, the results have been directly applied to the research questions and the corresponding conclusion has been drawn in terms of negating the thesis statement. When interpreting the answers and results, several limitations have to be considered which are stated hereinafter.

The majour part of this work bases on literature and several examples are fictive. Even the application scenario, although it bases on the exemplification of an experienced interviewee, does neither cover the whole software application (eRanger) nor does it reflect a real world implementation scenario.

Due to the number and complexity of the selected applications and languages it was not possible to analyse and develop an overall solution to reach a higher degree of certainty regarding the results.

Further, several representation mechanisms have been analysed by my best knowledge and belief, but there is no proof, that, what I regarded as limitations, can really not be solved in certain cases (with the considered representation languages).

Consequently, it is sure not unlikely that further requirements exist and possible, that some requirements were not relevant in some practical cases.

Finally, I did not weight the applications or application requirements, because the goal was to select the most important ones. The consequence from this fact is also twofold: When a representation language is selected for a project or a tool implementation for instance, there is more clearance about which applications have to be supported to what extent. Thus, there may be additional or less requirements (what could even lead to another conclusion).

10.4 Contributions

The literature review but also the development chapter (7) provides a broad and partially detailed view on goals, applications and existing efforts using semantic technologies all in the context of enterprise modelling.

This work has plausibly shown some requirements for enterprise models and limitations of representation languages. Even if one may question the importance of the selected applications this work also founds a basis for selecting representation languages given a certain case and specific requirements.

In particular, I consider the results presented in 6 as valuable input for futher language evaluations: The mapping table of section 6.4 can be used to select representation requirements and the mapping table 6.3 allows for selecting a suitable language (if there is one). Additional representation languages and newly identified representation requirements can be added, of course.

10.5 Recommendations and Future Work

Due to the fact that the thesis statement has been negated, the investigation of other representation languages stands to reason.

As possible languages, the WSML family could be considered but to my best knowledge, this would provide decidable meta-modelling support, a rule based language profile and compatibility with OWL, but no closed world assumption. Further, Frame Logic (F-Logic) or Flora-2[32], a language based on F-Logic with extensions could be investigated. Possibly also the logical foundations of commercial offers may give further insights.

Possibly also an in depth study about available reasoners and eventually existing and supported language extensions or combinations could lead to new insights.

To make further use the output of this work, the mapping tables of chapter 6 could be used as spreadsheets and supplemented with weights (e.g. in the requirements mapping, Table 7) and with a function to select the most suitable representation language (in the language support mapping, Table 6). As mentioned above, this has not been done within this work, because there were no priorities between applications and rather a solution with complete coverage of application requirements was searched for. Thus, a qualitative comparison has been chosen. However, the tables could be helpful for the selection of formal languages for any application, especially, if further representation languages would be added.

[32] FLORA-2: An Object-Oriented Knowledge Base Language: http://flora.sourceforge.net/

11 Bibliography

Anon, 2009. OWL 2 Web Ontology Language: Document Overview. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/owl2-overview/ [Accessed November 1, 2010].

Antoniou, G., Franconi, E. & van Harmelen, F., 2005. Introduction to Semantic Web Ontology Languages. In N. Eisinger & J. Maluszynski, eds. *Reasoning Web, Proceedings of the Summer School, Malta, 2005*. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag.

Antoniou, G. & van Harmelen, F., 2003. Web Ontology language: OWL. In S. Staab & R. Studer, eds. *Handbook on Ontologies in Information Systems*. Springer-Verlag, pp. 67-92. Available at: http://portal.acm.org/citation.cfm?doid=1295289.1295290.

Baader, F., Horrocks, I. & Sattler, U., 2009. Description Logics. In S. Staab & R. Studer, eds. *Handbook on Ontologies*. Berlin Heidelberg: Springer-Verlag, pp. 21-43.

Bao, J. et al., 2009. OWL 2 Web Ontology Language: Quick Reference Guide. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027.

Battle, S. et al., 2005. Semantic Web Services Ontology (SWSO). *W3C Member Submission 9 September 2005*. Available at: http://www.w3.org/Submission/SWSF-SWSO/.

Bechhofer, S. et al., 2004. OWL Web Ontology Language - Reference.

Becket, D., 2004. RDF/XML Syntax Specification (Revised). *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ [Accessed October 30, 2010].

Bernus, P., 2003. Enterprise models for enterprise architecture and ISO9000:2000. *Annual Reviews in Control*, 27(2), pp.211-220. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1367578803000294 [Accessed October 2, 2010].

Brickley, D. & Guha, R.V., 2004. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/2004/REC-rdf-schema-20040210 [Accessed October 30, 2010].

Carroll, J., Herman, I. & Patel-Schneider, P.F., 2009. OWL 2 Web Ontology Language: RDF-Based Semantics. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/ [Accessed November 4, 2010].

Celino, I. et al., 2007. Semantic business process analysis. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007)*.

Ceravolo, P., Fugazza, C. & Reed, K., 2008. Enforcing and monitoring company policies on business process orchestrations. In *2nd IEEE International Conference on Digital Ecosystems and Technologies, DEST 2008*. pp. 65-70.

Connolly, D. et al., 2001. DAML+OIL (March 2001) Reference Description. *W3C Note 18 December 2001*. Available at: http://www.w3.org/TR/daml+oil-reference [Accessed November 2, 2010].

Corcho, O., with Gómez-Pérez, A., Fernández-López, 2004. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, London: Springer.

Davis, R., 2008. *ARIS Design Platform - Advanced Process Modelling and Administration*, London: Springer-Verlag.

de Bruijn, J. et al., 2005. OWL DL vs. OWL flight: conceptual modeling and reasoning for the semantic Web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA: ACM Press, pp. 623-632. Available at: http://dx.doi.org/10.1145/1060745.1060836.

De Bruijn, J. et al., 2005. D16.1v0.21 The Web Service Modeling Language WSML. *WSML Final Draft 5 October 2005*.

Di Francescomarino, C. et al., 2008. Reasoning on Semantically Annotated Processes. In *Proceedings of the International Conference on Service-Oriented Computing*. pp. 132-146. Available at: http://selab.fbk.eu/tonella/papers/icsoc2008.pdf.

Di Francescomarino, C. et al., 2009. Semantically-aided business process modeling. *The Semantic Web-ISWC 2009*, p.114–129. Available at: http://data.semanticweb.org/pdfs/iswc/2009/paper278.pdfhttp://data.semanticweb.org/pdfs/iswc/2009/paper278.pdf [Accessed January 13, 2011].

Dimitrov, M. et al., 2007. A BPMO Based Semantic Business Process Modelling Environment. In M. Hepp et al., eds. *SBPM*. CEUR-WS.org. Available at: http://dblp.uni-trier.de/db/conf/esws/sbpm2007.html#DimitrovSSK07.

Feldkamp, D., Hinkelmann, K. & Thönssen, B., 2007. KISS-knowledge-intensive service support: an approach for agile process management. In *Proceedings of the 2007 international conference on Advances in rule interchange and applications*. Berlin, Heidelberg: Springer-Verlag, pp. 25-38. Available at: http://portal.acm.org/citation.cfm?id=1785383.1785388.

Fischer, R., Aier, S. & Winter, R., 2007. A federated approach to enterprise architecture model maintenance. In M. Reichert, S. Strecker, & K. Turowski, eds. *Enterprise Modelling and Information Systems Architectures*. Bonn: Gesellschaft für Informatik, pp. 9-22. Available at: http://www.alexandria.unisg.ch/publications/67486 [Accessed January 25, 2011].

Fox, M.S. & Gruninger, M., 1998. Enterprise Modeling. *AI Magazine*, 19(3), pp.109-122.

Fox, M.S. & Grüninger, M., 1997. On Ontologies And Enterprise Modelling. In *International Conference on Enterprise Integration Modelling Technology 97*. Springer-Verlag.

Franke, U. et al., 2009. EAF2- A Framework for Categorizing Enterprise Architecture Frameworks. In *Software Engineering, Artificial Intelligences, Networking and*

Parallel/Distributed Computing, 2009. SNPD '09. 10th ACIS International Conference on. pp. 327-332.

Ghidini, C., Rospocher, M. & Serafini, L., 2008. *A formalisation of BPMN in description logics*, Available at: https://dkm.fbk.eu/index.php/BPMN_Related_Resources [Accessed January 13, 2011].

Golbreich, C. & Wallace, E.K., 2009. OWL 2 Web Ontology Language - New Features and Rationale. *W3C Recommendation 27 October 2009*.

Grüninger, M., Hull, R. & McIlraith, S., 2005. A First-Order Ontology for Semantic Web Services.

Guarino, N. & Giaretta, P., 1995. Ontologies and knowledge bases: Towards a terminological clarification. In N. J. I. Mars, ed. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. Amsterdam: IOS Press, pp. 25-32. Available at: http://www.citeulike.org/user/tobiasweigel/article/4745825 [Accessed November 7, 2010].

Guarino, N., Oberle, D. & Staab, S., 2009. What Is an Ontology. In S. Staab & R. Studer, eds. *Handbook on Ontologies*. Berlin Heidelberg: Springer-Verlag, pp. 1-17.

Hallmark, G., 2008. Smart Business Processes using Oracle Business Rules. Available at: http://www.oracle.com/technetwork/middleware/ias/smart-processes-obr-1-129729.pdf.

Handler, R.A. & Wilson, C., 2009. *Magic Quadrant for Enterprise Architecture Tools*, Available at: http://imagesrv.gartner.com/media-products/pdf/reprints/ibm/external/volume4/article28.pdf.

Hanschke, I., 2010. *Strategisches Management der IT-Landschaft - Ein praktischer Leitfaden für das Enterprise Architecture Management* 2nd ed., München: Carl Hanser Verlag.

Harel, D. & Rumpe, B., 2004. Meaningful modeling: what s the semantics of "semantics"? *Computer*, 37(10), pp.64-72.

Hayes, P., 2004. RDF Semantics. *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ [Accessed November 30, 2010].

He, X. et al., 2007. A metamodel for the notation of graphical modeling languages. In *Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01*. Washington, DC, USA: IEEE Computer Society, pp. 219-224. Available at: http://dx.doi.org/10.1109/COMPSAC.2007.27.

Hepp, M. et al., 2005. Semantic business process management: a vision towards using semantic Web services for business process management. In *e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on*. pp. 535-540.

Hinkelmann, K., Nikles, S. & Arx, L.V., 2007. An Ontology-based Modeling Tool for Knowledge- intensive Services. In *Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007) in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck*. pp. 1-4.

Hinkelmann, K. et al., 2007. An ontology-based modelling tool for knowledge intensive e-government services. In A. Corradini, F., Polzonetti, ed. *Proceedings of the 1st International Conference on Methodologies, Technologies and Tools Enabling e-Government, MeTTeG07*. Halley Editrice SRL, p. 43–56.

Hinkelmann, K. et al., 2010. An Enterprise Architecture Framework to organize Model Repositories. In A. Woitsch, Robert and Micsik, ed. *OKM Open Knowledge Models, Workshop W3 at EKAW 2010*.

Hitzler, P. et al., 2009. OWL 2 Web Ontology Language: Primer. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/2009/REC-owl2-primer-20091027 [Accessed November 1, 2010].

Hitzler, P. et al., 2008. *Semantic Web*, Berlin Heidelberg: Springer Verlag.

Horridge, M. & Patel-Schneider, P.F., 2009. OWL 2 Web Ontology Language - Manchester Syntax. *W3C Working Group Note 27 October 2009*. Available at: http://www.w3.org/TR/owl2-manchester-syntax/.

Horrocks, I. et al., 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission 21 May 2004*. Available at: http://www.w3.org/Submission/SWRL/.

Hysom, R., 2003. Enterprise Modelling - The Readiness of the Organization. In P. Bernus, L. Nemes, & G. Schmidt, eds. *Handbook on Enterprise Architecture*. Berlin Heidelberg: Springer-Verlag, pp. 374-415.

IEEE, 2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE Std 1471-2000*.

Janusch, W., 2009. *SUPER (Semantics Utilised for Process management within and between EnteRprises) Deliverable 4.3, Behavioural Process Mediation Reasoning Component*, Available at: http://www.ip-super.org/res/Deliverables/M36/D4.3.pdf.

Junisphere Systems AG, 2010. Solution Overview - Whitepaper, Version 3.2. Available at: http://www.junisphere.ch/wp/JS_doc_CTO_2010-09_Solution-Overview-v3.2.pdf.

Karagiannis, D., 1995. BPMS: business process management systems. *SIGOIS Bull.*, 16(1), pp.10-13. Available at: http://doi.acm.org/10.1145/209891.209894.

Karagiannis, D. & Kühn, H., 2002. Metamodelling Platforms. In K. Bauknecht, A. Min Tjoa, & G. Quirchmayer, eds. *Proceedings of the Third International Conference EC-Web 2002 - Dexa 2002*. LNCS 2455. Berlin, Heidelberg: Springer-Verlag, 182.

Karastoyanova, D. et al., 2008. A Reference Architecture for Semantic Business Process Management Systems. In M. Bichler et al., eds. *Multikonferenz Wirtschaftsinformatik 2008*. Berlin: GITO-Verlag, pp. 371-372. Available at: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2008-09&engl=1.

Kim, H.M. & Fox, M.S., 2002. Using Enterprise Reference Models for Automated ISO 9000 Compliance Evaluation. In *Proceedings of the 35th Hawaii International Conference on Systems Science*.

Kim, H.M., Fox, M.S. & Sengupta, A., 2007. How To Build Enterprise Data Models To Achieve Compliance To Standards Or Regulatory Requirements (and share data). *Journal of the Association for Information Systems*, 8(2), pp.105-128.

Klyne, G. & Carroll, J.J., 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/rdf-concepts [Accessed October 30, 2010].

Kähmer, M. & Gilliot, M., 2009. Extended Privacy Definition Tool. In A. Heinzl et al., eds. *PRIMIUM - Process Innovation for Enterprise Software*. Bonn: Gesellschaft für Informatik, pp. 43-59. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Extended+Privacy+Definition+Tool#0 [Accessed February 6, 2011].

Kähmer, M., Gilliot, M. & Müller, G., 2008. Automating Privacy Compliance with ExPDT. In *10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2008*. IEEE, p. 87–94. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4785051 [Accessed February 6, 2011].

Kühne, T., 2005. What is a Model? In J. Bezivin & R. Heckel, eds. *Language Engineering for Model-Driven Software Development*. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

Lankhorst, M. et al, 2009. *Enterprise Architecture at Work: Modelling, Communication and Analysis* 2nd ed., Berlin Heidelberg: Springer.

Manola, F. & Miller, E., 2004. RDF Primer. *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/rdf-primer/.

Martin, D. et al., 2004. OWL-S: Semantic Markup for Web Services. *W3C Member Submission 22 November 2004*. Available at: http://www.w3.org/Submission/OWL-S/.

McGuinness, D.L. & van Harmelen, F., 2004. OWL Web Ontology Language. *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/2004/REC-owl-features-20040210 [Accessed October 30, 2010].

Motik, B. et al., 2009. OWL 2 Web Ontology Language: Profiles. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/2009/REC-owl2-profiles-20091027 [Accessed November 1, 2010].

Motik, B., Patel-Schneider, P.F. & Parsia, B., 2009. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/ [Accessed November 1, 2010].

Motik, B., Sattler, U. & Studer, R., 2004. Query Answering for OWL-DL with Rules. In S. A. McIlraith, D. Plexousakis, & F. van Harmelen, eds. *Proc. of the 3rd Int. Semantic Web*

Conference (ISWC 2004). Hiroshima, Japan, November 7–11 2004: Springer, pp. 549-563. Available at: https://www.comlab.ox.ac.uk/people/boris.motik/pubs/mss04dl-safe.pdf.

Na, H.-S., Choi, O.-H. & Lim, J.-E., 2006a. A Metamodel-Based Approach for Extracting Ontological Semantics from UML Models. In K. Aberer et al., eds. *Web Information Systems - WISE 2006*. Springer Berlin / Heidelberg, pp. 411-422. Available at: http://dx.doi.org/10.1007/11912873_43.

Na, H.-S., Choi, O.-H. & Lim, J.-E., 2006b. A Method for Building Domain Ontologies based on the Transformation of UML Models. In *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*. pp. 332-338.

Nikles, S. & Brander, S., 2009. Separating Conceptual and Visual Aspects in Meta- Modelling. In *Proceedings of AER 2009 - 1st International Workshop on Advanced Enterprise Repositories, Milan*.

Nitzsche, J., Wutke, D. & van Lessen, T., 2007. An Ontology for Executable Business Processes. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007)*. Innsbruck, Austria. Available at: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-251/paper8.pdf.

Obitz, T. & Babu K, M., 2009. *Enterprise Architecture Expands its Role in Strategic Business Transformation - Infosys Enterprise Architecture Survey 2008/2009*, Available at: http://www.infosys.com/offerings/it-services/architecture-services/ea-survey/.

OMG, 2010. Business Process Model and Notation (BPMN) - Version 2.0. , (June).

OMG, 2009a. OMG Unified Modeling Language (OMG UML) - Superstructure, Version 2.2. , (February).

OMG, 2009b. Ontology Definition Metamodel - Version 1.0. , 09(May).

OMG, 2008. Semantics of Business Vocabulary and Business Rules (SBVR). , (January).

O'Connor, M.J. & Das, A.K., A Method for Representing and Querying Temporal Information in OWL. In *Biomedical Engineering Systems and Technologies*. Springer. Available at: http://bmir.stanford.edu/file_asset/index.php/1534/BMIR-2010-1414.pdf [Accessed December 12, 2010].

Patel-Schneider, P.F. & Horrocks, I., 2004. OWL Web Ontology Language - Semantics and Abstract Syntax - Section 2. Abstract Syntax. *W3C Recommendation 10 February 2004*. Available at: http://www.w3.org/TR/owl-semantics/syntax.html [Accessed November 30, 2010].

Pedrinaci, C. & Domingue, J., 2007. Towards an ontology for process monitoring and mining. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management*. Innsbruck, Austria. Available at: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-251/paper10.pdf.

Pedrinaci, C. et al., 2008. SENTINEL: a semantic business process monitoring tool. In *Proceedings of the first international workshop on Ontology-supported business*

intelligence. New York, NY, USA: ACM, p. 1:1--1:12. Available at: http://doi.acm.org/10.1145/1452567.1452568.

Raub, D. & Steinwandt, R., 2006. An Algebra for Enterprise Privacy Policies Closed Under Composition and Conjunction. In Günter Müller, ed. *Emerging Trends in Information and Communication Security*. Berlin, Heidelberg: Springer-Verlag, pp. 130-144.

Roman, D. et al., 2005. Web Service Modeling Ontology. *Applied Ontology*, 1(1/2005), pp.77-106.

Saunders, M., Lewis, P. & Thornhill, A., 2009. *Research methods for business students* 5th ed., Harlow: Financial Times Prentice Hall.

Schekkerman, J., 2009. Enterprise Architecture Tool Selection Guide - Version 5.0. Available at: http://www.enterprise-architecture.info/Images/EA Tools/Enterprise Architecture Tool Selection Guide v50.pdf.

Schmidt, R., Bartsch, C. & Oberhauser, R., 2007. Ontology-based Representation of Compliance Requirements for Service Processes. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM-2007)*.

Smith, M. et al., 2009. OWL 2 Web Ontology Language: Conformance. *W3C Recommendation 27 October 2009*. Available at: http://www.w3.org/TR/owl2-conformance/ [Accessed November 4, 2010].

Sowa, J.F. & Zachman, J.A., 1992. Extending and formalizing the framework for information systems architecture. *IBM Systems Journal*.

Studer, R., Benjamins, V.R. & Fensel, D., 1998. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2), pp.161-197. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0169023X97000566.

The Open Group, 2009. TOGAF 9 - The Open Group Architecture Framework Version 9. , p.744.

Urbaczewski, L. & Mrdalj, S., 2006. A Comparison of Enterprise Architecture Frameworks. *Issues in Information Systems*, 7(2), pp.18-23. Available at: http://www.iacis.org/iis/2006_iis/PDFs/Urbaczewki_Mrdalj.pdf.

Vaishnavi, V. & Kuechler, W., 2004. Design Research in Information Systems. *January 20, 2004, last updated August 16, 2009*. Available at: http://desrist.org/design-research-in-information-systems [Accessed September 20, 2010].

van Buuren, R. et al., 2004. Composition of Relations in Enterprise Architecture Models. In H. Ehrig et al., eds. *Graph Transformations*. Springer Berlin / Heidelberg, pp. 183-186. Available at: http://dx.doi.org/10.1007/978-3-540-30203-2_5.

Veres, C., 2005. Aggregation in Ontologies: Practical Implementations in OWL. In D. Lowe & M. Gaedke, eds. *Web Engineering*. Springer Berlin / Heidelberg, pp. 285-295. Available at: http://dx.doi.org/10.1007/11531371_39.

Wetzstein, B., Ma, Zhilei & Leymann, F., 2008. Towards Measuring Key Performance Indicators of Semantic Business Processes. In W. Abramowicz & D. Fensel, eds. *Business Information Systems*. Springer Berlin Heidelberg, pp. 227-238. Available at: http://dx.doi.org/10.1007/978-3-540-79396-0_20.

Wilson, C. & Short, J., 2010. Magic Quadrant for Enterprise Architecture Tools. , 2(October). Available at: http://www.gartner.com/technology/media-products/reprints/softwareag/volume2/article10/article10.html [Accessed December 4, 2010].

Witschel, H.F. et al., 2010. A collaborative approach to maturing process-related knowledge. In *Proceedings of the 8th international conference on Business process management*. Berlin, Heidelberg: Springer-Verlag, pp. 343-358. Available at: http://portal.acm.org/citation.cfm?id=1882061.1882093.

Zachman, J.A., 1996. Enterprise Architecture : The Issue of the Century. *Design*.

Zachman, J.A., 1987. A framework for information systems architecture. *IBM Systems Journal*, 26(3), pp.276-292.

Österle, H., 1995. *Business in the Information Age - Heading for New Processes*, Berlin: Springer-Verlag.

List of Figures

List of Tables

Appendix 1 List of Abbreviations

BPEL	Business Process Execution Language
BPMN	Business Process Modelling Notation
CWA	Closed World Assumption
EA	Enterprise Architecture
MOF	Meta Object Facility
OWA	Open World Assumption
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SBVR	Semantics of Business Vocabulary and Rules
SWRL	Semantic Web Rule Language
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WSML	Web Service Modelling Language
WSMO	Web Service Modelling Ontology
XML	Extensible Markup Language