

Frank, Ulrich; Kaczmarek-Heß, Monika; de Kinderen, Sybren

Research Report

IT Infrastructure Modeling Language (ITML): A DSML for supporting IT management

ICB-Research Report, No. 72

Provided in Cooperation with:

University Duisburg-Essen, Institute for Computer Science and Business Information Systems (ICB)

Suggested Citation: Frank, Ulrich; Kaczmarek-Heß, Monika; de Kinderen, Sybren (2021) : IT Infrastructure Modeling Language (ITML): A DSML for supporting IT management, ICB-Research Report, No. 72, Universität Duisburg-Essen, Institut für Informatik und Wirtschaftsinformatik (ICB), Essen,
<https://doi.org/10.17185/dupublico/75252>

This Version is available at:

<https://hdl.handle.net/10419/248827>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.



ICB

Institut für Informatik und
Wirtschaftsinformatik

Ulrich Frank, Monika Kaczmarek-Heß, Sybren de Kinderen



IT Infrastructure Modeling Language (ITML): A DSML for Supporting IT Management

ICB-RESEARCH REPORT

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik stellen vorläufige Ergebnisse dar, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Daher sind die Autoren für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results, which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Die durch das Urheberrecht begründeten Rechte, insbesondere der Übersetzung, des Nachdruckes, des Vortrags, der Vervielfältigung, der Weitergabe, der Veränderung und der Entnahme von Abbildungen und Tabellen – auch bei auszugsweiser Verwertung – bleiben vorbehalten.

All rights reserved. No part of this report may be reproduced by any means, or translated.

Authors' Address:

Ulrich Frank, Monika Kaczmarek-Heß,
Sybren de Kinderen
University of Duisburg-Essen
Institute for Computer Science
and Business Informatics
Universitätsstr. 9, 45141 Essen, Germany

{ulrich.frank|monika.kaczmarek-hess|
sybren.dekinderen}@uni-due.de

ICB Research Reports

Edited by:

Prof. Dr. Frederik Ahlemann
Prof. Dr. Fabian Beck
Prof. Dr. Torsten Brinda
Prof. Dr. Peter Chamoni
Prof. Dr. Lucas Davi
Prof. Dr. Klaus Echtele
Prof. Dr. Stefan Eicker
Prof. Dr. Ulrich Frank
Prof. Dr. Michael Goedicke
Prof. Dr. Volker Gruhn
Prof. Dr. Tobias Kollmann
Prof. Dr. Pedro José Marrón
Prof. Dr. Klaus Pohl
Prof. Dr. Erwin P. Rathgeb
Prof. Dr. Stefan Schneegaß
Prof. Dr. Reinhard Schütte
Prof. Dr. Stefan Stieglitz

Contact:

Institute for Computer Science and
Business Information Systems (ICB)
University of Duisburg-Essen
Universitätsstr. 9
45141 Essen – Germany
Tel.: +49 201-1834041
Fax: +49 201-1834011
Email: icb@uni-duisburg-essen.de

ISSN 1860-2770 (Print)

ISSN 1866-5101 (Online)

DOI 10.17185/duepublico/75252

Abstract

In this research report, we present a new version of IT Modeling Language (ITML). It is integrated with the family of Domain-Specific Modeling Languages (DSMLs) that are part of Multi-Perspective Enterprise Modeling (MEMO). The design of the language followed a proven method. It provides for analysing requirements based on the analysis of possible use scenarios. The ITML was specified with the meta modeling language MEMO MML. Due to the considerable size of the meta model, its presentation is split into several partial meta models, which are described at a level of detail that is required for the implementation of the language within a modeling tool. The presentation of the meta model is supplemented by discussions of design conflicts and related decisions. In addition to the abstract syntax and semantics, which are represented by the meta model, we also propose a concrete syntax, which was designed with the support of a graphic designer. Finally, we also provide a short discussion of principal limitations of conventional meta modeling as well as an outlook on our future research.

Keywords: IT management, enterprise modeling, IT infrastructure, ITML, MEMO, multi-level modeling.

Contents

1	Introduction	1
2	Main Analysis Scenarios	3
2.1	Analysis of IT Landscape Elements	3
2.1.1	Identification of Existing IT Landscape Elements	4
2.1.2	Analysis of Main Characteristics of Existing IT Infrastructure Elements	6
2.1.3	Analysis of Dependencies Among IT Landscape Elements	7
2.2	IT-centric Analysis	8
2.2.1	Security Analysis	8
2.2.2	Maintainability	9
2.2.3	Portability Analysis	10
2.2.4	Performance Analysis	11
2.2.5	Availability Analysis	12
2.2.6	Analysis of Vendors and Existing Relationships	12
2.3	Integration Analysis	13
2.4	Integrated IT Infrastructure and Action System Analysis	16
2.4.1	Analysis of IT-Business Alignment	16
2.4.2	Analysis of Organizational Assignment	17
3	Language Design: Abstract Syntax and Semantics	19
3.1	Main IT Landscape Elements	21
3.2	Selected IT-centric Analysis	40
3.3	Integration: Technologies, Languages, and Conceptual Integration	41
3.4	IT Architecture	44
3.5	Integrated IT Infrastructure and Action System Analysis	47
3.6	Auxiliary Types	51
3.7	Constraints	52
3.8	Requirements and Their Fulfillment	56
4	Language Design: Concrete Syntax and ITML Diagram Types	60
4.1	Concrete Syntax	60
4.2	Selected ITML Diagram Types	62

4.2.1	IT Infrastructure Diagram	62
4.2.2	Topic Diagram and Corresponding Analysis	67
4.2.3	Architecture Diagram	68
4.3	Integration of ITML Diagram Types With Other MEMO Diagram Types	70
4.4	Selected Design Principles	71
5	Conclusions	77
A	Concrete Syntax – Concepts	81
	Bibliography	86

1 Introduction

IT management incorporates a variety of topics that range from the development and operation of enterprise information systems, to the generation of value through the use of IT (Hanschke 2010; Luftman and Bullen 2004; Mangiapane and Buechler 2015). IT management encompasses planning, controlling and organizing aspects involving IT infrastructure (Hanschke 2010). In turn, an IT infrastructure encompasses (1) different kinds of IT artifacts such as, e.g., computational hardware, software, data, networks and data center facilities, (2) various dependencies among those artifacts, as well as (3) actors designing or maintaining the IT infrastructure, cf. (Duncan 1995; Hanschke 2010; Laan 2017; Nyrhinen 2006). However, what exactly is to be perceived as part of IT infrastructure, and at which level of granularity, depends to a large extent on stakeholders involved and/or IT management analyses targeted, cf. (Kaczmarek-Heß and Kinderen 2017; Laan 2017).

In the era of digital transformation, elements of an organization action system, such as business processes, business goals, strategies, or organizational structures, are heavily affected by the adoption of digital technologies (Parviainen et al. 2017). Acknowledging the importance of IT infrastructure management for an enterprise as a whole, different modeling languages in the field of enterprise modeling have been proposed, among others, to: (1) promote better understanding of an enterprise and the role IT plays in it, (2) creating a common language between business and IT, thus making explicit links between the action system (e.g., processes, goals) and IT artifacts (e.g., information systems, IT services), and thus, explicitly showing the complex inter-dependencies that exist among and within these domains, as well as (3) providing support for processes that are considered to be at the core of IT management and which aim at documenting and analysing IT landscape, as well as at governing and controlling its evolution¹. Examples of such approaches are, among others, ArchiMate (The Open Group 2012), Architecture of Integrated Information Systems (ARIS) (Scheer 2001), and Multi-Perspective Enterprise Modeling (MEMO) (Frank 2014b), with an IT Modeling Language called ITML.

Although, conceptual overlaps between these IT modeling languages can be assumed, as all of them are oriented towards developing structured descriptions of IT infrastructure, the existing modeling approaches differ substantially when it comes to the modeling foundations and assumptions, domain coverage, as well as semantic richness of offered concepts, and thus, also the set of scenarios supported, cf. (Bock, Kaczmarek, et al. 2014; Kinderen and Kaczmarek-Heß

¹For overview of IT management processes as such see, e.g., (Hanschke 2010)

2018). In this research report, we focus on the ITML being part of a comprehensive method for multi-perspective enterprise modeling (MEMO) (Frank 2014b). MEMO includes various other domain-specific modeling languages (DSMLs), e.g., languages for business processes, resources, or goal modeling. Being part of MEMO, ITML models can be enhanced with the relevant aspects of an enterprise to allow for analysis in an IT-business alignment context, and foster communication between stakeholders with different professional backgrounds.

The ITML as proposed by Frank et al. (2009) and further extended by Heise (2013), and Kinderen and Kaczmarek-Heß (2018), offers the basic concepts reconstructed from the domain of discourse, e.g., Software, Computer, Server, Peripheral Device, Service, Physical Medium or Application System. However, the concepts offered do not allow to support all analysis scenarios that are currently of interest, as well as, considering the current developments, do not account for all relevant aspects and technologies (e.g., containers, distributed ledger). In this research report, driven by the need to further extend the set of analysis scenarios that should be supported, we present a new version of the ITML. Here we focus on the definition of analysis scenarios, which should be supported by the ITML, and associated conceptualizations in terms of its abstract syntax created with the MEMO Meta Modeling Language (MEMO MML) (Frank 2011), semantics, as well as the concrete syntax and associated diagram types.

Creating a domain-specific modeling language is a non-trivial endeavor, cf. (Frank 2013), involving not only challenges regarding the reconstruction of domain concepts, but also challenges surrounding meta modeling specifically, such as deciding what concepts are to be part of the language specification, and what concepts are to be part of the language application. Therefore, to support the DSML design process, we follow the DSML design method as proposed by Frank (2013). This method, which already has proven useful in other projects (e.g., Goldstein and Frank 2016; Kinderen and Kaczmarek-Heß 2018; Overbeek, Frank, and Köhling 2015), provides a macro-process for language design, as well as corresponding roles and guidelines. It consists of 7 steps: starting from the clarification of scope and purpose, through the analysis of requirements, specification of language (abstract syntax and semantics), provision of graphical notation (concrete syntax) and the optional development of a modeling tool. The process ends with the evaluation and refinement of the developed language, and (potentially) the corresponding software tool. As already mentioned, in this report we discuss analysis scenarios of interest, requirements, as well as the design of a language.

The research report is structured as follows. First, we provide an overview of typical IT infrastructure analyses, and derive a set of high-level requirements towards a language. Then, the abstract syntax, language semantics as well as selected design decisions are shortly discussed. Next, a short overview of the concrete syntax follows, complemented by a short presentation of selected diagram types. Finally, in the conclusions an outlook is provided on challenges associated with modeling IT infrastructure with conventional meta modeling, as well as on promises of the multi-level paradigm.

2 Main Analysis Scenarios

The main goal of the IT Modeling Language (ITML) is to provide support for the needs of IT management, cf. (Hanschke 2010; Luftman and Bullen 2004; Mangiapane and Buechler 2015). As the ITML's design is driven by use scenarios, which we consider being central to the development of modeling languages (Frank 2013), in this chapter, we discuss the main use scenarios for the ITML. We do this in terms of both (i) analyses of the IT infrastructure itself that should be supported by a standalone IT Infrastructure modeling language, and (ii) analyses that cut across the IT infrastructure and other organizational perspectives (such as business processes, organizational goals, organizational structure, or otherwise) that are of interest, if an IT modeling language is part of an enterprise modeling approach.

The scenarios discussed in this chapter are accompanied by the formulation of a set of requirements. These requirements, in turn, will inform the design of the ITML, as discussed in Chapter 3. In our discussion we also briefly touch upon the need for a modeling language and analysis of models in order to fulfill the identified requirements.

2.1 Analysis of IT Landscape Elements

The analysis of elements of an IT landscape and their characteristics is the basic scenario being of interest for IT management (Hanschke 2010; Luftman and Bullen 2004; Mangiapane and Buechler 2015). Namely, based on the analysis of the created diagram, it should be possible to answer questions regarding (1) IT infrastructure elements, (2) their properties and functionalities they provide, and finally, (3) relationships and dependencies between those elements. Please note that we are interested in both information applicable to types of IT artifacts, e.g., properties of an operating system, or of some type of hardware platform; as well as information applicable to some specific installations (e.g., specific installation of some specific version of an operating system on some specific hardware platform in some specific location) and exemplars (e.g., configuration and properties of some specific exemplar of some model of printer). Thus, we are interested in both information applicable to the type level, as well as the instance level. This leads us to the following requirement (R).

2 Main Analysis Scenarios

R1: The IT infrastructure modeling language should provide differentiated concepts supporting comprehensive modeling of IT infrastructure elements, their properties and dependencies between different elements, both on the type-level as well as on the instance-level.

It is noteworthy that in emphasizing the modeling of, both, type and instance level information, the ITML should build on a language architecture that is equipped with features allowing to differentiate between these two types of information.

2.1.1 Identification of Existing IT Landscape Elements

Purpose: Analysis of IT landscape elements focuses on answering questions regarding elements belonging to IT infrastructure and their role.

From a technical perspective, the IT infrastructure is used to collect, store and process data. This requires hardware that provides computing power, as well as temporary and persistent storage. It also includes various devices for recording and providing data, as well as infrastructure allowing for exchange of messages between components. Software is required to make these basic hardware functions accessible in a safe and convenient way. Basic software, such as an operating system, serves to enable an abstraction from the resources provided with the hardware (storage space, processor capacity, etc.). This not only reduces the dependency on specific hardware, but also creates convenient access to the hardware. The separation between hardware and software is fluid. Functions that are provided by software can also be implemented using dedicated hardware, for example to achieve performance advantages. Conversely, hardware can be simulated by software. The software that is used to manage the hardware resources, as well as the data on which it operates, also represent resources that are used by other software systems.

IT infrastructures usually encompass several computers connected via a network and other devices. Therefore, functionalities are required that support distribution of resources, as well as dealing with heterogeneity. Here, heterogeneity primarily refers to different processor architectures, different basic software and different programming languages. A distributed system consists of several computers that are connected via a network with one another. The resources of a distributed system are to be managed in a similar way to the resources of a single computer. Corresponding functionalities include administration and access to distributed storage systems and peripheral devices, loading, executing and scheduling programs, and protecting distributed resources from unauthorized access. Since the resources of a distributed IT infrastructure are sometimes used by several users simultaneously (“multi-user operation”), functions are required that synchronize concurrent access to resources in such a way that the integrity of the accesses is preserved. For this, the secure execution of transactions must be supported.

Exemplary questions of interest:

- Which types of hardware platform are used?
- What printers are there?
- Do we have any application server running?
- Which type of an enterprise system do we use?
- How many licenses for some operating system are available?
- What IT-services are offered?
- What functionalities are offered by IT landscape elements?
- What application programming interfaces are offered?
- Do we employ a distributed ledger technology?
- Do we employ inductive systems, e.g., inductive reasoners¹?
- What networks are there?
- Where are our servers located?

Although many of the exemplary questions formulated may seem to be easily answered by getting information, e.g., from a configuration management database, as subsequent sections show, please note that the analysis scenarios that we target at are much more complex in nature and require not only looking at some selected properties of IT artifacts, but defining and analyzing complex dependencies among them, processing acquired information, as well as communicating it effectively. To this aim application of conceptual models created using a dedicated domain-specific modeling language is particularly suitable.

Key concepts: various types of hardware as well as software, interface, network, network access, data storage, location

Requirements:

R2: The IT infrastructure modeling language should provide differentiated concepts supporting comprehensive modeling of hardware platforms and associated concepts.

R3: The IT infrastructure modeling language should provide differentiated concepts supporting comprehensive modeling of software artifacts and related concepts.

¹In recent years, various systems that make use of inductive approaches received growing attention. They comprise statistics software, e.g., for data analysis, or “machine learning” approaches. We use the term ‘inductive system’ to name this class of software.

2 Main Analysis Scenarios

R4: The IT infrastructure modeling language should provide differentiated concepts supporting comprehensive modeling of IT services.

R5: The IT infrastructure modeling language should provide differentiated concepts supporting comprehensive modeling of hardware devices (e.g., peripheral devices).

R6: The IT infrastructure modeling language should provide differentiated concepts supporting comprehensive modeling of other elements of IT infrastructure, such as network or data storage.

2.1.2 Analysis of Main Characteristics of Existing IT Infrastructure Elements

Purpose: The relevant characteristics to be covered, are partly determined by the analysis scenarios of interest to IT management, for a partial overview, cf. (Bucher et al. 2006; Hanschke 2010; Niemann 2005). These scenarios range from relatively straightforward, such as the life-cycle stage of an IT infrastructure element, to more comprehensive ones. For scoping purposes we focus here on relatively straightforward analysis questions, whereas more comprehensive ones are covered in the IT-centric analysis described in Section 2.2.

Exemplary questions of interest:

- What functions are delivered by some software artifact?
- Which implementation language was used to create some software artifact?
- What is the size of the external storage, e.g., external hard-drive used?
- What is the printing speed of a given printer?
- What is the type of the software licenses in use?
- Which programming languages, of which we are explicitly aware, are used in-house?
- What is the rationale for using a particular distributed ledger technology?
- What is the rationale for using a particular inductive system?
- Which interface an application is using? How is the application and/or the interface implemented in technical terms?
- What type of middleware are we using in our architecture?
- What is the life-cycle stage of a given IT artifact?
- What data and on what topic is exchanged via existing interfaces?

- How is data used by some software applications, meaning: does the application read or write data?
- What is the assessment of application software in terms of user satisfaction?

Key concepts: Properties of software and hardware artifacts, related concepts such as interface, programming language, communication protocol, transport protocol, data, data format, etc.

Requirements:

R7: The IT infrastructure modeling language should express properties of different IT infrastructure concepts as they are part of the professional IT infrastructure discourse. For example, this includes properties of a hardware platform or a software application, as described under the identification of IT infrastructure elements.

2.1.3 Analysis of Dependencies Among IT Landscape Elements

Purpose: The main focus is to analyze dependencies and relationships between existing IT infrastructure elements. Also, among others, analyses on location and deployment options are here in focus. In a most trivial case, we can deploy the software on premises or in the cloud. Please note that each option has benefits and drawbacks. E.g., on the one hand, cloud deployment allows to abstract from the complexity of managing the physical infrastructure and offer a flexible, cost-effective business model. However, on the other hand, cloud deployment comes at the cost of losing control over the physical IT infrastructure. Finally, deployment can take the shape of so-called containers, which are said to provide everything required to run a program, cf. (Ernst, Bermbach, and Tai 2016; Newman 2015; Sultan, Ahmad, and Dimitriou 2019; Syed and Fernandez 2018), such as libraries, APIs, etc. Containers are especially relevant, if single applications are designed as being composed of many loosely coupled and independently deployable components or services, and it is of interest to, e.g., make deployment dependencies explicit (Ernst, Bermbach, and Tai 2016; Newman 2015; Sultan, Ahmad, and Dimitriou 2019; Syed and Fernandez 2018).

Exemplary questions of interest:

- What are the dependencies among existing software artifacts?
- Which software communicates with which software and what are the characteristics of such a communication (e.g., protocol used, data exchanged)?
- Which web server is part of a middleware solution?
- Which software is wrapped by a component?
- Which software artifacts run on which platforms?

2 Main Analysis Scenarios

- Where are different (hardware) platforms physically located?
- Where is application software deployed/hosted? Is it running on our premises or in the cloud environment?
- Can some IT artifact also be used with other hardware and/or within other software environments?
- What IT services are offered by an application?
- Which IT services are used by some application?
- Which interfaces are used by IT services?

Key concepts: IT service, software and its specializations, hardware and its specializations, runs on, executable on, requires, provides, uses

Requirements:

R8: The IT infrastructure modeling language should allow to model dependencies among various elements of the IT landscape (e.g., between software artifacts, software and platforms, or software artifacts and IT services), as indicated by the questions posed above.

2.2 IT-centric Analysis

In the following, we discuss specific IT-centric analysis scenarios focusing on qualitative features of IT artefacts. Although there is no commonly accepted notion of “quality” of IT artefacts, to organize our discussion according to a common denominator, we use the well-established Systems and software Quality Requirements and Evaluation (SQuaRE) model (ISO 2011).

2.2.1 Security Analysis

Purpose: while security is important to IT management, it is still unclear how to suitably assess the level of security in a company, since the definition of the topic is ambiguous (Goldstein and Frank 2016; Johansson and Johnson 2005). We assume that analysis of security in its simplest form focuses on three elements: authorization, authentication, and confidentiality. ITML should allow to conduct a basic analysis of security related aspects for those three elements.

Exemplary questions of interest:

- Is the network protected by a firewall?

- Is the transport protocol used secure?
- Is the exchanged data encrypted? What kind of encryption is used?
- Is our server storing the sensitive data in the protected network?
- Are all of the desktops and notebooks protected by a personal firewall?
- Is the exchange of sensitive data protected?
- Which process types and which organizational units are accessing which data? In which mode (read or write)?

Key concepts: Properties of software and hardware artifacts, supporting concepts such as firewall, transport protocol, encryption method

Requirements:

R9: The IT infrastructure modeling language should account for supporting basic analysis related to authorization, authentication and confidentiality. To this aim relevant properties of IT artifacts (e.g., authorization type applied), their interactions (e.g., transport protocols), as well as dedicated concepts (e.g., firewall) and their configuration, should be accounted for.

2.2.2 Maintainability

Purpose: Maintainability “is the capability of the software product to be modified” (ISO 2011) or “the ease with which it can be modified to changes in the environment, requirements or functional specification” (Bengtsson et al. 2004). Maintainability is considered to be important for IT infrastructure in an organization. The evaluation of maintainability may deliver important insights into IT infrastructure changeability, which facilitates the choice of more appropriate versions of architecture, or may reveal its flaws before commencing the development. In a general sense modifications to software systems can include corrections, improvements or adaptation of the software, but they also include such activities as installation of updates and upgrades.

Maintainability analysis requires architectural information that allow to assess the effort required to conduct a change (modification), but also to analyze the impact of the change (i.e., identifying the architectural elements, directly and indirectly, affected by a change). Note here that we focus on *estimation* of “maintainability”, i.e., we assess any indicators related to it (such as aforementioned effort, or impact) as it is may be estimated by an analyst/practitioner. A fully-fledged maintainability analysis in turn, requires information that cannot be delivered by ITML diagrams alone.

2 Main Analysis Scenarios

Exemplary questions of interest:

- Is the source code available?
- Is documentation available?
- What is the current version of the system and when was the last update installed?
- What language has been used for implementation purposes?
- What are relationships with the other elements of IT infrastructure, e.g., what other IT elements are using the considered IT artifact? What IT elements are used by the considered IT artifact?
- What is the estimated effort of conducting a modification of a piece of software?

Key concepts: properties of concept *Software* such as estimated effort, source code availability, documentation availability; relationships between components being part of a software artifact, implementation language used

Requirements:

R10: The IT infrastructure modeling language should account for supporting basic analysis of maintainability of the IT infrastructure. Thus, it should account for architectural information as well as dependencies among elements of IT landscape allowing to analyze the impact of a modification.

2.2.3 Portability Analysis

Portability, according to ISO, encompasses adaptability, installability and replaceability, and may be defined as “degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another” (ISO 2011). Please note that at the core, the assessment of portability requires conducting analysis of existing (constraining) dependencies among IT artifacts, as well as their constraining characteristics (e.g., supported data formats, supported system versions). Please also note that the analysis of replaceability may reduce lock-in risk.

Exemplary questions of interest:

- Which types of hardware and software environment is a given software artifact suited for?
- What standards, e.g., when it comes to standardized file formats, are supported by a given software artifact?

- Does the new version of the system has any new dependencies/requirements compared to the previous one?
- What are the constraining characteristics of some IT artifact, e.g., a maximal number of users, maximal data storage supported?

Key concepts: a set of dedicated relationships: runs on, suited for, requires, a set of dedicated attributes: scalability, standards followed, etc.

Requirements:

R11: The IT infrastructure modeling language should account for additional relationships reflecting various dependencies between IT artifacts supporting a portability analysis.

R12: The IT infrastructure modeling language should account for characteristics of IT artifacts supporting portability analysis (e.g., constraining characteristics, usage of standards).

2.2.4 Performance Analysis

Purpose: Performance analysis focuses on what is the desired, average, as well as actual performance of various IT infrastructure elements, with the aim to answer a question, whether the performance offered is satisfactory from a business perspective.

Please note that, like security and maintainability related analysis, in our analysis questions on performance we focus on a simplified notion of *estimated* performance only. The reason for doing so is similar to that discussed previously. On the one hand, performance related scenarios are important, especially when considering IT in conjunction with the organizational action system using it (e.g., a business process “insurance claim processing” relying on the throughput of a “claim processing” software application). On the other hand, an in-depth performance analysis is a complicated issue, which requires dedicated languages, such as the Architectural Analysis and Design Language (AADL), which lists performance analysis as one of its key analysis of the quality of performance-critical systems (Feiler, Gluch, and Hudak 2006, p. 5). Such dedicated performance analyses however, are out of our scope.

Exemplary questions of interest:

- What is the estimated average performance of some type of an IT landscape element (e.g., a hardware platform type, software artifact type)?,
- What is the actual performance of some specific IT landscape element?

Key concepts: estimated performance of software and hardware artifacts.

2 Main Analysis Scenarios

Requirements:

R13: The IT infrastructure modeling language should express estimated performance of IT landscape elements, whereby a specification of performance depends on the type of IT landscape element.

2.2.5 Availability Analysis

ISO (2011) considers availability as part of artifact reliability. In the following, we focus on the technical availability only.

Purpose: The availability analysis focuses on such basic questions like “*what is the average and actual availability of various elements of IT infrastructure*”? Similar to indicators for our analyses discussed earlier, being aware of the complexity of the availability analysis and its operationalization for different IT artifacts², we focus on the simplified notion of estimated availability only.

Exemplary questions of interest:

- What is the estimated average availability of some type of an IT landscape element (e.g., a hardware platform type)?
- What is the actual availability of some specific IT landscape element (e.g., an instance of hardware platform)?

Key concepts: Properties of software and hardware artifacts.

Requirements:

R14: The IT infrastructure modeling language should express (technical) availability characteristics of IT landscape elements.

2.2.6 Analysis of Vendors and Existing Relationships

Purpose: It is important to have an overview of all vendors and what they deliver to our organization with an eye to, e.g., vendor lock-in. Related analyses entail the identification of vendors of all IT landscape elements, analysis of existing contracts and support agreements, etc. It should be possible to assess vendor risks, as well as vendor power.

²Consider the difference in availability of storage devices, network storage, and a network generally. For storage devices, availability is influenced by the use of a (hot swappable) array of disks, whereas for network storage, the used data replication mechanism counts. Availability of a network is different still, since it is substantially influenced by the underlying network topology.

Exemplary questions of interest:

- With how many different vendors are we collaborating?
- To what extent are we dependent on some vendor?
- How critical to our business are products and solutions offered by some vendor? What risks can be identified, in case the vendor will get out of the market, or stop offering product support?

Key concepts: vendor and its characteristics, contract, agreement, provided IT landscape elements

Requirements:

R15: The IT infrastructure modeling language should provide concepts delivering information required for the needs of conducting a basic vendor analysis.

2.3 Integration Analysis

Integration is a pivotal concern when it comes to the analysis of IT infrastructures. The benefits of integration are significant, cf. also (Kattenstroth, Frank, and Heise 2013; Luftman, Zadeh, et al. 2012). In this section we focus specifically on two aspects (1) data integration and (2) functional integration. As an excursus, we also briefly discuss selected conflicts.

Data integration

Purpose: Integration requires the ability of two or more systems to exchange information, and use the information exchanged. Based on the modeled IT landscape one should be able to conduct a manual analysis in order to assess the level of integration, by focusing on the communication between various software artifacts and their properties. This includes types of data objects exchanged, exchanged file format document used, information covered, as well as characteristics of the software artifacts (e.g., implementation languages). Information should be also provided regarding whether, e.g., middleware, is necessary to enable the communication between different artifacts.

Note here again that “integration” is a complicated issue, touching not only on technical issues like the used implementation language, but also on many topics that are intricate and contingent in their own right, like formal and material semantics, or the notions of data and information. The information provided by the ITML diagrams should support manual analysis and provide some hints regarding additional integration possibilities.

2 Main Analysis Scenarios

Exemplary questions of interest:

- Which software artifacts communicate and what are the characteristics of this communication?
- What is the type of data being exchanged, e.g., in terms of “primitive” data types (such as byte, string, or boolean), data structures, and classes?
- Which data structures are used in the communication?
- Which middleware types are used in the organization, and what other elements they include?
- What data is read and written by various software artifacts? Is there any hint to investigate the potential for further integration?
- What data is covered by existing databases? What software artifacts use specific databases?
- Are two application systems using the same or similar data? Do they share a common database/repository?
- Are two applications based on the same data model?
- Are two applications using databases having the same schema?

Key concepts: data, middleware, implementation language, file format, wraps, has connector to, accesses data, reads data, writes data, data storage

Requirements:

R16: The IT infrastructure modeling language should explicitly account for data and its formats, as well as implementation languages of different IT infrastructure elements.

R17: The IT infrastructure modeling language should distinguish different types of middleware and corresponding relationships to other IT artifacts.

R18: The IT infrastructure modeling language should explicitly account for abstractions of the used data structures, allowing for classification of processed data.

Functional integration

Purpose: In the analysis of functional integration, functions provided by different IT artifacts are considered together with their data sharing interfaces. One is interested in identifying similar functions offered by different artifacts in order to identify potential redundancies and assess the potential for integration. Note here that our analysis questions regarding functional integration, and its according conceptualization (e.g., in terms of the concept function topic, cf. Chapter 3),

can be foreseen to be elaborated in terms of dedicated additional concepts and according diagrams. For example, it may be used to combine functional notions as contained in a Data Flow Diagram, with information and action system notions, such as software applications provisioning certain functions. Such an “enriched Data Flow Diagram” is however out of the scope of the current research report.

Exemplary questions of interest:

- What functions are provided by which artifacts?
- What interfaces are provided by a software artifact?
- Where which functions are provided and used?
- Which artifacts provide similar functions?
- Which artifacts provide some specific function?

Key concepts: function, provides function, uses function, interface

Requirements:

R19: The IT infrastructure modeling language should explicitly account for functions covered by IT artifacts belonging to the IT landscape. It should be possible to relate software artifacts to functions that they cover. Finally, it should be possible to state assessed similarity between functions.

R20: The IT infrastructure modeling language should explicitly account for interfaces and functionalities offered by IT artifacts.

Integration and reuse

As an noteworthy excursus, it is worth pointing out some design conflicts that should be considered here. Consider for instance two objects/components, which share a common semantic reference system. Firstly, the lower the range of interpretation of the concepts in this common semantic reference system, the higher is the level of integration between the two components. For example, it could be that both objects/components share the class “Printer” with specific attributes like “printingSpeed”. When one object/component communicates about a specific printer, the class “Printer” lowers the range of interpretation compared to something as generic as an “Object”. At the same time, a higher integration comes at the cost of a lower range of reuse. Namely, the lower range of interpretation inherent to higher integration also - by definition - excludes interpretations that - as a systems analyst - one at times might want to include, e.g., in the communication between two objects/components. In the most extreme case, this can be achieved by labeling everything as an “Object”, which has a very high range of reuse. However, due to its low level of information content “Object” can

2 Main Analysis Scenarios

hardly be considered to foster a high level of integration. As such, while not in the focus of this research report, finding an appropriate balance between reuse and integration is a key task for an information systems analyst. Therefore an IT infrastructure modeling language should allow representing this conflict, e.g., by expressing levels of semantic on an ordinal scale. For further discussion, cf. (Frank 2014a).

2.4 Integrated IT Infrastructure and Action System Analysis

In this section, we mention a few exemplary analysis scenarios that become possible once an IT infrastructure modeling language is part of an enterprise modeling approach, i.e., IT landscape may be analyzed in the context of enterprise action system (e.g., business processes, organizational structure, or defined goals). Thus, to support integrated analysis scenarios mentioned subsequently, an IT modeling language needs to be integrated with other modeling languages, and all involved languages need to be used in tandem.

2.4.1 Analysis of IT-Business Alignment

Purpose: IT-business alignment focuses on how effectively the IT landscape supports and enables the business strategy of a company. Via the analysis of IT Infrastructure models, in tandem with models which express a different perspective on the enterprise action system (such as business processes), it should be possible to (1) assess and, if necessary improve, the alignment of IT and business to ensure that they work as a partnership and support enterprise business processes; (2) conduct analysis of the IT environment for strategic planning (e.g., by determining which elements of IT infrastructure support the fulfillment of enterprise's goals).

In order to assess the IT-business alignment, it is necessary to define (1) which elements of IT infrastructure support directly business processes, as well as the properties of such a support. Such properties include relevance for a process, or frequency of usage; (2) which elements of IT infrastructure support which enterprise's goals, as well as properties of such a support.

Subsequently, knowing which services/applications are critical (to both business processes as well as defined goals) allows to investigate the physical IT infrastructure underneath, in terms of the (characteristics of) used operating systems, computational infrastructure, and storage infrastructure.

Exemplary questions of interest:

- Which applications are critical for the core business?

- What is the contribution to the strategical goals of an organization that each application make?
- Which business processes are supported by which application?
- Which business units use which applications for which business processes?
- What are the characteristics of the support provided by some applications/services to business processes and their elements? Which processes in the organization depend highly on the IT infrastructure?
- Which organizational actors would need to be considered for additional software training, if we change the application software used to support some process?

Key concepts: IT service, software artifact, support relation, business process, organizational unit

Requirements:

R21: The IT infrastructure modeling language should allow to link elements of IT landscape and processes they support. The characteristics of the support provided should be accounted for.

R22: The IT infrastructure modeling language should allow to relate IT landscape elements to goals of organization, in terms of the influence of these elements on goal achievement. The characteristics of the influence provided should be accounted for.

As a brief excursus note that, especially with analysis which cut across different concerns, the need for a modeling language again becomes apparent. In this case, to get stakeholders coming from different professional backgrounds on the same page, a visual modeling language as an instrument for gaining consensus would be called for.

2.4.2 Analysis of Organizational Assignment

Purpose: The organizational assignment should be clearly communicated. It should be also possible to point to existing responsibilities assignment for each IT artifacts. When it comes to specifying the “who”, various organizational structure elements can be meant, such as a particular organizational role (e.g., an IT administrator), or various kinds of organizational units (such as a department). By analyzing the organizational assignment, we are not only able to answer the questions regarding some specific responsibilities (e.g., responsible, accountable, consulted and informed (RACI), cf. also (Feltus, Petit, and Dubois 2009)), but also check whether each IT artifact type has someone responsible assigned, whether the appropriate organizational units are assigned, whether there is some optimization potential (e.g., similar

2 Main Analysis Scenarios

applications are assigned to different organizational units, or applications written using Java are assigned to a C# Programmer). It is also possible to identify problems resulting out of the incorrect division of responsibilities (e.g., different organizational units involved in the maintenance process). In addition, one can check the workload of, e.g., some specific roles.

Exemplary questions of interest:

- Which organizational unit is assigned, e.g., as a responsible or accountable for an element of IT landscape of interest?
- Do all elements of the IT landscape have a responsible person assigned?

Please note that the consideration of the location, performance, availability, as well as organizational assignment may trigger analysis of *consolidation opportunities*, which are there regarding the shared use of infrastructure, taking into account aspects such as performance, security and maintenance windows.

Key concepts: Organizational unit, software artifact, hardware platform, responsibility relation

Requirements:

R23: The IT infrastructure modeling language should allow to link elements of IT landscape with organizational units and express the role of linked organizational units.

3 Language Design: Abstract Syntax and Semantics

The analysis of the requirements performed in the previous section clearly shows that a language for modeling IT infrastructures requires a careful reconstruction of technical concepts from the field of IT management. Those concepts should be, on the one hand, (1) general enough to account for all relevant IT infrastructure elements, and, on the other hand, (2) specific enough, to support differentiated analyses, necessary to answer the questions mentioned in the previous section. Therefore, in line with the identified requirements, on the one hand, the ITML should provide semantically rich concepts reconstructing the technical terminology of the IT domain, and on the other hand, the ITML's concepts should be applicable to a wide range of enterprise settings and over a longer period of time, therefore, they should be generic. The reconstruction should account for different perspectives and granularity levels, as well as consider concepts from the business domain.

The concepts offered with the ITML are designed to support IT management with relevant analyses. The corresponding analysis questions however, cannot always be answered using a model of the IT infrastructure only. Sometimes it is necessary to include models of the action system, specific models of the software used, or acquire additional data from within or outside of the organization in question.

Taking the above into consideration, in what follows, we discuss the abstract syntax and semantics of the designed modeling language. Note that the borderline between abstract syntax and semantics is blurred. Specific aspects of a language can often be specified with its abstract syntax or, alternatively, with its semantics. For the sake of simplification, we assume that the abstract syntax is defined through the part of a meta model that corresponds to its graphical representation in a diagram, while additional semantics is defined through constraints, e.g., Object Constraint Language (OCL) constraints. In addition to these formal characteristics of a language, conceptual modeling also requires accounting for material semantics or the meaning contributed to a concept by humans. The meaning of concepts found in the domain of IT management is relevant for two reasons. First, it serves us as a starting point for the reconstruction of domain languages through the ITML. Second, the concepts of the ITML themselves need to be explained to prospective users, which recommends referring to corresponding domain-specific concepts.

As we create the ITML with an intention to be used with other MEMO languages, its abstract syntax is defined using the MEMO method's common Meta Modeling Language (MML)

(Frank 2011) to foster the DSML's integration into MEMO method's language architecture (Frank 2014b, pp. 947-950). The MML in addition to the other concepts commonly used in meta modeling, provides the possibility to model intrinsic features¹ and intrinsic relations. This intrinsicness, marked by the literal "i" in white color on a black background, allows for modeling concepts, attributes and relations that may be instantiated only on the instance level, and not on the type level. In addition, the MML allows for modeling so called Language Level Types – visualized with a black name of the concept on a grey background – which allow for specifying concepts that represent instances already on the model level (M_1), and cannot be instantiated on the instance level (M_0) anymore (Frank 2011, pp. 23-24).

In the following, we present and discuss excerpts of an ITML meta model. We point to our understanding of selected concepts, as well as analysis possibilities resulting out of the proposed conceptualization.

While looking at the presented meta model's excerpts, please note the following:

- Defined attributes and relationships encompass both type-level as well as instance-level aspects (intrinsic attributes and relationships). As visible in the requirements identified in the previous sections (cf. especially R1), IT management focuses on both individual IT artifacts as well as their types, depending on the goals of the analysis;
- The meta types could be described in a more differentiated manner. For instance, in the case of mobile computers the size and resolution of the screen could be taken into account. We have refrained from such details in order to maintain the clarity of the diagrams;
- As various meta types may appear multiple times in different meta model excerpts, their attributes are presented only within one excerpt, and in other visualizations replaced by "...";
- The relevant integrity constraints are formulated using the Object Constraint Language (OCL), whenever possible. Please note however, that one is not able to use OCL to formulate constraints that relate to the instance level (i.e., that relate to intrinsic attributes and/or relationships). This would be however necessary to ensure integrity on the instance level. For instance, a specific central processing unit is mounted on some specific hardware platform, but a type of CPU fits one or more types of hardware platforms. Therefore, a constraint would have to be specified that ensures that some specific CPU is mounted on only specific hardware platforms, whose types the CPU type fits. In those cases, we formulate the constraints in the natural language only. All constraints may be found in Table 3.7.

¹Intrinsic features "can be instantiated only from the instances of their instances", i.e., a meta type or meta attribute can be instantiated only on M_0 , but not on the M_1 level (Frank 2011).

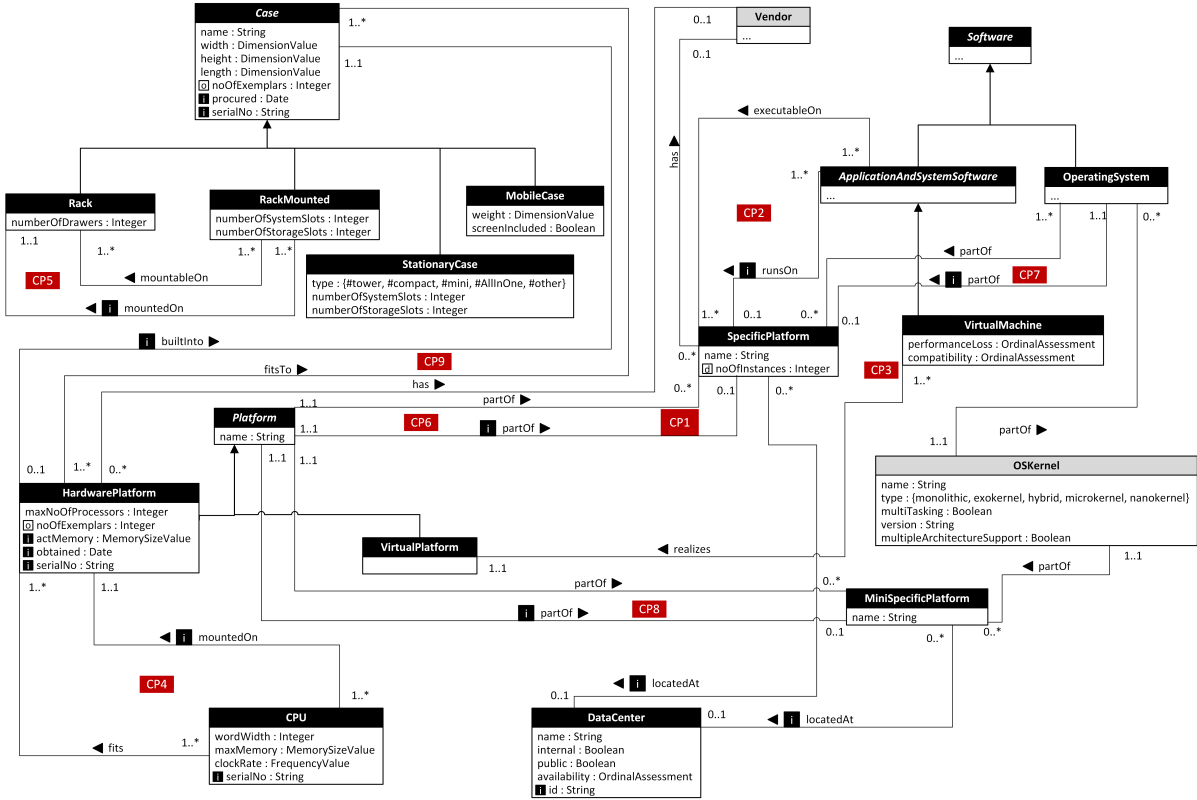


Figure 3.1: Meta Model Excerpt: Platform, for constraints, cf. Tab. 3.7

3.1 Main IT Landscape Elements

In what follows, we provide an overview of the IT landscape elements whose analyses are supported by the ITML, with a focus on the scenarios introduced in Section 2.1.1–2.1.3.

Platform. Computer hardware, whose resources are made accessible via operating systems, constitutes a basic element of an IT infrastructure. To represent it, we define a concept ‘specific platform’, cf. Fig. 3.1. In a simplified representation, a specific platform (cf. `SpecificPlatform`) consists of a hardware platform, encompassing among others a processor along with main memory, and an operating system. In addition to physical platforms, there are also virtual platforms, i.e., those that are simulated by software. Thus, we differentiate between a physical platform (cf. `HardwarePlatform`), provided in some case, and a virtual one (cf. `VirtualPlatform`, Fig. 3.1), which can be realized by a virtual machine. A virtual machine is conceptualized in the ITML as a specialization of an abstract meta type `Software`, discussed further in this section, cf. Figs. 3.3–3.5.

An `OperatingSystem` is a specialization of the abstract meta type `Software`. Following Tanenbaum and Bos (2014), operating systems, on the one hand, manage the different parts

of the system efficiently, i.e., provide “a clean abstract set of resources instead of the messy hardware ones and managing these hardware resources” (p. 4), on the other hand, the job of the system is “to provide the users with abstractions that are more convenient to use than the actual machine” (p. 80) (e.g., processes, files). An operating system has an operating system kernel (modeled as a language level type, `OSKernel`), responsible for managing communication between hardware and software, memory, cache, hard drive, etc. Kernels vary widely in function and scope, and significantly affect their operating system’s capability. Operating systems come in different types (e.g., embedded, real-time, multi-user), and have various features. For details regarding properties of an operating system, cf. Figure 3.3.

Table 3.1: *Language specification: Comments on **Hardware Platform** and related concepts, cf. Fig. 3.1*

Concept: Hardware Platform	Description: Hardware Platform, next to Virtual Platform, is a basic element of an IT infrastructure. Resources offered by a hardware platform are made accessible via an operating system. It consists, among others, of a central processing unit along with a main memory.	
Supertypes		
Platform		
Attributes on the type level		
name	String	Allows for stating the name of the hardware platform.
maxNoOfProcessors	Integer	Allows for stating the maximal number of processors supported.
noOfExemplars	Integer	It is an obtainable attribute allowing to state how many exemplars of this specific hardware platform type there are.
Attributes with reference to the instance level		
actMemory	MemorySizeValue	Allows for stating the size of the memory (e.g., 1 GB).
obtained	Date	Allows for stating when the specific exemplar of a hardware platform has been obtained.
serialNo	String	Allows for stating the serial number of a hardware platform instance.
Selected Associations		
fits and [i] mountedOn (source: CPU, target: HardwarePlatform)	1,* – 1,* (fits) 1,* – 1,1 (mountedOn)	Allows to state what type of CPU fits to a specific type of a hardware platform as well as, on the instance level, what instance of a CPU has been mounted on some specific exemplar of a hardware platform.
fitsTo and [i] builtInto (source: HardwarePlatform, target: Case)	1,* – 1,* (fitsTo) 0,1 – 1,1 (builtInto)	Allows to state what case a specific type of a hardware platform fitsTo (on a type level), as well as, on the instance level, what case some specific exemplar of a hardware platform has been built into.

Sometimes a specific platform does not contain a complete operating system, but only the operating system kernel (OSKernel), which handles input/output, memory management etc., cf. (Tanenbaum and Bos 2014). Such a platform is shown in the meta model as `MinSpecificPlatform`. This concept is important in connection with the installation and deployment of software, cf. Fig. 3.8.

Table 3.2: *Language specification: Comments on **Operating System Kernel**, cf. Fig. 3.1*

Concept: OSKernel	Description: Language Level Type. Kernel is a core feature of any operating system. It manages communication between hardware and software, memory, cache, hard drive etc. Kernels vary widely in function and scope and significantly affect their operating system’s capability.	
Attributes		
name	String	Allows for stating the name of the kernel.
version	String	Allows for stating the version of the kernel.
type	Enumeration	Allows for stating the type of the kernel, such as: mono-lithic, hybrid, microkernel, nanokernel, exokernel.
multiTasking	Boolean	Allows for stating whether the kernel supports multitask-ing.
multipleArchi- tectureSupport	Enumeration	Allows for stating whether the kernel supports multiple CPU instruction sets and micro architectures.
Selected Associations		
partOf (source: OSKernel; target: OperatingSystem)	1,1 – 0,*	Allows to state that a specific kernel is part of an operating system.
partOf (source: OSKernel; target: MiniSpecific-Platform)	1,1 – 0,*	Allows to state that a specific kernel is part of a mini spe-cific platform.

With respect to the relation between software and platforms, it is important to distinguish two cases. First, software may be executable on some kind of platform type. Second, software may be executed on a certain platform. The first case is represented by the association `executableOn` between `ApplicationAndSystemSoftware` (being a specialization of `Software`, cf. also Fig. 3.5) and some `SpecificPlatform`. The second one is represented by the intrinsic, i.e., instantiated at M_0 , association `runsOn`, between `ApplicationAndSystemSoftware` and some `SpecificPlatform`.

A physical platform can take different forms. It can be mounted in a case (Case) that is intended for either stationary or mobile use. Such physical platforms are often also mounted in racks, each of which has a certain number of slots, cf. Fig. 3.1. Physical components of a

hardware platform (cf. `HardwarePlatform`), such as special case types, are only relevant, if the corresponding devices are managed in the company. If platforms are located in a separately operated data center, this physical element (like the case) can be abstracted away.

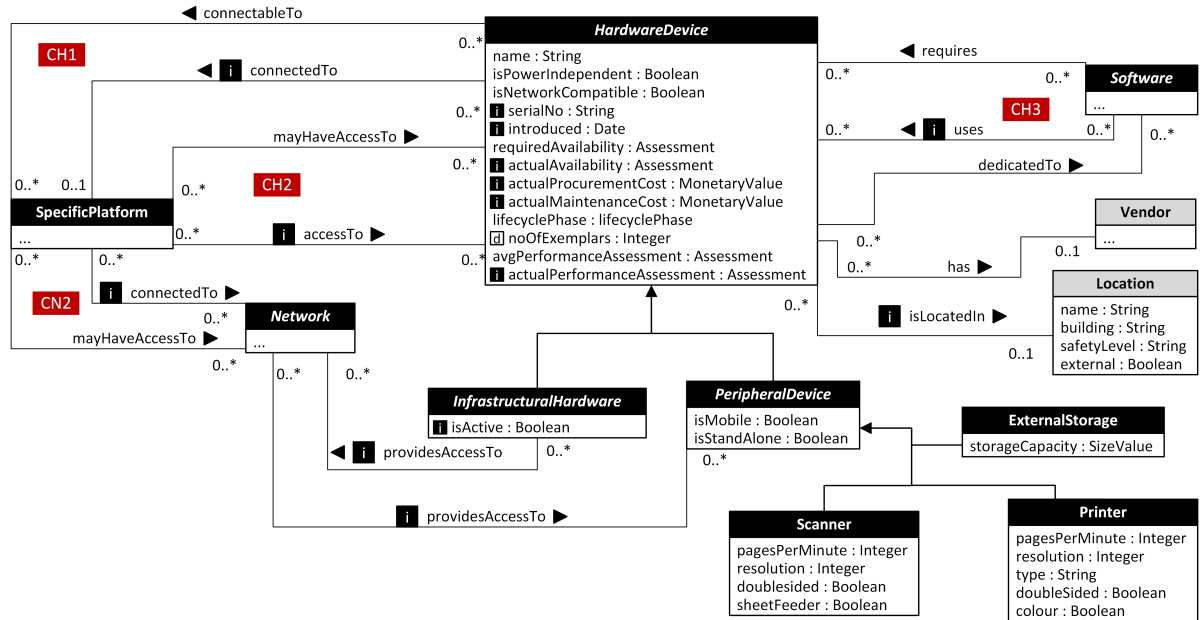


Figure 3.2: Meta Model Excerpt: Hardware, for constraints, cf. Tab. 3.7

Peripheral Device and Infrastructural Hardware. To support the variety of analysis scenarios from Chapter 2, one can distinguish between different types of hardware devices according to their primary purpose, architecture, or a role to a user. From a modeling perspective such differentiation can be expressed by using relevant meta types, by using generalization/specialization, or by using an enumeration attribute of the generic meta type.

The excerpt of the meta model in Fig. 3.2 shows a possible conceptualization of a hardware device. Traditionally a hardware device has some `Location` (a language level type), is connectable to some `SpecificPlatform` or may be accessed to by it. A software artifact may require some hardware device, and actually use some device on the instance level (cf. an intrinsic relationship `uses`).

We specifically differentiate between `PeripheralDevice` (with such specializations as, e.g., `Printer` or `External Storage`) and `InfrastructuralHardware` (with such possible specializations as, e.g., `Access Point` or `Router`). Please note that each meta type has a set of dedicated attributes and specific relations. And so, `PeripheralDevice` can be accessible via a `Network` or be accessible only via some specific `SpecificPlatform`: a `PeripheralDevice` may be connectableTo (on the type level) or connectedTo (on

the instance level) to some `SpecificPlatform`. `InfrastructuralHardware` may in turn provide an access to a `Network`.

Table 3.3: *Language specification: Comments on **Hardware Device**, cf. Fig. 3.2*

Concept: Hardware Device	Description: a hardware device which a platform can interact with, and which typically is not a core part of the platform.	
Subtypes		
Peripheral device, Infrastructural device		
Attributes on the type level		
name	String	Allows to provide a name.
isPower-Independent	Boolean	Allows to state if a separate power source is necessary.
isNetworkCompatible	Boolean	Indicates compatibility with a given network.
reqAvailability	OrdinalAssessment	Allows to express the required availability of a hardware device.
lifecyclePhase	LifecyclePhase	Indicates the life-cycle phase of a hardware device, ranging from acquisition, operation, to disposal.
noOfExemplars	Integer	Used to track the number of hardware devices.
avgPerformance-Assessment	OrdinalAssessment	Expresses the estimated average performance.
Attributes with reference to the instance level		
serialNo	String	Allows to indicate the unique serial number.
introduced	Date	Expresses the date a hardware device has been introduced.
actAvailability	OrdinalAssessment	Allows for expressing the actual availability, which in an analysis can be compared to the required availability (in support of the availability analysis scenarios from Chapter 2).
actProcurement-Cost	MonetaryValue	Can be used to specify the upfront investment for acquiring a hardware device.
actMaintenance-Cost	MonetaryValue	Allows for specifying the maintenance cost of a hardware device.
actPerformance-Assessment	OrdinalAssessment	Expresses the actual performance of a hardware device (forming input to the attribute average performance on the type level).
Selected Associations		
conectableTo	0,* – 0,*	with SpecificPlatform.
[i] connectedTo	0,* – 0,1	with SpecificPlatform.
mayHaveAccessTo	0,* – 0,*	from SpecificPlatform.
[i] accessTo	0,* – 0,*	from SpecificPlatform.
Constraints		
CH1	A hardware device may be connected to a specific platform being an instance of a type of a specific platform that a hardware device in question is connectable to.	
CH2	A specific platform may access a hardware device being an instance of a type that a platform may have access to.	

Continued on next page

3 Language Design: Abstract Syntax and Semantics

Table 3.3 – Continued from previous page

CH3	A software uses a hardware device being an instance of a type that it requires.
CN2	A specific platform is connected to a network being an instance of a type that the given specific platform may have access to, cf. Fig. 3.7

Table 3.4: Language specification: Comments on *Peripheral Device*, cf. Fig. 3.2

Concept: Peripheral Device	Description: auxiliary device which a platform can interact with, and which typically is not a core part of the platform.	
Supertypes		
HardwareDevice		
Subtypes		
Printer, Scanner, ExternalStorage		
Attributes on the type level		
isMobile	Boolean	Indicates how mobile the peripheral device is (e.g., in terms of size, or mass).
isStandAlone	Boolean	Indicates the autonomy of a peripheral device, e.g., the need for additional power bricks for external hard drives.
Attributes with reference to the instance level		
isActive	Boolean	As implied by its name, this attribute indicates the activity status of the peripheral device.
Selected Associations		
providesAccessTo (from Network to PerhiperalDevice)	0,* – 0,*	Allows to express that a network can provide access to a peripheral device.

Software. Software is ubiquitous and, at first glance, one may have a clear idea of what constitutes software. Whereas hardware is usually considered to consist of tangible objects (e.g., integrated circuits, circuit boards, cables, powers supplies, memories, card readers), software consists of algorithms (detailed instructions on how to do something) and their computer representations, cf. (Tanenbaum 2006, p. 9). However, on closer inspection the conceptualization of software is by no means trivial.

The excerpt of the meta model in Fig. 3.3 shows a possible conceptualization of software. Here a particular software is represented as a type that is instantiated from one of the concrete subclasses of the abstract meta type *Software*, implemented using some *ProgrammingLanguage*. An *Installation* is required to use software. The software is installed from its file representation, whereby different forms of representation have to

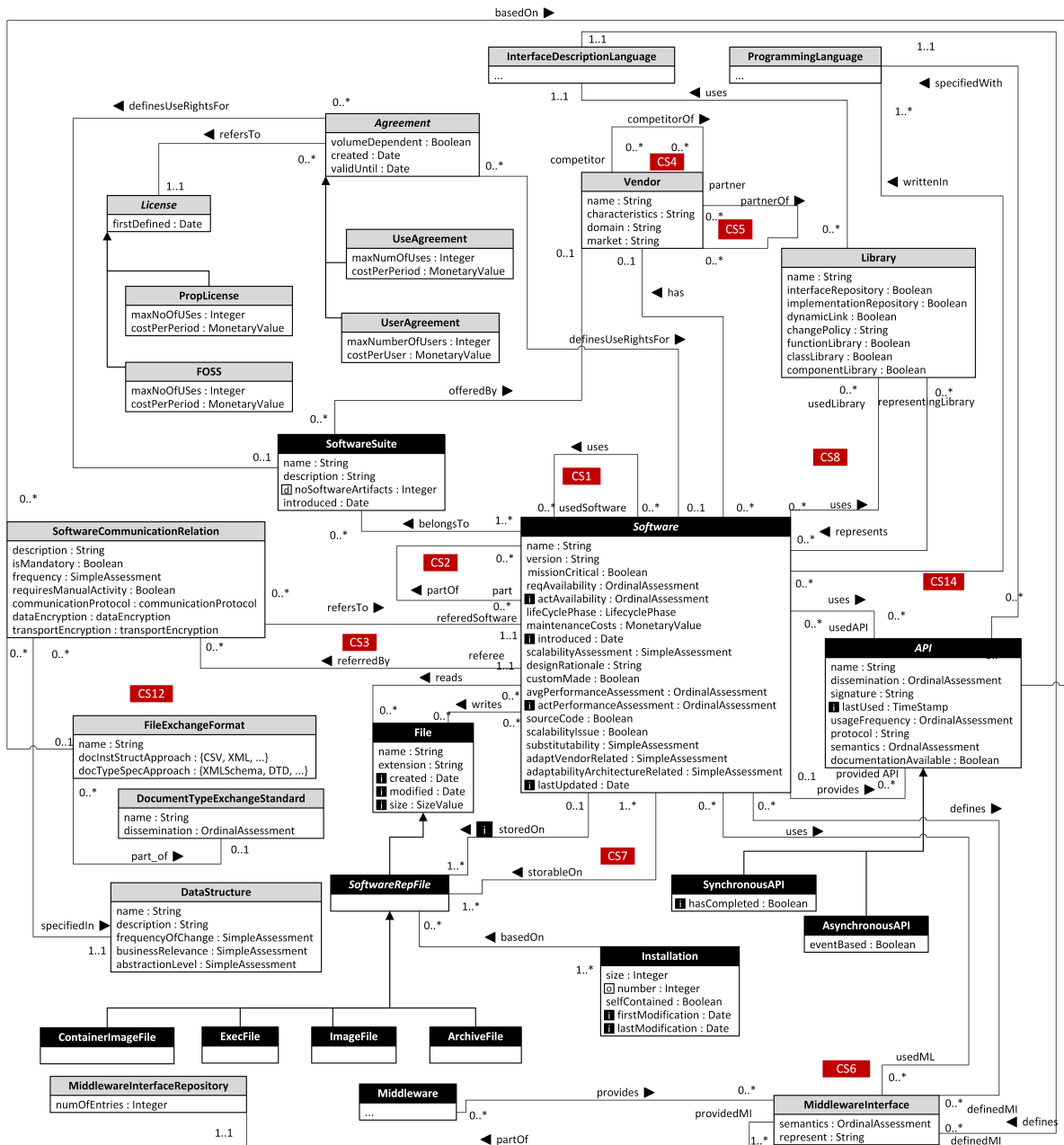


Figure 3.3: *Meta Model Excerpt: Software, for constraints, cf. Tab. 3.7*

be taken into account. In addition to executable files (`ExecFile`), special archive files (`ArchiveFile`), such as JAR files for Java programs, should be considered. There are also image files (`ImageFile`) that contain a binary, machine independent, but not directly executable, representation of a program. Finally, container images (`ContainerImage`) are files which, in addition to the actual software, also contain the resources required by the software, so that their execution by a container engine (`ContainerEngine`) requires only a minimal specific platform, cf. also the deployment excerpt of a meta model, Fig. 3.8.

A `Software` is typically installed on some file management system (cf. `FileSystem`). Software is then represented as one or more files in a form that allows for its execution. Alternatively, the installation can take place in a `DataCenter`. In the latter case, one can abstract away from the `FileSystem` a `Software` has been installed on, since this is the responsibility of the `DataCenter`. When a data center hardware and software is used to make (application) software accessible over interfaces, we speak of a so-called “cloud”.

Whereas one could introduce here common categories of cloud services, such as *Software as a Service* (SaaS), *Infrastructure as a Service* (IaaS), and *Platform as a Service* (PaaS), this would however considerably increase the complexity of analysis scenarios considered. Therefore, although modeling of “cloud” aspects is certainly a problem area of its own, we decided to follow a pragmatic solution, and account for cloud-based elements via an attribute `inCloud`. This has the advantage that cloud-based application systems can be modeled and classified in the IT infrastructure in the same way as conventional applications.

Software can have an interface for other programs, which can be used to access, or possibly modify, the functions it provides. Such an interface, also referred to as an API (“Application Programming Interface”), can be synchronous or asynchronous. Therefore, a `Software` provides and/or uses some API, which in turn is specialized into `Synchronous` and `Asynchronous` one. In order to depict dependencies between software, a communication relation, which may be based on API and use some exchange format, allows to specify how data is being exchanged. For instance, one may state whether there is a need for manual activity, or what is the frequency of communication. Furthermore, `DataStructure` allows for a high-level description of the exchanged data, a kind of common concepts, as well as expected frequency of change, business relevance and abstraction level. In addition, a `Software` may define (and also use) a `MiddlewareInterface`, which is defined using an `InterfaceDescriptionLanguage`, and is provided by some `Middleware`. `MiddlewareInterfaces` are stored in a `MiddlewareInterfaceRepository`.

Additional dependencies between software artifacts are represented by the relationships `requires` (`Software requires Software`), and `partOf` providing a possibility to model a system/complex software type offered as one product. Finally, a software artifact may be distributed (be part of) a software suite (cf. `Software belongsTo SoftwareSuite`).

The legal use of `Software` requires a `License`. To this end, one can, roughly speaking, distinguish two types of `License`. Proprietary licenses (`PropLicense` in Figure 3.3) are issued individually by software vendors. Open Source Licenses, also called FOSS (“Free and Open Source Software”), exist in various forms, from which we abstract in this research report.

`License Agreements` refer to a license type and establish the conditions of software use, like the maximum number of concurrent users. Such conditions are unusual in the case of FOSS, but they are still possible.

Table 3.5: *Language specification: Comments on **Software**, cf. Fig. 3.3*

Concept: Software	Description: A detailed set of instructions that tells a hardware (platform) how to carry out a certain task.	
Subtypes		
OperatingSystem, ApplicationAndSystemSoftware, Component, Webclient, Middleware		
Attributes on the type level		
name	String	Allows for specifying a name.
version	String	Allows for specifying a version.
missionCritical	Boolean	Indicates if the software is mission critical.
reqAvailability	OrdinalAssessment	Indicates the required availability of the software (cf. the analysis scenarios in Chapter 2).
maintenanceCosts	MonetaryValue	Indicates the maintenance costs of a software.
scalability-Assessment	SimpleAssessment	Can be used to express the estimated scalability of a software (cf. the analysis scenarios in Chapter 2).
designRationale	String	Can be used to denote the expressed rationale standing behind a software.
customMade	Boolean	Used to express if the software has an off-the-shelf character.
avgPerformance-Assessment	OrdinalAssessment	Used to express the average estimated performance of a software (in line with the analysis scenarios from Chapter 2).
sourceCode	Boolean	Indicates availability of a software’s source code.
scalabilityIssue	Boolean	Allows, in a rudimentary manner, to indicate if a scalability issue exists with a software.
substitutability	SimpleAssessment	Indicates the extent to which a software can be substituted.
adaptVendor-Related	SimpleAssessment	Indicates the adaptability as far as the vendor is concerned.
adaptArchitectureRelated	SimpleAssessment	Indicates the adaptability as far as the architecture is concerned.
Attributes with reference to the instance level		
actAvailability	OrdinalAssessment	Allows for expressing the actual availability (and thus also for a comparison to the required availability on the type level, cf. analysis scenarios in Chapter 2).
introduced	Date	Indicates the date on which a software has been introduced.
actPerformance-Assessment	OrdinalAssessment	Indicates the estimated actual performance of a software (in line with the analysis scenarios from Chapter 2).
Selected Associations		
partOf	1,* – 0,*	with Software; a software may be part of another software.
uses	0,* – 0,*	with API, Library, MiddlewareInterface, pointing that a software artifact uses an API, a Library or a MiddlewareInterface, respectively.
has	0,* – 0,1	with Vendor; allows to point to a vendor of a software artifact.

Continued on next page

3 Language Design: Abstract Syntax and Semantics

Table 3.5 – Continued from previous page

referredBy	1,1 – 0,*	with SoftwareCommunicationRelation; SoftwareCommunicationRelation allows to define software artifacts communicating, as well as characteristics of this communication.
provides	0,1 – 0,*	with API; allows to define APIs provided by a software artifact.
belongsTo	1,* – 0,*	with SoftwareSuite.
Constraints		
CS1	Software cannot use itself.	
CS2	Software cannot be part of itself.	
CS3	Software cannot communicate with itself.	
CS7	Software can be stored only on specific replication files that software, on the type level, is storable.	
CS8	Software cannot use a library that it represents.	

Software may use a Library. A library is a collection of non-volatile resources, such as implementations of functions or classes. A Library has a well-defined interface that allows for accessing its resources.

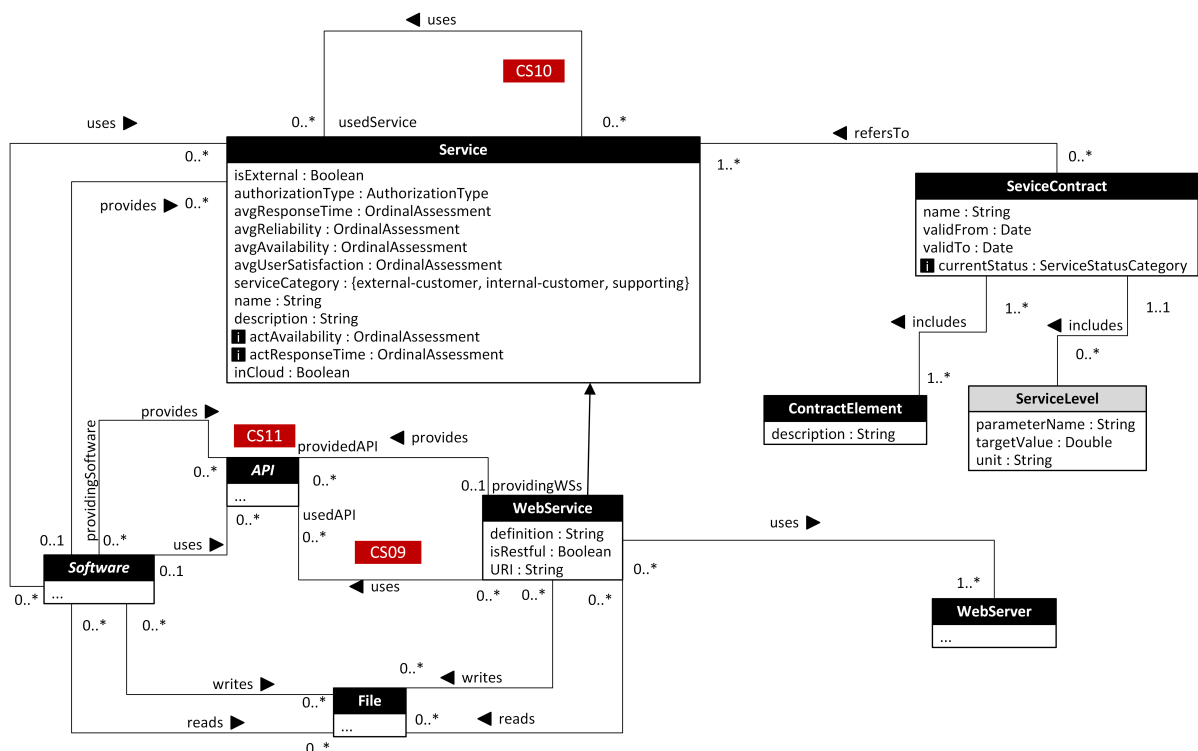


Figure 3.4: Meta Model Excerpt: IT-Service, for constraints, cf. Tab. 3.7

Functions of Software may use or provide some Services. A service, in turn, offers some function that can be invoked over the network. Please note that we consciously opt here for a

narrow service definition, which focuses on the technical – software – side only. In line with the notion of service-orientation (Papazoglou and Van Den Heuvel 2007, p. 390), the service notion abstracts away from *how* the function offered by a service is realized. Such a service is usually characterized by a specific interface that typically abstracts from a specific programming language and thus, also from its specific implementation. A subtype of a service is a `WebService`, which uses a `WebServer`, providing or using some API. The function offered by a service (expressed through, e.g., a service level) can be defined within a `ServiceContract`.

Table 3.6: Language specification: Comments on *Service*, cf. Fig. 3.4

Concept: Service	Description: An IT artifact, usually a piece of software, offering some function that can be invoked over the network. The function offered by a service and its characteristics may be defined within a service contract.	
Subtypes		
Web service		
Attributes on the type level		
name	String	Allows for specifying a name.
description	String	Allows for specifying a description.
isExternal	Boolean	Allows for expressing if a service is external to an organization.
authorizationType	AuthorizationType	Allows for expressing the type of authorization needed to access a service.
avgResponseTime	OrdinalAssessment	Allows for expressing the average response time.
avgReliability	OrdinalAssessment	Allows for expressing the average reliability.
avgAvailability	OrdinalAssessment	Allows for expressing the average availability (e.g., to support the analysis scenarios discussed in Chapter 2).
avgUserSatisfaction	OrdinalAssessment	Allows for expressing the average user satisfaction with a service.
serviceCategory	external-customer, internal-customer, supporting	Allows for expressing the service category.
inCloud	Boolean	Allows to state whether a service is offered as a cloud-based service.
Attributes with reference to the instance level		
actAvailability	OrdinalAssessment	Allows for expressing the actual availability of a service (it can be then compared to its average availability on the type level for analysis purposes).
actResponseTime	OrdinalAssessment	Allows for expressing a service’s actual response time (it can then be compared to its average response time on the type level for analysis purposes).
Selected Associations		
uses	0,* – 0,*	with oneself; a service may use another service.
uses (source: Software, target: Service)	0,* – 0,*	Points to what software uses the given service.

Continued on next page

Table 3.6 – Continued from previous page

provides (source: Software, target: Service)	0,1 – 0,*	Points to what software provides the given service.
refersTo (source: ServiceContract, target: Service)	0,* – 1,*	Allows to attach one or more service level contracts to a service.
Constraints		
CS10	A service cannot use itself.	

As already mentioned, a particular software is represented as a type instantiated from one of the subclasses of the abstract meta type *Software*. Please note that taking into account the identified analysis scenario, a classification of a generic concept like software seems to be necessary. However, software can be categorized with regard to its primary purpose, architecture (e.g., client/server), visibility to a user (e.g., back-end, front-end) or by looking at its role. From a modeling perspective such differentiation can be modeled in different ways: by using relevant meta types, using generalization/specialization, using an enumeration attribute of the generic meta type (e.g., type of software), or as a role. Taking into account identified requirements as well as analysis scenarios, we have decided to account for this mostly by using the specialization relation (particularly subtypes of *Software*). This is visible in Fig. 3.5, where the hierarchy of software artifacts, starting from abstract software concepts, through, e.g., application and system software, to application server, IT management tools and others, is presented. In addition, by taking advantage of enumerations, we have offered a possibility to assign an application software to a software category. Possible instances of software category may be, among others, Enterprise Software, Office Software.

Table 3.7: Language specification: Comments on *Server*, cf. Fig. 3.5

Concept: Server	Description: A Server provides resources and functionality to clients (other types of software), typically over a network as a response to a request from aforementioned clients.	
Supertypes		
Application and Technology Software, Software		
Subtypes		
Web server, Application Server, File Server, Database Management Server		
Attributes on the type level		
encrypted access	Boolean	Allows for expressing if the server access is encrypted.
Attributes with reference to the instance level		
address	String	Indicates the address at which a particular server can be accessed (e.g., an URL).

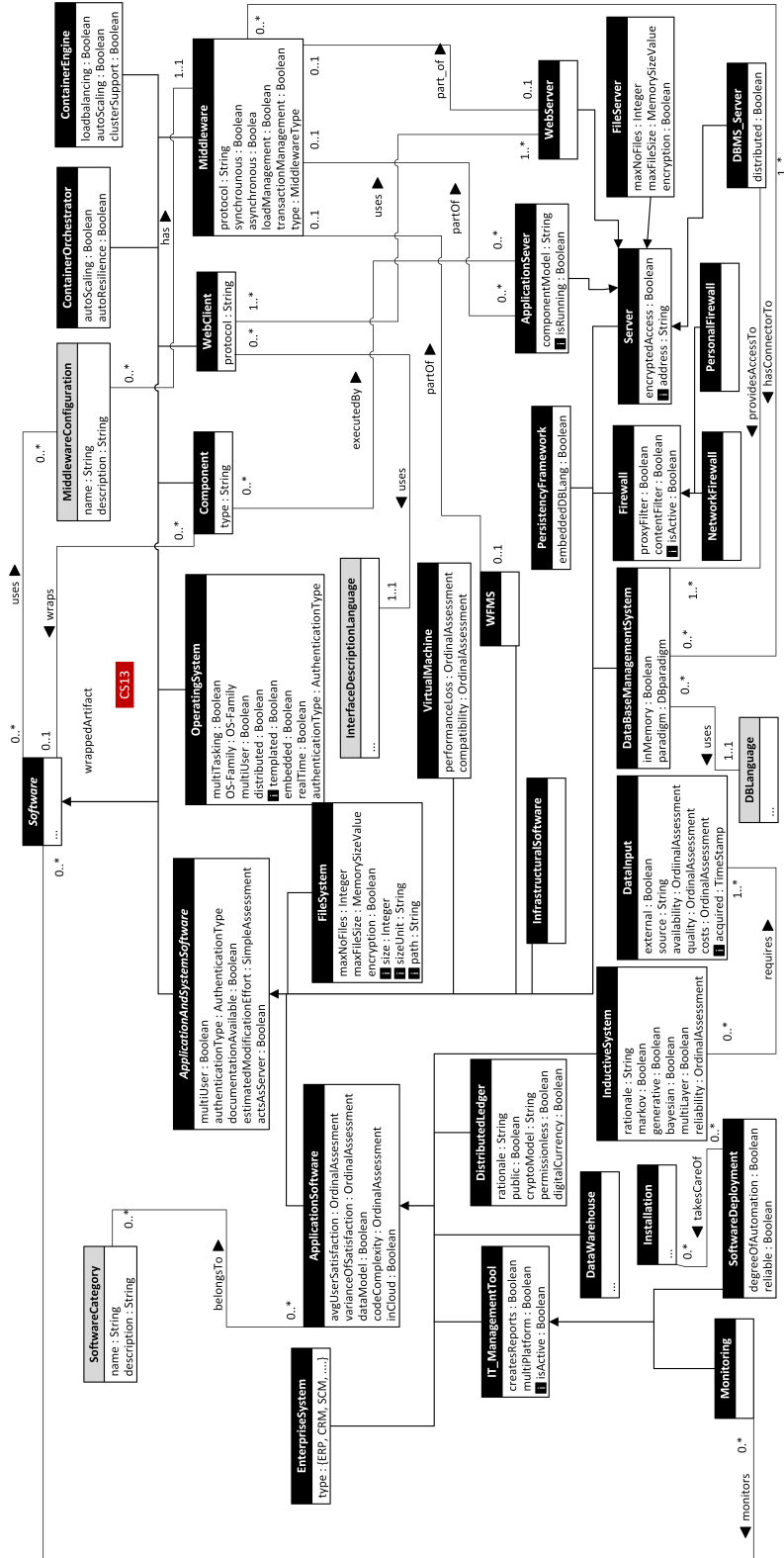


Figure 3.5: Meta Model Excerpt: Software Hierarchy, for constraints, cf. Tab. 3.7

Table 3.8: *Language specification: Comments on **Application Software**, cf. Fig. 3.5*

Concept: Application-Software	Description: Application Software provides a functionality to the end user, different from functionality required to run a system itself.	
Supertypes		
Application and Technology Software, Software		
Subtypes		
e.g., EnterpriseSystem, IT.ManagementTool, DistributedLedger, DataWarehouse, InductiveSystem		
Attributes on the type level		
avgUserSatisfaction	OrdinalAssessment	Allows for expressing the average user satisfaction.
varianceOfSatisfaction	OrdinalAssessment	Allows for expressing the variance of user satisfaction.
dataModel	Boolean	Allows for expressing if a data model exists for the Application Software.
codeComplexity	OrdinalAssessment	Allows for expressing the code complexity of the Application Software.
inCloud	Boolean	Allows for expressing whether the given application is cloud-based.
Selected Associations		
has	0,* – 0,*	with SoftwareCategory.

Please note that as part of the mentioned hierarchy of software concepts, in order to account for systems that make use of inductive approaches, we introduce the concept `InductiveSystem`. `InductiveSystem`, being an `ApplicationSoftware`, is an abstraction over software applications relying on machine learning approaches. Instead of programming the solution directly, a machine learning algorithm follows a data-based approach, and based on training data produces a solution program (called the learned model). Machine learning relies on well-established algorithms from mathematics and statistics, such as, e.g., regression, bayesian, or decision trees (Barber 2012; Deisenroth, Faisal, and Ong 2020).

Persistence. Figure 3.6 depicts our conceptualization of persistence. Here, two dedicated software systems play a notable role. A `FileSystem` manages persistent data in the form of files. `DatabaseManagementSystems` (DBMSs) manages data in the form of `Databases`, which are typically structured in accordance with a database schema. A database can be stored on a data management system, or directly on a physical storage medium (*PhysicalStorage*), so as

to potentially increase performance. We assume that software reads data from and writes data to either a file or a database. In the case of a database, one can access a database either directly or indirectly through some kind of intermediate framework. The latter case is expressed by the association uses between `Software` and `PersistencyFramework`. Since it will often be important to know what database is used by a software system, the associations `read` and `write` between `Software` and `Database` are to represent both cases, direct and indirect access. The association uses between `Software` and `PersistencyFramework` only serves to add the information that some, or maybe all, database accesses are done indirectly through a persistency framework. An additional constraint ensures that a persistency framework can be assigned to a software system only, if it is linked to at least of one of the databases the software system writes to or reads from.

It may be irritating that `File` is represented by a meta type, while `Database` is a type only. This is indeed not satisfactory, but it is, once again, a reflection of the limitations of (traditional) meta modeling languages. One could either specify that a certain software system (not a particular instance of it) writes data either to a particular file or to a type of file (which could be expressed by the extension of the file name). The first alternative would be strange, since it would, e.g., imply that a certain type of software like *MS Word* always writes to a particular file. The second alternative is not convincing either, but is the lesser evil. It does not allow assigning particular instances of a software systems to particular files. The use of intrinsic attributes (`created`, `modified`, `size`) allows to express properties of particular files, but is, in the end, a poor workaround only, as long as it is not possible to create instances from instances.

The associations between `Software` and `Database` represent a similar constellation. Nevertheless, we decided for a different option here. It is based on an assumption that is questionable. Software systems of a certain kind that use databases are likely to use one (or a few) specific instances only. First, they will often exist in one or a few instances only, e.g., an ERP system. Second, even if there are a few instances, and this is the problematic part of the assumption, they are likely using the same database. Therefore, `Database` is, different from `File` located on M_0 .

Distributed resources. Here, we focus on a set of dedicated (predominantly software, but also hardware) components that allow to treat distributed resources, as if these are local resources. In other words, these components allow for transparent access to distributed resources (Tanenbaum and Bos 2014).

`Networks`, cf. Fig. 3.7, of which there exist several specialized types (like LAN, or WLAN), offer a communication infrastructure to various platforms (`SpecificPlatform`) connected by the network(s), which can be used by software running on these platforms.

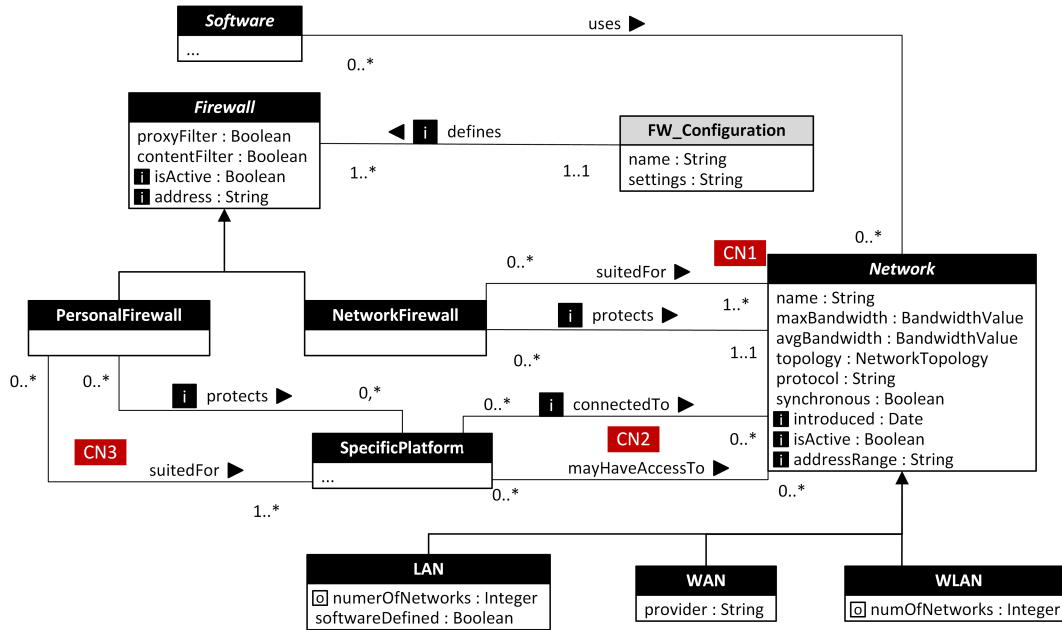


Figure 3.7: Meta model Excerpt: Network and related concepts, for constraints, cf. Tab. 3.7

In turn, cf. Fig. 3.8, Servers allow to access functionality offered by a software. There exist a variety of software systems, which are implemented as a server. Of particular importance are Servers which allow for remote data access, being `FileServer`, `DBMS_Server`, and a `Webserver`, which hosts websites. Additionally, `ApplicationsServers`, which are often considered as being part of `Middleware`, ensure distributed execution of programs, and ensure availability of the programs over the `Middleware`. `Middleware` offers operating system functions in a heterogeneous, distributed environment, so that distribution is (ideally speaking) made transparent. The user and programmer (again, ideally) receives the impression of working on a local machine. `Workflow-Management-Systems` (`WFMS` in Figure 3.8) enable the use of functions from different, distributed, software systems in the context of process execution.

One notion has gained particular importance in recent years: container and its associated concepts. Important to containers are container images (`ContainerImage`), whereby a container image is a file that contains all the resources required to run a program, except for the operating system kernel. This enables a high degree of portability. At the same time, the installation can be largely automated. A container image is loaded when required, which creates a container, one could also say: the container is instantiated from the container image. A container is executed by a special virtual machine (`ContainerEngine`). Containers can be combined into groups so that the containers in a group share resources in a common namespace. In addition, a `ContainerOrchestrator` can be used. The container orchestrator manages the life cycle of container systems, so when the load increases, it starts additional containers or replaces

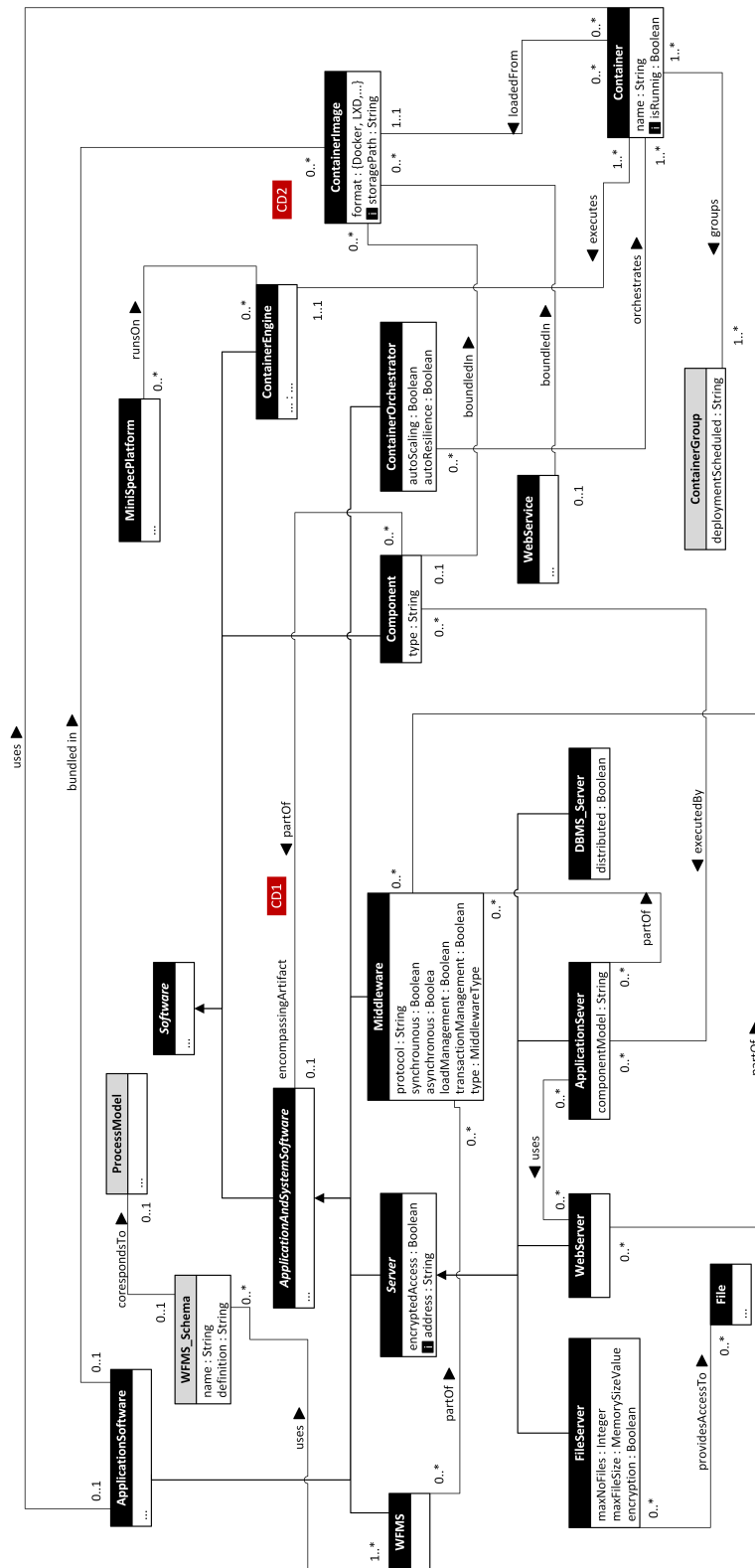


Figure 3.8: Meta model excerpt: Aspects related to selected distributed resources, for constraints, cf. Tab. 3.7

containers that fail. A well-known example of a program used to develop and run containers is Docker, while Kubernetes is currently the best-known container orchestration software.

Table 3.9: *Language specification: Comments on **Container Image**, cf. Fig. 3.8*

Concept: ContainerImage	Description: a file containing all the resources required to run a program (except for the kernel of the operating system).	
Attributes on the type level		
format	Docker,LXD,...	Indicates the type of container image. E.g., Docker supports more the deployment of applications, whereas LXD supports more the deployment of Linux virtual machines.
Attributes with reference to the instance level		
storagePath	String	Indicates the the location of a Container Image.
Selected Associations		
bundledIn	0,1 – 0,*	with ApplicationSoftware/Component/WebService respectively, to indicate that a container image bundles these to provide a self-contained environment offering resources to run a program.

Table 3.10: *Language specification: Comments on **Middleware**, cf. Fig. 3.8*

Concept: Middleware	Description: software offering operating system-like functions in a distributed environment.	
Supertypes		
ApplicationAndSystemSoftware		
Attributes on the type level		
protocol	String	Prominently middleware protocols support communication over the middleware.
synchronous	Boolean	Indicates if the middleware is of a synchronous nature.
asynchronous	Boolean	Indicates if the middleware is of an asynchronous nature.
loadManagement	Boolean	Indicates the support of load balancing, i.e., the efficient distribution of resources, by the middleware.
transaction- Management	Boolean	In transaction middleware, transactionManagement indicates support for ensuring atomicity, isolation, and durability of a transaction.
type	MiddlewareType	Often occurring types include object-oriented middleware, transaction middleware, message-oriented middleware.
Attributes with reference to the instance level		
storagePath	String	Indicates the the location of a container image.
Selected Associations		
partOf	0,* – 0,*	with WebServer, ApplicationServer and WFMS.

3.2 Selected IT-centric Analysis

IT Security. In line with our analysis scenarios discussed in the previous chapter, security is a central issue in IT management, the importance of which is expected to continue to grow. On the one hand, this is due to the increasing penetration of software by companies and the associated dependency of the service creation processes on software and the managed data. On the other hand, more and more valuable and confidential data is being managed in information systems.

The meta model excerpt in Figure 3.9 depicts two aspects of essential importance to IT security: (i) the protection from non-authorized attacks on networks or (virtual) platforms, by means of a `Firewall` and its various specializations (cf. also Fig. 3.7), and (ii) data encryption, which is indicated as an attribute for various components like a `FileSystem` (`encrypted: Boolean`), or a `Server` (`encryptedAccess: Boolean`). Please note that the ITML has a simplified conception of security only, as it is focused on supporting IT management analyses – for whom security is only one concern among many. In addition, it must be taken into account that IT security does not only depend on technical protective measures, but is also endangered by incompetence, negligence or malevolence. Therefore, just looking at the IT infrastructure is not sufficient, but rather, threats and protective measures that consider human actions must also be taken into account. A more elaborate conception of IT security can be found in Goldstein and Frank (2016).

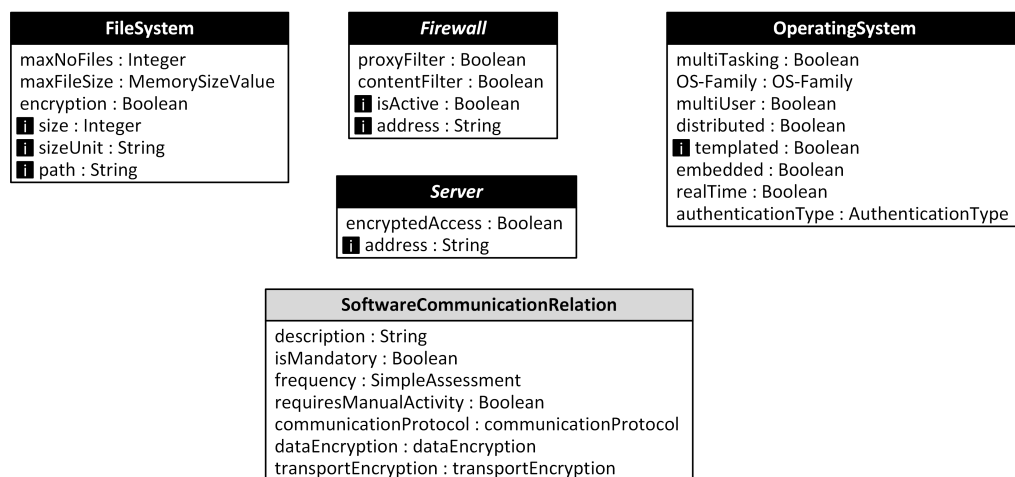


Figure 3.9: Selected aspects related to IT security

Maintainability, Performance, Availability: As noted in Section 3.2.2-3.2.5, we support analyses of the “qualities” maintainability, performance, and availability with a focus on capturing the perception of the end user. The meta model excerpt in Figure 3.10 highlights this focus on capturing the end user perception of these qualities, in terms of the attributes of the

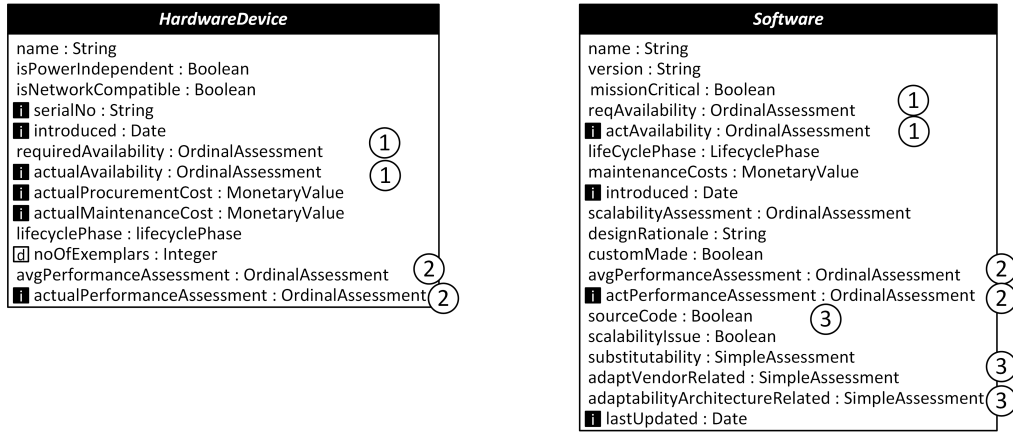


Figure 3.10: Meta Model Excerpt: supporting Maintainability, Performance, Availability

two meta types `Hardware` and `Software`, from which specific hardware or software artifacts inherit, like peripheral hardware for hardware cf. Section 3.1.

Specifically, availability is captured with attributes labeled as 1, whereby for both hardware and software, both the required availability (`reqAvailability` on the type level), and the actual availability (`actAvailability` on the instance level) are captured. Performance is captured with attributes labeled as 2, whereby for both hardware and software artifacts a distinction is made between the average estimated performance and the actual estimated performance. Finally, maintainability is captured in terms of the software attributes `adaptVendorRelated`, `adaptArchitectureRelated`, both of which focus on capturing estimated values, and the software attribute `sourceCode`, which captures the availability of source code.

3.3 Integration: Technologies, Languages, and Conceptual Integration

With regard to the evaluation of IT infrastructures, the analysis of possible integration deficits is of particular importance. The ITML supports such analyzes with dedicated concepts. Some of these concepts have already been presented in connection with persistence and distribution: databases that can be used by several software systems, middleware that allows communication between heterogeneous applications, or WFMS that enables the dynamic integration of functions of different software systems.

As per our scenarios in Section 2, integration covers various aspects. Therefore, even if the consideration of dedicated integration technologies such as databases and middleware systems is very informative, it is not sufficient in order to conduct a differentiated assessment of the integration of two software systems. If, for example, two application systems use the same database, it does not necessarily follow that all of the data required for communication between these systems is also jointly used. To make an assessment regarding the level of integration, it is

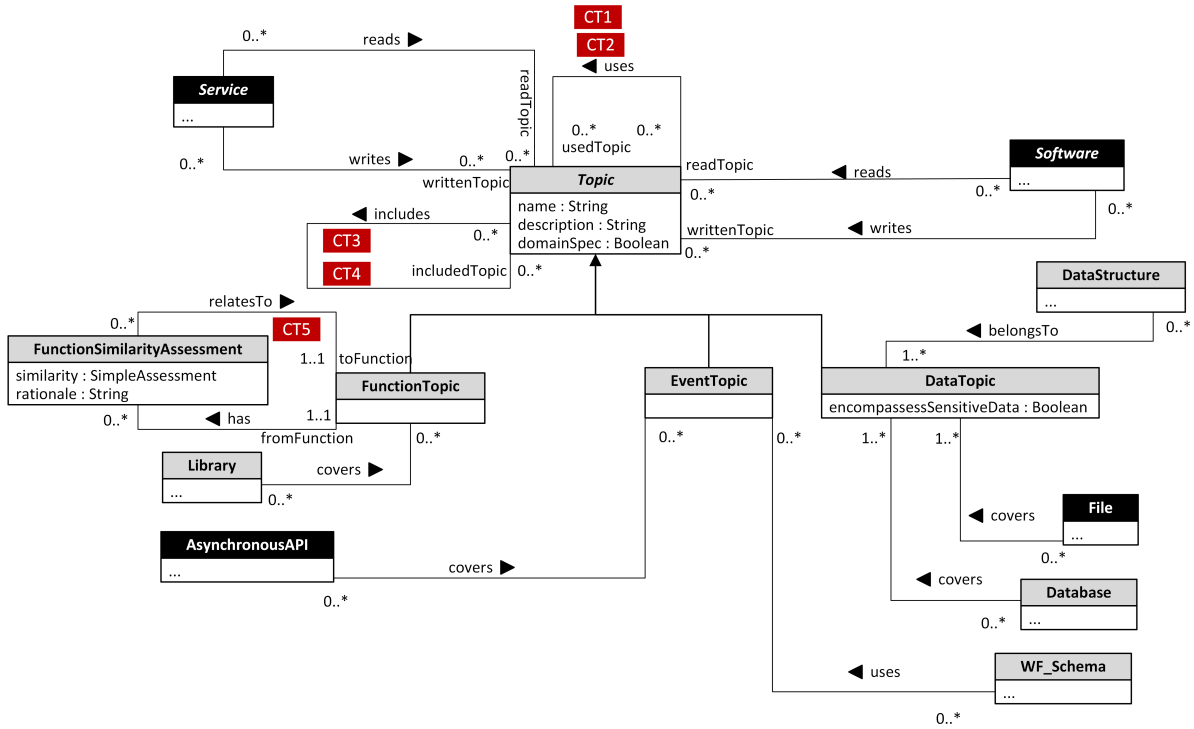


Figure 3.12: Meta Model Excerpt: Topics, for constraints, cf. Tab. 3.7

implementations are the same, in terms of either their contents and naming conventions. For example, to represent a Customer an application A could use the class “Customer” with the attributes “firstName”, “lastName”, and “birthdate”, while for the same Customer representation application B uses a data structure “Kunde” with five attributes each having a specific name.

Table 3.11: Language specification: Comments on *Topic*, cf. Fig. 3.12

Concept: Topic	Description: Language Level Type. Abstraction over static, functional and dynamic aspects.	
Subtypes		
DataTopic, FunctionTopic, EventTopic		
Attributes		
name	String	Allows for assigning a name. Note that the name should be unique within the context of a subtype.
description	String	Allows for providing a description of a topic.
domainSpecific	Boolean	Serves the specify whether the given topic is domain specific or not.
Associations		
includes	0,* – 0,*	with self; allows to build hierarchies of topics.
uses	0,* – 0,*	with self; allows to state what other topics are required by a given topic.
Constraints , for a definition, see Table 3.7.		

Continued on next page

Table 3.11 – Continued from previous page

CT1	A topic cannot use itself.
CT2	A topic can only use another topic of the same type (e.g., data topic can only use another data topic).
CT3	A topic cannot include itself.
CT4	A topic can only include another topic of the same type (e.g., data topic can only include another data topic).
CT5	A function topic cannot be similar to itself.

A data topic represents a prototypical data structure or class, which abstracts away from the used structure, as well as the used identifiers. So it is an abstraction over concrete data objects. Examples of data topics include Customer, Product, Account, or Bill. Data topics allow to answer the following analysis questions: (1) What are data topics covered in our system? (2) Are there applications that share the same data topics? Please note, that when a first analysis shows that two software artifacts have the same data topics, a further differentiated analysis is recommended, as it may indicate data redundancy.

A function topic represents a prototypical function. Examples include “print document”, “create price list”, or “compute account balance”. When two software systems use the same function topic, one should check if there exists one implementation, or rather multiple ones. So the concept supports the following analysis purposes: (1) What functions are not yet covered by existing artifacts? (2) Do we have different systems covering the same functions?

Similarly, an event topic represents a prototypical event. Example event topics include “account overdrawn”, or “payment canceled”. When two applications generate the same event topic, one should clarify, if one deals with the same event type. If this is the case, one should analyze, if the according concrete events can also be generated by outside systems, for example by a workflow management system, and which operations or processes the concrete events trigger. When an event triggers different processes, this can hint at redundant processes, functions or – in a related manner – a problem with consistency.

3.4 IT Architecture

Finally, the notion of IT architecture is of interest to IT management, especially to gain an abstract overview of those aspects of an IT infrastructure that are of high relevance in light of certain design objectives. At the same time, for an IT architecture one should focus on invariant aspects, less likely to be subject to change.

Figure 3.13 shows an ITML meta model excerpt, which allows for modeling diverse sets of IT architectures. IT architectures contain groups (Clusters) of software systems and hardware components. Clusters may depend on other clusters (cf. *ArchitectureDependency*). The design goals and principles, which an IT architecture should satisfy, can be documented in a corresponding Architecture Pattern (*ArchitecturePattern*).

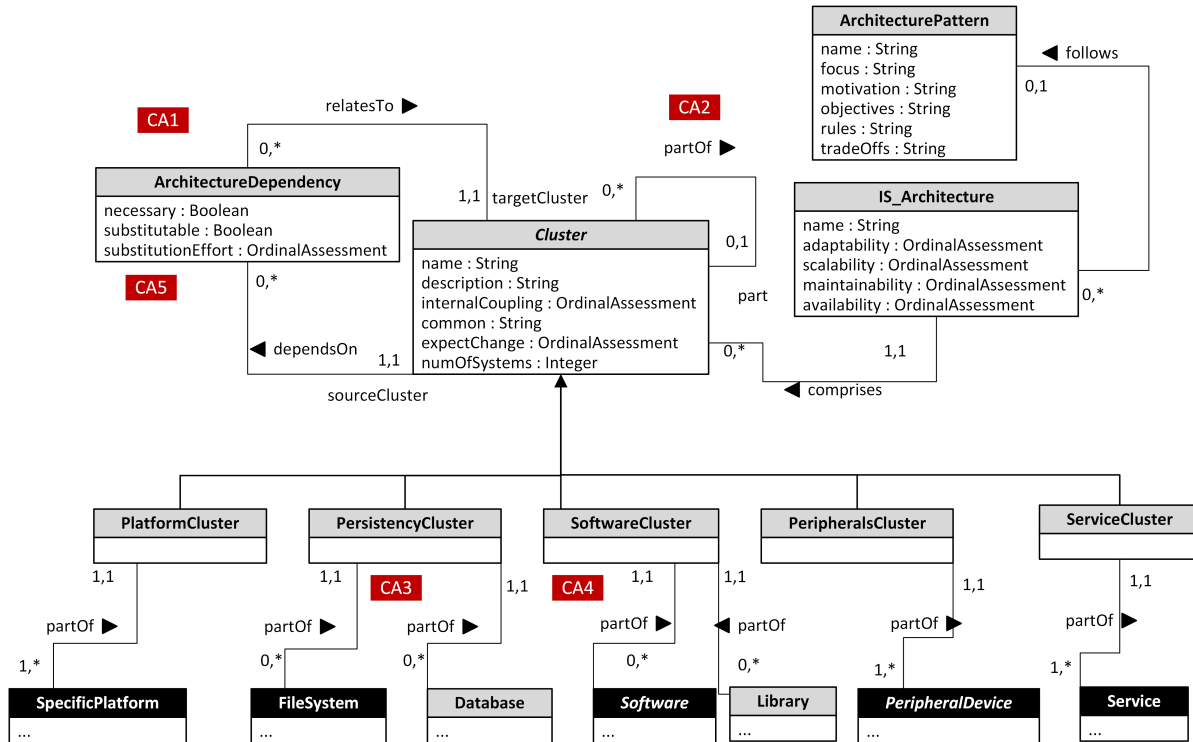


Figure 3.13: Meta Model Excerpt: IS Architecture, and Clusters, for constraints, cf. Tab. 3.7

Table 3.12: Language specification: Comments on *Cluster*, cf. Fig. 3.13

Concept: Cluster	Description: Describes logical groupings of Hardware/Software elements, as a pre-requisite for describing architectural patterns.	
Subtypes		
PlatformCluster, PersistencyCluster, SoftwareCluster, PeripheralsCluster, ServiceCluster		
Attributes on the type level		
name	String	Allows for assigning a type name. Note that the name should be unique within the context of a cluster type.
description	String	Allows for providing a description of a cluster type, e.g., main focus or a reason for grouping.
internal-Coupling	OrdinalAssessment	Serves the assessment (on the ordinal scale) of the internal coupling within the cluster type.
common	String	Allows for specifying the common property or element of the elements being part of the cluster type.
expectChange	OrdinalAssessment	Serves to specify the assessment of the expected level of change within the cluster type.

Continued on next page

3 Language Design: Abstract Syntax and Semantics

Table 3.12 – Continued from previous page

[d] numOfWorkSystems	Integer	Provides information on the number of elements/exemplars/... being part of the cluster type.
Associations		
partOf	0,* – 0,1	with self
dependsOn	1,1 – 0,*	via AssociationClass ArchitectureDependency with self. ArchitectureDependency through the following properties allows to capture: necessity of the dependency (necessary: Boolean), its substitutability (substitutable : Boolean) as well as effort required to do that (substitutionEffort: OrdinalAssessment), in case substitutability is possible.
relatesTo	0,* – 1,1	
Constraints		
CA1	A cluster cannot depend on itself.	
CA2	A cluster cannot be part of itself.	
CA3	At least one artifact supporting persistency needs to be part of a Persistency Cluster.	
CA4	At least one (software) artifact needs to be part of a Software Cluster.	
CA5	If the architecture dependency can be substituted, the expected substitution effort should be stated.	

Table 3.13: Language specification: Comments on *IS_Architecture*, cf. Fig. 3.13

Concept:	Description: used to represent the structure of an IS	
IS_Architecture	in terms of its functional units, and how these units interact.	
Attributes		
name	String	Allows for assigning a name to the IS Architecture type.
adaptability	OrdinalAssessment	Allows to specify the assessed level of adaptability of the IS_Architecture type using an ordinal scale.
scalability	OrdinalAssessment	Allows to specify the assessed level of scalability of the IS_Architecture type using an ordinal scale.
maintainability	OrdinalAssessment	Allows to specify the assessed level of maintainability of the IS_Architecture type using an ordinal scale.
availability	OrdinalAssessment	Allows to specify the assessed level of availability of the IS_Architecture type using an ordinal scale.
Associations		
comprises	1,1 – 0,*	with Cluster.
follows	0,* – 0,1	with ArchitecturePattern.

Table 3.14: *Language specification: Comments on **Architecture Pattern**, cf. Fig. 3.13*

Concept: Architecture- Pattern	Description: A language level type allowing to specify the core features of an architecture pattern that may be followed by an information system’s architecture.	
Attributes		
name	String	Allows for assigning an architectural pattern a name.
focus	String	Allows for defining the focus of the architecture pattern, e.g., selection and composition of information systems.
motivation	String	Allows for stating the motivation of the architecture pattern.
objectives	String	Allows for stating the objectives of the architecture pattern.
rules	String	Allows for defining rules one should follow while using the pattern.
tradeOffs	String	Allows to state existing trade offs that should be considered if a given pattern is followed.

3.5 Integrated IT Infrastructure and Action System Analysis

The concepts offered by ITML are designed to support analyses relevant to IT management. However, to address some analysis questions models of the enterprise action system are required in addition to IT infrastructure models. For example, we require an organizational structure model in addition to an IT infrastructure model to address questions on IT infrastructure ownership or responsibility.

Taking that into account, the ITML meta model is linked with concepts from other MEMO DSMLs: OrgML Business Processes and Organizational Structure (Frank 2014b), GoalML for Goal Modeling (Overbeek, Frank, and Köhling 2015), as well as DecisionML for Decision Modeling (Bock 2015). The excerpt from the meta model which shows the connections to the concepts from the other MEMO DSMLs is presented in Fig. 3.14.

Firstly, `SpecificSupport` (cf. Fig. 3.14) is used to create a bridge between elements of IT infrastructure and, respectively, business process (`AnyProcess`), a business goal (`AbstractGoal`), or a decision scenario (`DecisionScenario`) (the concept `UseContext` is used here as a surrogate). Secondly, `Specific Support` can be linked to the concepts `FunctionTopic` and `DataTopic`. As such, not only can we establish a link from IT infrastructure elements to elements of the enterprise action system (a process, decision scenario, or goal), but we can also establish a link to required functionality, or analyse the involvement of some data. Finally, with the attributes of specific support we can characterize the nature of the cross-model relation in terms of: `relevance`, `supportQuality`, `dependency`, and

3 Language Design: Abstract Syntax and Semantics

performance. Thus, the introduced concepts serve the following analysis purposes: What contexts are supported? How good is the support? How much does a use context depend on the software?

Specifically, consider the conjoint use of a business process model and an IT infrastructure model. This allows to conduct an assessment of the extent to which the (core) business processes rely on the IT infrastructure. Also, we can identify critical business/IT services. Knowing which services are critical allows to investigate further the IT infrastructure supporting those services, i.e., used systems and hardware. One can also analyze what functionalities are needed, as well as what data is being used, if relevant.

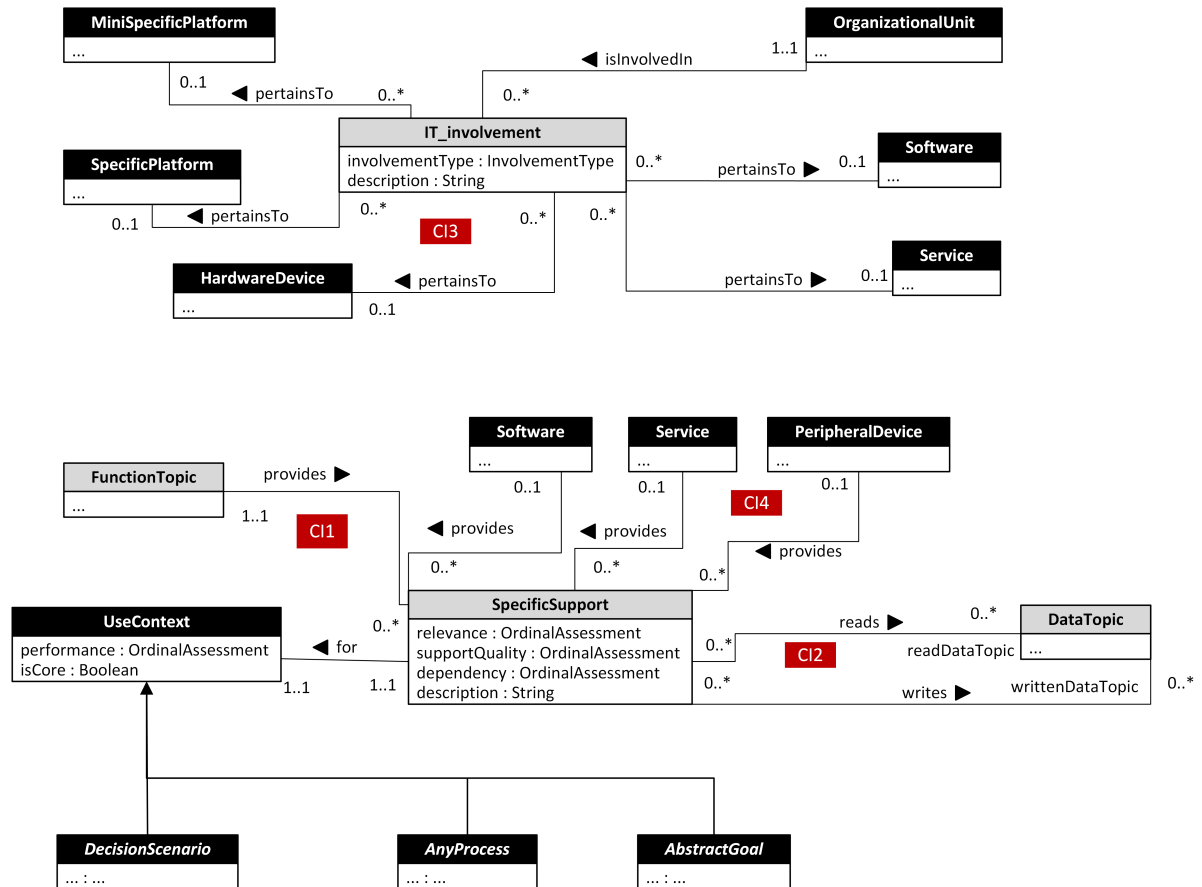


Figure 3.14: Meta Model Excerpt: Integration with other MEMO DSMLs, for constraints, cf. Tab. 3.7

Additionally, considering the conjoint use of goal models and IT infrastructure models, it is possible to state to what extent the given element of the IT infrastructure supports/hinders the achievement of defined goals (by looking at the defined attributes, e.g., relevance or dependency). One may also investigate what function exactly (FunctionTopic) or what data (DataTopic) are here of interest.

Table 3.15: *Language specification: Comments on **Specific Support**, cf. Fig. 4.10*

Concept: Specific Support	Description: Language Level Type allowing for relating elements of the ITML to be related to elements of DecisionML, OrgML and GoalML.	
Attributes		
relevance	OrdinalAssessment	Serves the assessment (on the ordinal scale) of the estimated relevance of the IT artifact and functionalities provided by it to the Business Process, Goal or a Decision Scenario, in question.
supportQuality	OrdinalAssessment	Allows stating the estimated quality of the support provided.
dependency	OrdinalAssessment	Serves to specify the assessment of the dependency, i.e., to what extent is the execution of a business process or achievement of a goal dependent on the IT artifact and functionalities provided by it.
description	String	Allows to provide additional characteristics of interest.
Selected Associations		
for	1,1 – 1,1	Allows to state the context of usage, i.e., point to a process, goal or a decision scenario, dependent on an IT artifact.
uses	1,1 – 0,*	and further refersTo (0,* – 1,1) DataTopic allows to point to data (and their usage characteristics, i.e., extent and type of usage) used (e.g., used by a process, or required to accomplish a goal).
provides (source: FunctionTopic, Target: SpecificSupport) provides (source: IT_Artifact, Target: SpecificSupport)	1,1 – 0,* 1,1 – 0,*	It is possible to state what IT artifact and what functionality (FunctionTopic) is being provided.
Constraints , for the OCL definition see Tab. 3.7		
CI1	A function topic assigned to a specific support needs to be one covered by an IT artifact providing the specific support.	
CI2	A data topic related to a specific support needs to be one covered by an IT artifact providing the specific support.	

Finally, by using the concept `IT Involvement`, one can detail what is the involvement of organizational units, `OrganizationalUnit` (OrgML), e.g., by using the RACI dependencies (responsible, accountable, consulted, informed), when it comes to different IT artifacts (e.g., `Software`, `SpecificPlatform` or `Service`). One can also show how different units of work are involved in the management of IT infrastructure, and identify problems that result out of the incorrect division of responsibilities (e.g., different organizational units involved in the maintenance process).

3 Language Design: Abstract Syntax and Semantics

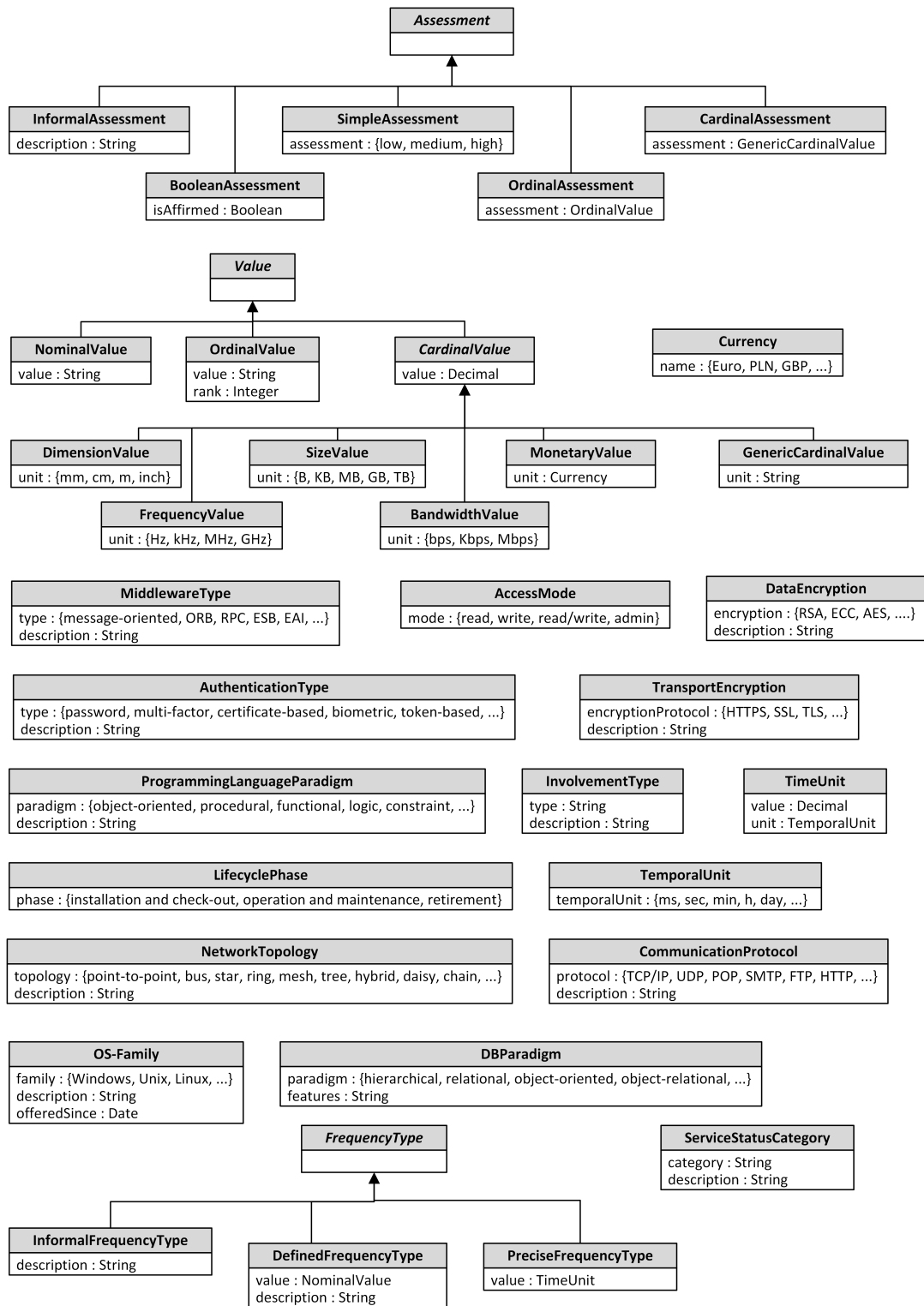


Figure 3.15: Auxiliary Types

3.6 Auxiliary Types

In addition to the central concepts of the modeling language, the meta model also contains a large number of auxiliary types. These auxiliary types are used to make attributes of meta types semantically richer than the basic data types available in the MEMO MML. The use of the auxiliary types enables the option to further differentiate the description of a meta type at a later point in time without having to adapt the meta types. The referenced auxiliary types are listed below. The supporting types are defined as Language Level Types, as they are specified on the meta level, but can only be instantiated once on the type level.

- the abstract auxiliary type `Assessment` is used as an abstraction over various assessment possibilities, characterized by various degrees of formalization, i.e., from simple true / false determinations (`BooleanAssessment`) to ordinal scales (e.g., from “very low” to “very high”, or as some specific categories of, e.g., availability assessment, e.g., “0%–20%” to “90%-100%”; `OrdinalAssessment`), to cardinal scales (`CardinalAssessment`). It is recommended to use a dedicated specialized type of the assessment, and apply the abstract type `Assessment` only if the actual type of assessment cannot be determined;
- the abstract auxiliary type `Value` and its numerous specializations are used to specify values on a nominal, ordinal or cardinal scale. A cardinal value (i.e., measured value and corresponding measurement units) encompasses such specific types as, among others, `DimensionValue`, `WeightValue`, `ClockRateValue`, `MemorySizeValue`, `BandwidthValue` or `MonetaryValue`. Similarly like in case of `Assessment`, it is recommended to use specialized types and apply the abstract `Value` type, only if the actual type cannot be determined yet;
- `CommunicationProtocol` – serves to indicate and describe characteristics of communication protocols. Communication protocols are defined as an enumeration list (e.g., Transmission Control Protocol (TCP), Internet Protocol (IP), User Datagram Protocol (UDP), Post office Protocol (POP), Simple mail transport Protocol (SMTP), File Transfer Protocol (FTP), Hyper Text Transfer Protocol (HTTP)).
- `DataEncryption` – serves to define data encryption standards (e.g., Triple DES, RSA, Twofish, Elliptic Curve Cryptography (ECC), The Advanced Encryption Standard (AES)), and their relevant characteristics.
- `TransportEncryption` – serves to define transport encryption standards (e.g., HTTPS, SSL, TLS) and their relevant characteristics.
- `NetworkTopology` – serves to define topologies of network (e.g., point-to-point, bus, star, ring, mesh, tree, hybrid, daisy chain) and their relevant characteristics.

3 Language Design: Abstract Syntax and Semantics

- `MiddlewareType` – serves to define middleware types(e.g., message-oriented, object request broker, remote procedure call, enterprise service bus, enterprise application integration, data integration, database-oriented middleware, transaction-oriented middleware) and their relevant characteristics.
- `LifeCyclePhase` is used to indicate the phase of the lifecycle a given artifact is in, e.g., installation and check-out, operation and maintenance, retirement .
- `AuthenticationType` is used to indicate the type of authentication used, e.g., password, multi-factor, certificate-based, biometric, token-based.
- `AccessMode` used to indicate the mode of access, i.e., read, write, read/write, admin.
- `OS-Family` used to indicate the root family the operating system type belongs to, e.g., windows OS, Unix, Linux, z/OS, or Mac OS.
- `DBParadigm` is used to describe a database paradigm, currently in the form of an enumeration, e.g. relational, object-oriented, hierarchical.
- `ProgrammingLanguageParadigm` is used to describe a programming language paradigm (e.g., object-oriented, procedural, functional, logic and constraint).
- `FrequencyType` supports the description of a frequency in various degrees of formalization, e.g., in the simplest form as an enumeration list with the following values: rarely, often, very often, once per day, once per week, once per month, once per period, every few hours.
- `InvolvementType` enables a more detailed qualification of the participation of an organizational unit, for example in text form or with reference to, e.g., RACI matrix.
- `ServiceStatusCategory` is used to describe different categories of a status a service may have.

3.7 Constraints

As already mentioned, whenever possible, we formulate the constraints using the Object Constraint Language. If the constraint in question cannot be expressed using OCL, because it actually spans both type and instance level, we formulate it using a natural language instead. The following table encompasses all constraints defined within the ITML meta model.

Table 3.16: Constraints

ID	OCL	Natural language
CA1	context ArchitectureDependency inv: <i>self.sourceCluster <> self.targetCluster</i>	A cluster cannot have an architectural dependency to itself.
CA2	context Cluster inv: <i>self.part <> self</i>	A cluster cannot be part of itself.
CA3	context PersistencyCluster inv: <i>self.FileSystem- > notEmpty()</i> OR <i>self.Database- > notEmpty()</i>	At least one artifact supporting persistency needs to be part of a Persistency Cluster.
CA4	context SoftwareCluster inv: <i>self.Software- > notEmpty()</i> OR <i>self.Library- > notEmpty()</i>	At least one (software) artifact needs to be part of a Software Cluster.
CA5	context ArchitectureDependency inv: <i>(self.substituable implies self.substitutionEffort- > notEmpty())</i> AND <i>(not(self.substituable) implies self.substitutionEffort- > isEmpty())</i>	If the dependency is substituable, the expected substitution effort needs to be stated. If not, it should not be stated.
CI1	context SpecificSupport inv: <i>self.IT_Artifact- > notEmpty() implies self.IT_Artifact.Topic- > exists(self.FunctionTopic)</i>	A function topic assigned to a specific support needs to be also covered by an IT artifact providing the specific support. Please note that IT_Artifact is used here as a surrogate for Software, Service, PeripheralDevice.
CI2	context SpecificSupport inv: <i>(self.PeripheralDevice- > notEmpty() implies (self.writtenDataTopic- > isEmpty() and self.readDataTopic- > isEmpty()) AND ((self.Software- > notEmpty() and self.readDataTopic- > notEmpty()) implies self.readDataTopic- > for All(d self.Software.readTopic- > exists(d)) and ((self.Software- > notEmpty() and self.writtenDataTopic- > notEmpty()) implies self.writtenDataTopic- > for All(d self.Software.writtenTopic- > exists(d)) and ((self.Service- > notEmpty() and self.readDataTopic- > notEmpty()) implies self.readDataTopic- > for All(d self.Service.readTopic- > exists(d)) and ((self.Service- > notEmpty() and self.writtenDataTopic- > notEmpty()) implies self.writtenDataTopic- > for All(d self.Service.writtenTopic- > exists(d))</i>	A data topic related to a specific support needs to be also covered by a Software or a Service providing the specific support. Please note that if a support is provided by a peripheral device, no data topics are assigned.

Continued on next page

3 Language Design: Abstract Syntax and Semantics

Table 3.16 – Continued from previous page

ID	OCL	Natural language
CI3	context IT.Involvement inv: <i>self.Software</i> – > <i>notEmpty()</i> <i>XOR self.Service</i> – > <i>notEmpty()</i> <i>XOR self.HardwareDevice</i> – > <i>notEmpty()</i> <i>XOR self.SpecificPlatform</i> – > <i>notEmpty()</i> <i>XOR self.MiniSpecificPlatform</i> – > <i>notEmpty()</i>	IT Involvement needs to pertain to one of the following artifacts: <i>MiniSpecificPlatform</i> , <i>SpecificPlatform</i> , <i>HardwareDevice</i> , <i>Service</i> , or <i>Software</i> .
CI4	context SpecificSupport inv: <i>self.Software</i> – > <i>notEmpty()</i> <i>XOR self.Service</i> – > <i>notEmpty()</i> <i>XOR self.PeripheralDevice</i> – > <i>notEmpty()</i>	Specific specific support is provided by only one of the following: <i>Service</i> , <i>Software</i> , or <i>PeripheralDevice</i> .
CT1	context Topic inv: <i>self.usedTopic</i> – > <i>excludes(self)</i>	A topic cannot use itself.
CT2	context Topic inv: (<i>self.usedTopic</i> – > <i>forEach(t t.oclsTypeOf(self))</i>)	A topic can use only topics of the same type as it is.
CT3	context Topic inv: <i>self.includedTopic</i> – > <i>excludes(self)</i>	A topic cannot include itself.
CT4	context Topic inv: (<i>self.includedTopic</i> – > <i>forEach(t t.oclsTypeOf(self))</i>)	A topic can include only topics of the same type as it is.
CT5	context FunctionSimilarityAssessment inv: (<i>self.fromFunction</i> <> <i>self.toFunction</i>)	A function topic cannot be similar to itself.
CP1	context Platform inv: (<i>self.MiniSpecificPlatform</i> – > <i>notEmpty()</i>) <i>xor self.SpecificPlatform</i> – > <i>notEmpty()</i>	A platform type may be part of at least one specific platform or mini specific platform.
CP2	–	An Application and system software only runs on (<i>runsOn</i>) specific, specific platforms on whose types the software can be executed (<i>executableOn</i>).
CP3	–	A virtual machine cannot run on a virtual machine it is realizing.
CP4	–	CPU only mounts to (<i>mountedOn</i>) specific hardware platforms whose types the CPU fits.
CP5	–	A specific mounted rack mounts on specific racks on whose types it is mountable on.
CP6	–	An instance of platform being part of a specific specific platform must be of type defined by the <i>partOf</i> relation on the type level.
CP7	–	An instance of an operating system assigned to a specific specific platform needs to be of type defined by a <i>partOf</i> relation on the type level.

Continued on next page

Table 3.16 – Continued from previous page

ID	OCL	Natural language
CP8	–	An instance of platform being part of a specific mini specific platform must be of type defined by the <code>partOf</code> relation on the type level.
CP9	–	A specific hardware platform can be built into a specific case that it <code>fits</code> (defined on the type level).
CH1	–	A hardware device may be connected to a specific platform being an instance of a type of a specific platform that a hardware device in question is connectable to.
CH2	–	A specific platform may access a hardware device being an instance of a type that a platform may have access to.
CH3	–	A software uses a hardware device being an instance of a type that it requires.
CS1	context Software inv: <i>self.usedSoftware</i> – > <i>excludes(self)</i>	A software artifact cannot use itself.
CS2	context Software inv: <i>self.part</i> – > <i>excludes(self)</i>	A software cannot be part of itself.
CS3	context SoftwareCommunicationRelation: <i>self.referredSoftware</i> <> <i>self.referee</i>	Software cannot communicate with itself.
CS4	context Vendor inv: <i>self.competitor</i> – > <i>excludes(self)</i>	An organization being a vendor cannot be a competitor to itself.
CS5	context Vendor inv: <i>self.partner</i> – > <i>excludes(self)</i>	An organization being a vendor cannot be a partner to itself.
CS6	context Software inv: <i>self.usedMI</i> <> <i>self.definedMI</i>	Software cannot use the middleware interface it defines.
CS7	–	A specific software can be stored only on (instance level) replication files that this type of software is storable on (type level).
CS8	context Software inv: <i>self.usedLibrary</i> – > <i>excludes(self.representingLibrary)</i>	Software cannot use a library that it represents.
CS9	context Service inv: <i>self.usedAPI</i> – > <i>excludes(self.providedAPI)</i>	A Web service cannot use and provide the same API.
CS10	context Service inv: <i>self.usedService</i> – > <i>excludes(self)</i>	A service cannot use itself.
CS11	context Service inv: <i>self.providingSoftware</i> – > <i>notEmpty()</i> XOR <i>self.providingWS</i> – > <i>notEmpty()</i>	An API can be either provided by a Software or by a Web service.
CS12	context SoftwareCommunicationRelation inv: <i>self.API</i> – > <i>forAll(a a. oclIsTypeOf(AsynchronousAPI) implies self.isAsynchronous)</i>	Communication relation – if asynchronous API is attached, the the attribute <code>isAsynchronous</code> needs to be set to true.

Continued on next page

3 Language Design: Abstract Syntax and Semantics

Table 3.16 – Continued from previous page

ID	OCL	Natural language
CS13	context Component inv: <i>self.wrappedArtifact</i> – > <i>excludes(self)</i>	A component cannot wrap itself.
CS14	context SoftwareCommunicationRelation inv: <i>self.API</i> – > <i>notEmpty()</i> implies <i>self.API</i> – > <i>forall(a self.referredSoftware. providedAPI – > includes(a))</i>	Communication relation – using an API that referred or referee software provides.
CN1	–	A specific network can be protected only by specific network firewall (instance level) that is suited for this network (type level).
CN2	–	A specific network can be used by a specific Specific platform (instance level) only if this specific platform on a type level may have access to it.
CN3	–	A specific personal firewall protects a specific specific platform (instance level), if a personal firewall type is suited for this specific platform type.
CD1	context Component inv: <i>self.encompassingArtifact</i> – > <i>excludes(self)</i>	A component cannot be part of itself.
CD2	context ContainerImage inv: <i>self.WebService</i> – > <i>notEmpty()</i> XOR <i>self.Component</i> – > <i>notEmpty</i> XOR <i>self.ApplicationSoftware</i> – > <i>notEmpty()</i>	A container image may be bounding only one artifact.

3.8 Requirements and Their Fulfillment

In this section, we reflect upon the extent to which the ITML as discussed thus far fulfills the requirements as they have been introduced in Chapter 2.

Table 3.17: *Requirements and their fulfillment*

Requirement	Comment
R1: The ITML should provide differentiated concepts supporting comprehensive modeling IT infrastructure elements, their properties and dependencies between different elements, both on the type-level as well as instance-level.	As we have used MEMO Meta Modeling Language, we have used intrinsic properties (attributes and relations) to account for both type and instance-level information. This allowed us to account for both, e.g., average performance of some IT artifact, as well as its actual performance or date of usage, or when it comes to relations, e.g., point that while some software is on the type level executable on some type of a specific platform, a specific instance of software runs on a specific instance of a specific platform. Nevertheless, while accounting for this requirement we also faced numerous challenges and limitations which may be traced back to the language architecture used, for details please see Chapter 5.
R2: The ITML should provide differentiated concepts supporting comprehensive modeling of hardware platforms and associated concepts.	Next to providing an explicit conception of <code>HardwarePlatform</code> (specified in terms of attributes such as <code>actMemory</code> , and having an association with one or more CPUs), <code>HardwarePlatform</code> is also differentiated explicitly from other platform types, such as a <code>VirtualPlatform</code> , and (specific to the notion of a container), a <code>MiniSpecificPlatform</code> .
R3: The ITML should provide differentiated concepts supporting comprehensive modeling of software artifacts and related concepts.	A hierarchy of software artifacts has been created, whereby different types of software are modeled as subtypes of, either, directly the abstract meta type <code>Software</code> (like <code>ApplicationAndSystemSoftware</code>), or (generally speaking) as siblings of the subtype <code>Software</code> (like <code>ApplicationSoftware</code> , being a subtype of <code>ApplicationAndSystemSoftware</code>).
R4: The ITML should provide differentiated concepts supporting comprehensive modeling of IT services.	A notion of <code>Service</code> , in an IT-context, has been modeled, and has been related to directly important concepts such as a <code>ServiceContract</code> .
R5: The ITML should provide differentiated concepts supporting comprehensive modeling of hardware devices (e.g., peripheral devices).	Two subtypes of an abstract meta type <code>HardwareDevice</code> have been distinguished. Next to <code>PeripheralDevice</code> (which in turn has its subtypes like <code>Printer</code>) we also distinguish the subtype <code>InfrastructuralHardware</code> , mostly to point to hardware devices which provide a network access.
R6: The ITML should provide differentiated concepts supporting comprehensive modeling of other elements of IT infrastructure, such as network or data storage.	Next to aforementioned main types of respectively software or hardware, additional IT infrastructure concepts needed for analysis purposes, like network and related concepts, library, agreement, etc. have been accounted for. To support other than considered analysis scenarios, additional concepts may be required.

Continued on next page

Table 3.17 – Continued from previous page

Requirement	Comment
R7: The ITML should express properties of different IT infrastructure concepts as they are part of the professional IT infrastructure discourse.	Several properties have been defined for IT infrastructure concepts to differentiate them according to the analysis needs. Furthermore, the ITML caters for auxiliary data types to provide further differentiation for said properties, such as the auxiliary type <code>Assessment</code> or <code>MonetaryValue</code> .
R8: The ITML should allow to model dependencies among various elements of the IT landscape.	Various kinds of dependencies have been catered for, depending on the analysis needs / needs for semantic differentiation. Additionally, for dependencies which introduce cycles (as is the case for reified associations), where needed, constraints have been introduced to avoid models, which are not permissible.
R9: The ITML should account for supporting basic analysis related to authorization, authentication and confidentiality. To this aim relevant properties of IT artifacts, their interactions, as well as dedicated concepts and their configuration, should be accounted for.	From a technical perspective, security related aspects have been accounted for in terms of attributes like <code>dataEncryption</code> (for various ITML concepts), or dedicated security concepts such as <code>Firewall</code> . However, as mentioned in Section 3.2 human factors regarding security are not taken into consideration.
R10: The ITML should account for supporting basic analysis of maintainability of the IT infrastructure. Thus, it should account for architectural information as well as dependencies among elements of IT landscape allowing to analyze the impact of a modification.	To account for this requirement, firstly, a set of attributes has been accounted for, e.g., pointing to the availability of the source code and documentation, version of the system. Secondly, through defined dependencies one can learn what language has been used for the implementation, what APIs/Libraries/Other Software Artifacts are used/required, and many more.
R11: The ITML should account for additional relationships reflecting various dependencies between IT artifacts supporting a portability analysis.	Various semantically rich types of relations have been added in support of portability analysis, both on the type as well as on the instance level. For example, to state that a <code>HardwareDevice</code> is <code>connectableTo</code> a <code>SpecificPlatform</code> (on the type level), or that a <code>HardwareDevice</code> is <code>connectedTo</code> a specific <code>SpecificPlatform</code> (on the instance level).
R12: The ITML should account for characteristics of IT artifacts supporting portability analysis (e.g., constraining characteristics, usage of standards).	Where appropriate, the concepts in the ITML have various characteristics which allow for a portability analysis (like the attribute <code>isMobile</code> of <code>PeripheralDevice</code>). Of additional note is the notion of a container, and related concepts, which - by providing resources needed for an application to run - also supports portability.
R13: The ITML should express estimated performance of IT landscape elements, whereby a specification of performance depends on the type of IT landscape element.	For all concepts, relevant attributes, e.g., <code>avgPerformance</code> , <code>actPerformance</code> , have been defined.
R14: The ITML should express availability characteristics of IT landscape elements.	The required attributes to express availability have been included in the definition of relevant concepts.

Continued on next page

Table 3.17 – Continued from previous page

Requirement	Comment
R15: The ITML should provide concepts delivering information required for the needs of conducting a basic vendor analysis.	We explicitly define a <code>Vendor</code> as well as dependencies such as <code>competitor</code> or <code>partner</code> . By using a <code>has</code> relationship one can attach a vendor to software and hardware artifacts, including a platform.
R16: The ITML should explicitly account for data and file formats as well as implementation languages of different IT infrastructure elements.	The ITML encompasses an elaborate conception of different types of languages, including concepts to express the implementation language used. Additionally file and data formats are accounted for.
R17: The ITML should distinguish different types of middleware and corresponding relationships to other IT artifacts.	For the meta type <code>Middleware</code> we can express different types of middleware through the attribute type: <code>MiddlewareType</code> . Additionally several differentiating associations have been accounted for, like <code>ApplicationServer</code> and <code>WebServer</code> being part of a <code>Middleware</code> .
R18: The ITML should explicitly account for abstractions of the used data structures, allowing for classification of processed data, and provide means to model data flow in the system.	As part of an elaborate conception of different topic types, the meta type <code>DataTopic</code> allows for an abstraction of the used data structures.
R19: The ITML should explicitly account for functions covered by IT artifacts belonging to the IT landscape. It should be possible to relate software artifacts to functions (e.g., function topics) that they cover. Finally, it should be possible to state assessed similarity between functions.	As part of an elaborate conception of different topic types, the meta type <code>FunctionTopic</code> allows for an abstraction of the used functions. In addition, using a concept <code>FunctionSimilarityAssessment</code> we may state whether some function topics are similar.
R20: The ITML should explicitly account for interfaces and functionalities offered by IT artifacts.	The ITML specifically offers the meta types (1) <code>API</code> for a <code>Software</code> , and (2) <code>MiddlewareInterface</code> for <code>Middleware</code> . Relevant associations are offered as well.
R21: The ITML should allow to link elements of IT landscape and processes they support. The characteristics of the support provided should be accounted for.	Through the Language Level Type <code>SpecificSupport</code> and its attributes like <code>supportQuality</code> , an association of ITML concepts to concepts from <code>OrgML</code> for business processes is possible (through <code>AnyProcess</code>).
R22: The ITML should allow to relate IT landscape elements to goals of an organization, in terms of the influence of these elements on goal achievement. The characteristics of the influence provided should be accounted for.	Akin to R21, through both <code>SpecificSupport</code> and <code>AbstractGoal</code> a semantically rich relation can be established between the ITML and concepts from the <code>GoalML</code> .
R23: The ITML should allow to link elements of IT landscape with organizational units and express the role of linked organizational units.	The ITML can be related to an organizational unit as expressed in <code>OrgML</code> for structures through the Language level Type <code>IT_involvement</code> . <code>IT_involvement</code> also allows to express the specific responsibilities of an organizational unit.

4 Language Design: Concrete Syntax and ITML Diagram Types

In this chapter, first, the concrete syntax of ITML is introduced and different diagram types are discussed. The chapter concludes with providing a set of guidelines for modeling and designing IT infrastructure.

4.1 Concrete Syntax

While the concrete syntax of a modeling language is considered by some as “syntactic sugar”, which is of little relevance only, we assume that the graphical representation shown with a diagram is of considerable relevance for the comprehensibility and acceptance of the underlying model. That is why we paid a lot of attention to the design of the concrete syntax and also hired a professional graphic designer.

Figure 4.1 shows selected symbols used to represent the language concepts of extended ITML. With the design of the concrete syntax, we follow the guidelines proposed by a method for the design of DSML (Frank 2010, p. 48 ff.) that build on and extend the guidelines from Moody (2009) for designing visual notations. Moody’s guidelines include *semiotic clarity*, *perceptual discriminability*, *semantic transparency*, *visual expressiveness* and *graphic economy*. For example, semantic transparency implies that the meaning (semantics) of a symbol is clear (transparent) from its appearance alone (cf. the visualization of peripheral devices, Fig. 4.1). In turn, the method proposed by Frank (2010), suggests, among others, to define semantic categories of concepts and use generic symbols for each category, cf. Guideline 1 and 2 (Frank 2010, p. 50), as well as to use composition of symbols by adding features in a monotonic fashion, cf. Guideline 8 (Frank 2010, p. 52). For instance, having a look at the visualization of the virtual specific platform with UNIX, it has been created by putting together a symbol of virtual machine, operating system and a desktop symbol.

While these guideline proved to be useful, they still leave room for demanding decisions. For example, the proposal to distinguish semantic categories does not clearly determine the definition of categories. While a category like, e.g., *Software* is intuitively distinguishable from the category *Hardware*, that does not necessarily mean to not subdivide software into further

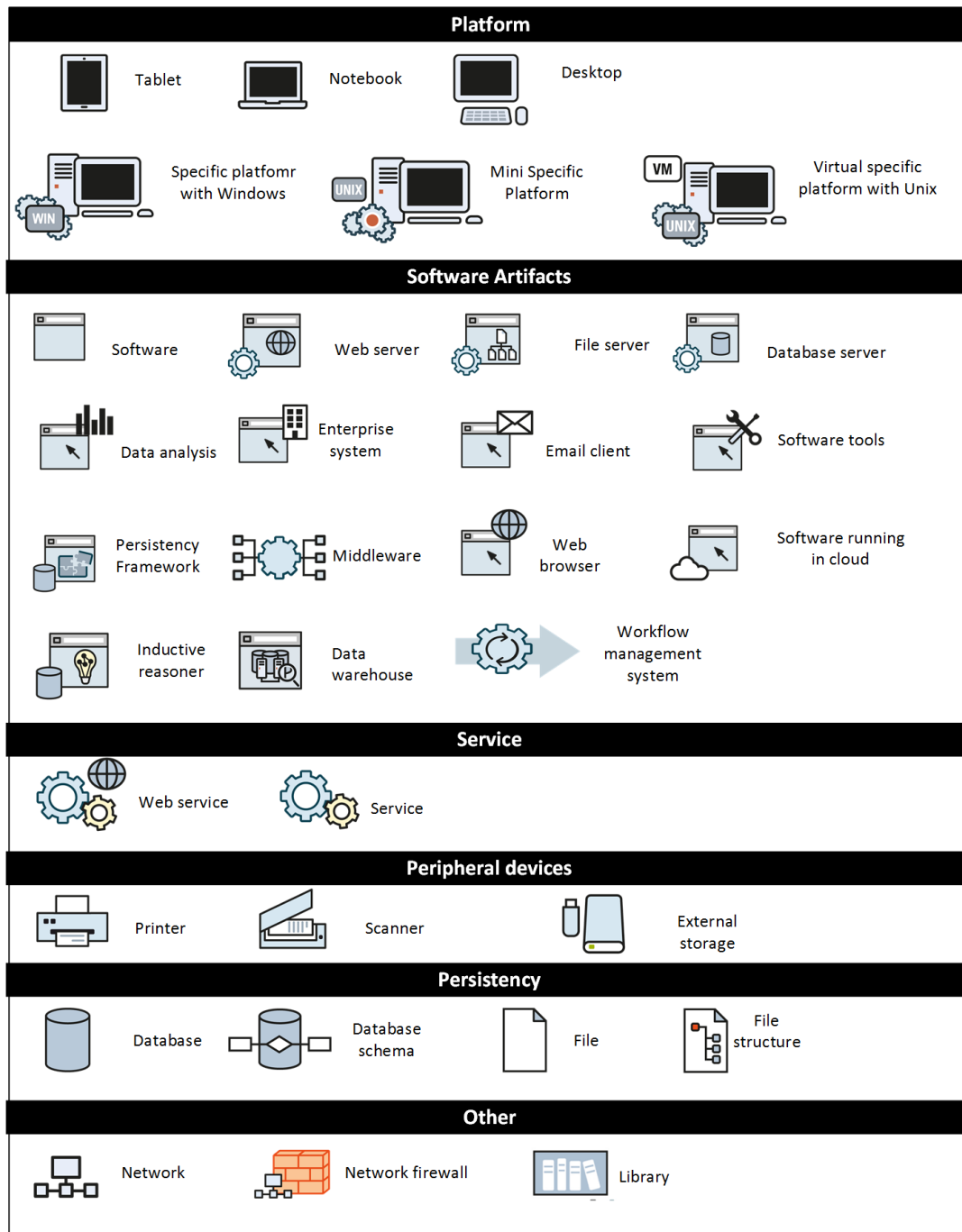


Figure 4.1: Selected concepts and their visualization, for a complete list of concepts, see Appendix A

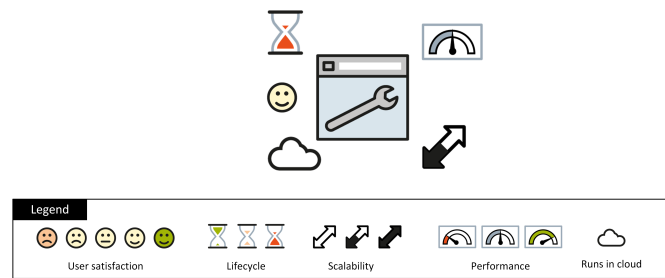


Figure 4.2: *Exemplary Decorations supporting dedicated analysis*

categories. Indeed, such a subdivision turned out to be necessary, taking into account the sheer number of concepts being software artifacts, as well as considering existing differences between those (e.g., cf. the visualization of Middleware vs. the visualization of Workflow Management System, Fig. 4.1).

Finally, please note that, as shown in Fig. 4.2, to support more specific analysis the basic symbols are enhanced with additional, context-specific symbols. For instance, in Fig. 4.2 we indicate that this is a mission critical application, being in use, and having a high user satisfaction and performance.

4.2 Selected ITML Diagram Types

The ITML proposed in this report allows for creating a few diagram types, among others, the IT Infrastructure Diagram, Topic Diagram and Architecture Diagram. Please note that each diagram type provides a different perspective on the IT infrastructure at hand, and/or shows the IT infrastructure in conjunction with the surrounding action system. In addition to that, each type is suited to represent both the current as well as targeted situation, cf. (Kinderen and Kaczmarek-Heß 2018). The main purposes, key analysis questions and concepts of selected diagram types are presented subsequently.

4.2.1 IT Infrastructure Diagram

Purpose: The main diagram type is the IT infrastructure diagram. It depicts elements of IT infrastructure and associations between them, and thus, supports basic analysis of IT landscape elements, IT-centric as well as integration. In addition, it allows to model an IT landscape and its connections to business process types as well as organizational structure, thus, the analyses of organizational assignment as well as IT-business alignment are enabled.

Key concepts: specific platform, software and its specializations, service, hardware device and its specializations, network etc., as well as relations: runs on, provides, uses, communications, access to etc.

Integrated with the following diagram types: Topic diagram, MEMO Organizational Chart, MEMO Business Process Control Flow Diagram.

The example in Figure 4.3 illustrates an IT Infrastructure of an exemplary company. The presented excerpt of the IT landscape diagram illustrates various dependencies between IT artifacts and services. It also points to the service orientation – the software applications are providing services/functionalities, which in turn may be used by enterprise stakeholders.

This view however, while it supports the basic analysis, cf. scenarios in Chapter 2, it does not provide detailed information regarding qualitative or quantitative aspects of IT artifacts. Therefore, the diagram in Fig. 4.4 shows a more detailed view on the exemplary model. Here

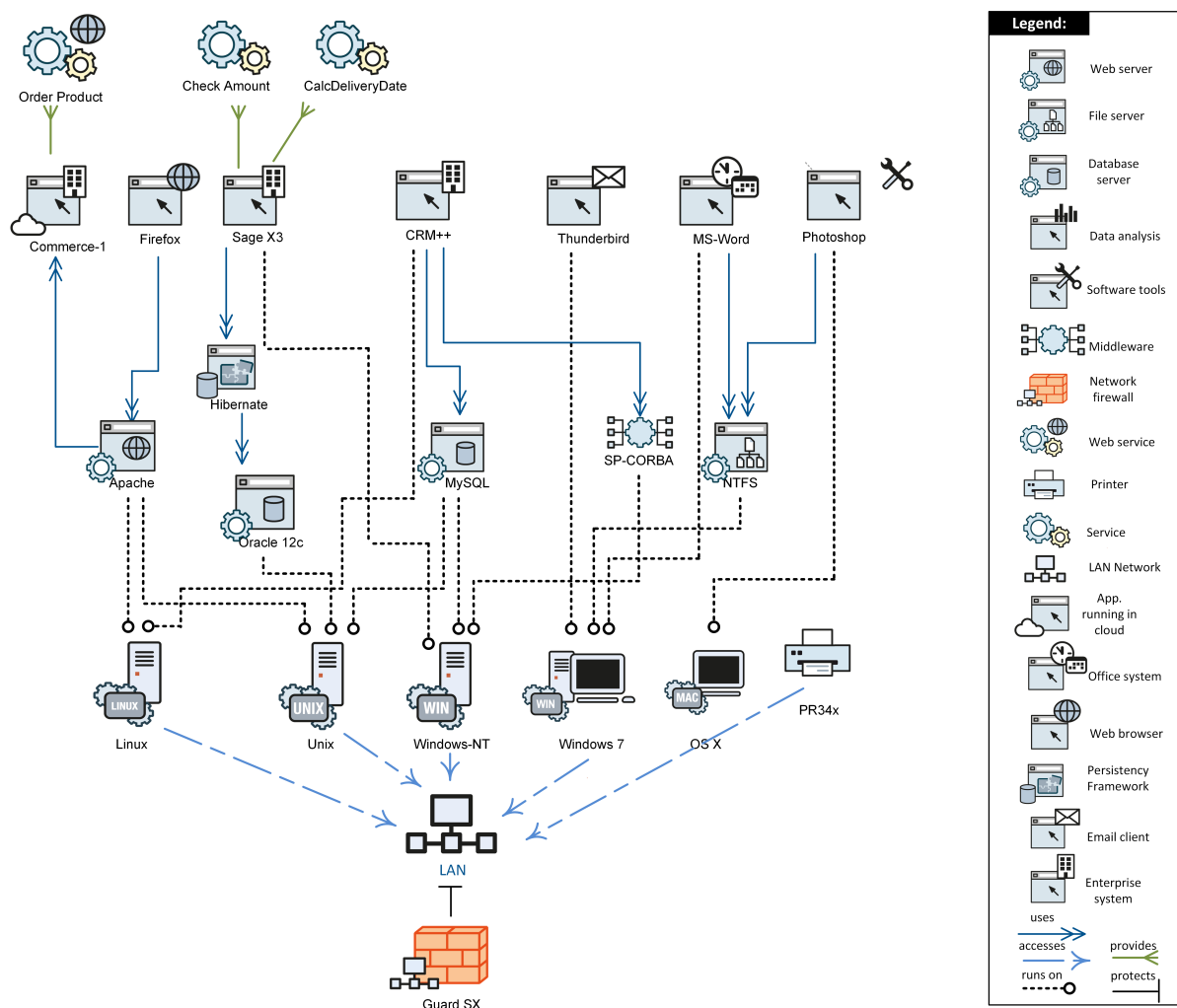


Figure 4.3: Exemplary Diagram

4 Language Design: Concrete Syntax and ITML Diagram Types

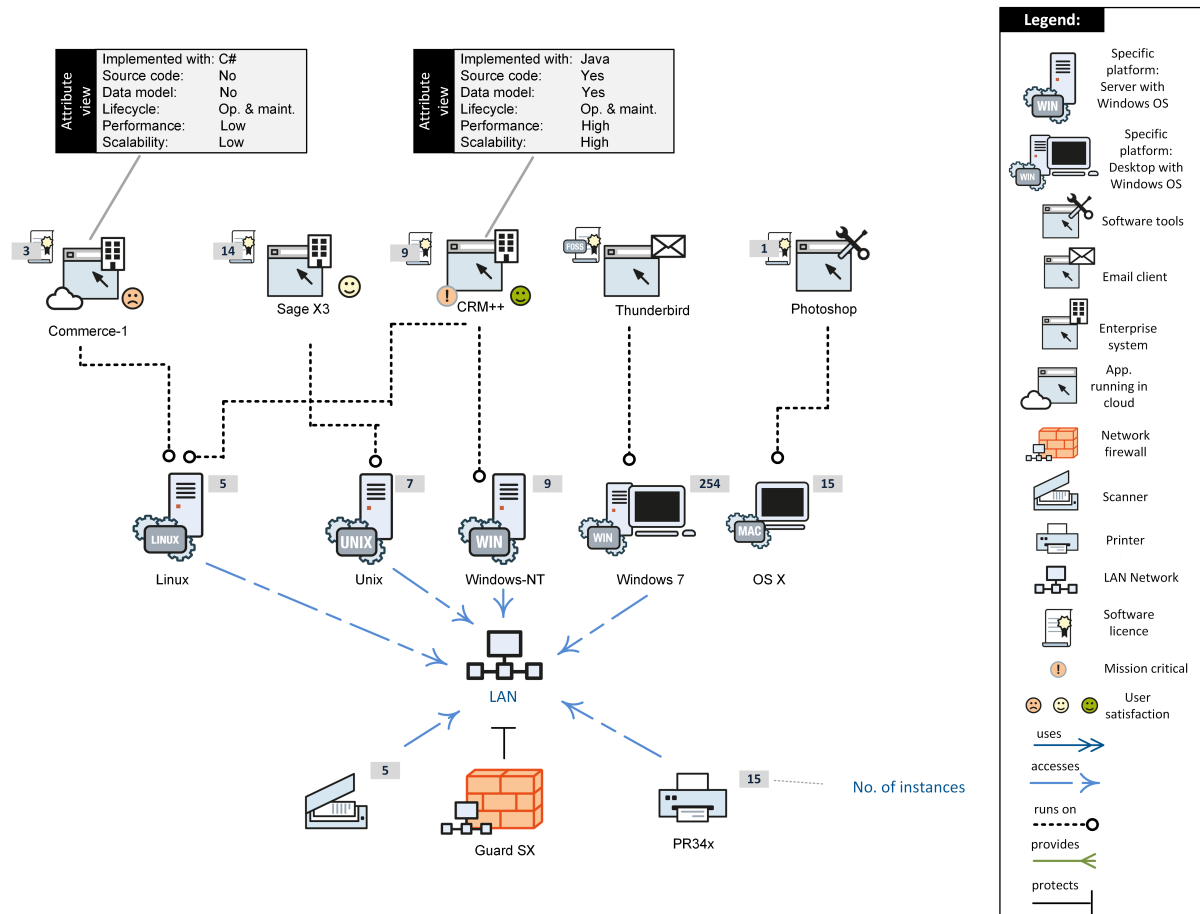


Figure 4.4: A selected detailed view

selected properties of individual model elements as well, if appropriate, also selected instance-level information, are provided. Which details should be represented and how exactly the information should be visualized/shown depends on the possibilities of a meta modeling platform used to create a corresponding modeling tool. For instance, the ADOxx meta modeling platform (Fill and Karagiannis 2013) that is used in our case to implement the ITML, allows to define different views on a model created, supporting different analysis scenarios (e.g., integration analysis vs. security analysis), as well as allows to capture relevant properties in form of so called Notebook.

A sample of typical analysis questions that can be answered on the basis of an IT Infrastructure diagram, as partly visualized in Fig. 4.3 and Fig. 4.4, are as follows:

- What is the heterogeneity of the IT infrastructure, in the sense of the different types of platforms and different types of operating systems?
 - Which types of platforms are there?
 - Which types of an operating system is running on a given platform type?

- Questions regarding the basic characteristics of a software, e.g.:
 - How many licenses for the given type of the application software are available?
 - What is the type of the license and its cost?
 - In which phase of the life-cycle is the given application?
- Which software artifacts have been implemented using some specific language?
- Which application types are characterized as critical ones?
- Questions regarding the basic characteristics of a platform type:
 - How many instances of a given platform type are there?
 - When it has been introduced?
 - Which operating system type does it have?
- Which software type runs on which platform types?
- Does some software application type offers some service types?
- What is the service-orientation level of our IT infrastructure?
- Which application types are running on the same type of a platform?
- Which types of peripheral devices are there in an enterprise?
- Do the server types belong to the same Local Area Network?
- Which databases are used by which application types?

Considering security-relevant concepts (e.g., firewall, network) and their attributes, as well as relevant relations (e.g., protects, access to), one may perform security related analysis as follows:

- What is the authorization type applied by some specific software artifact?
- What is the transport protocol used?
- Is the exchanged data encrypted? What is the encryption type?
- Is the network protected by a firewall?

Additional attributes of some software artifacts allow to assess qualitative aspects and answer the following questions:

- What is the average user satisfaction with some software type?

4 Language Design: Concrete Syntax and ITML Diagram Types

- What is the average availability and average reliability of some software type?
- What is the code complexity of application software types being used in the enterprise?
- How one is assessing its scalability?

Finally, the IT infrastructure diagram allows also to conduct integration analysis. For instance, looking at Fig. 4.3, as the two DBMS are only used by one software application each, it can be assumed that the data used or created by those applications is not integrated. Nevertheless, even if two applications would use the same DBMS, it would not necessarily follow that the data used in each case is integrated. Rather, it would have to be clarified whether both applications use the same database, and thus, the same schema. The same applies to software artifacts that use the same file management system: only if they also use the same files, they might be accessing the same data. A differentiated analysis would require consideration of complementary aspects, such as a data model or database schema, so as to determine which data is read or written by each software. It should also be considered what is the level of semantics of the used classes or data types, as it makes a difference to the quality of integration whether two software artifacts only exchange data in the form of bytes or whether they refer to common classes.

In addition to persistent data, integration also affects the exchange of volatile data between software artifacts. An indication of the possibility of such an exchange occurs when the systems in question are connected to the same middleware system. However, that does not necessarily mean that data between the systems can be actually exchanged. Here, it would need to be first determined whether the systems access the same interface repository of the respective middleware. In an interface repository all interfaces are listed, that may be used to communicate with the middleware and other software artifacts using it. To be able to evaluate the integration it would be necessary to determine which internal classes of each involved software artifacts are accounted for in the interfaces in the repository. These details are not included in an IT infrastructure model, but may be obtained by analysing the respective interface repositories.

The diagram in Fig. 4.5 focuses on data integration in an IT infrastructure. It shows how application systems access databases and files. Regarding the former, the structure of the data is indicated by a reference to a database schema. If the IT infrastructure model is managed using a corresponding modeling tool, it may be possible to navigate from a symbol representing a database schema to a representation of the schema, or to navigate to a diagram that contains, e.g., a corresponding data model. In addition, the diagram in Fig. 4.5 shows two further integration relevant aspects, namely a shared access to data stored in files and access to an interface repository. To assess the quality of integration also here additional analyses would be required, among others, of the definition of a file structure as well as the content of the interface repository.

Please note that the final aim of the conducted integration analysis is to answer the question: What is the current level of integration and where is the potential for further integration? Where the lack of integration may threaten the data integrity? Finally, please note that in order to examine integration deficits, among others, the ITML offers the concept of topics, discussed in subsequent section.

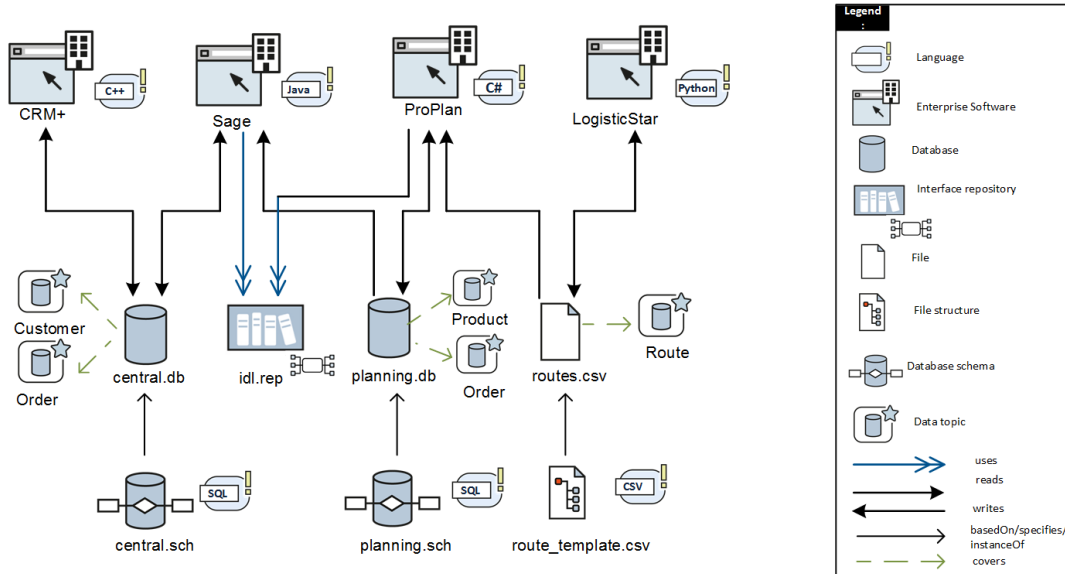


Figure 4.5: *Integration analysis*

4.2.2 Topic Diagram and Corresponding Analysis

Purpose: This diagram type allows to define Function Topics, Event Topics, and Data Topics (cf. Fig. 4.6). It is also possible to define different relations between them, e.g., that a data topic *uses* a data topic, as well as data topic *includes* other data topic.



Figure 4.6: *Concrete Syntax: Topics*

Please note that the modeled Topics are being referenced within the IT Infrastructure Diagram. Thus, one may perform analysis regarding different topics distinguished as well as identify (using the IT Infrastructure Diagram) which elements of IT infrastructure read or write some specific topic. This allows to perform basic integration analysis, as is also indicated later in this section.

Key concepts: Data Topic, Function Topic, Event Topic, uses, includes

Integrated with the following diagrams: IT Infrastructure Diagram

Exemplary questions include:

- What data/function/event topics exist?
- What are dependencies among various topics?
- If used together with an IT Infrastructure Diagram, what integration gaps and deficits may be identified?

Fig. 4.7 shows an exemplary “Data Topic Map” created based on the collected data (at least partially modeled in the IT Infrastructure diagram). First data topics need to be collected, then it is examined in pairs, whether one data topic needs the other. For example, the representation of an order requires a reference to a customer. Then, for each application system is to be clarified what read and write access to which data topics it has. Subsequently files and databases used as well as topics they cover would need to be analyzed as well. Please note here, as already mentioned by taking advantage of the modeled topics, one may perform analysis supporting static, functional and dynamic integration of IT landscape. For instance, by analyzing covered and required functions (function topics) and read and written data (data topics), one may identify similarities between some software artifacts (e.g., applications offering the same functionality, or if two software applications are using the same data topics, then there is a data similarity relation between them).

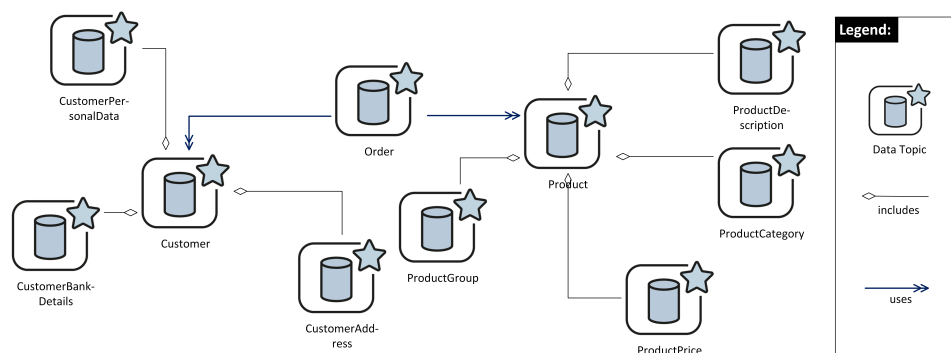


Figure 4.7: Exemplary Data Topics

4.2.3 Architecture Diagram

Purpose: An IT architecture diagram, compared to other diagram types discussed, features a higher degree of aggregation, which is made possible by using various specializations of the language concept Clusters.

Key concepts: Cluster, Architecture Pattern, architecture dependency

Integrated with the following diagram types: IT Infrastructure Diagram

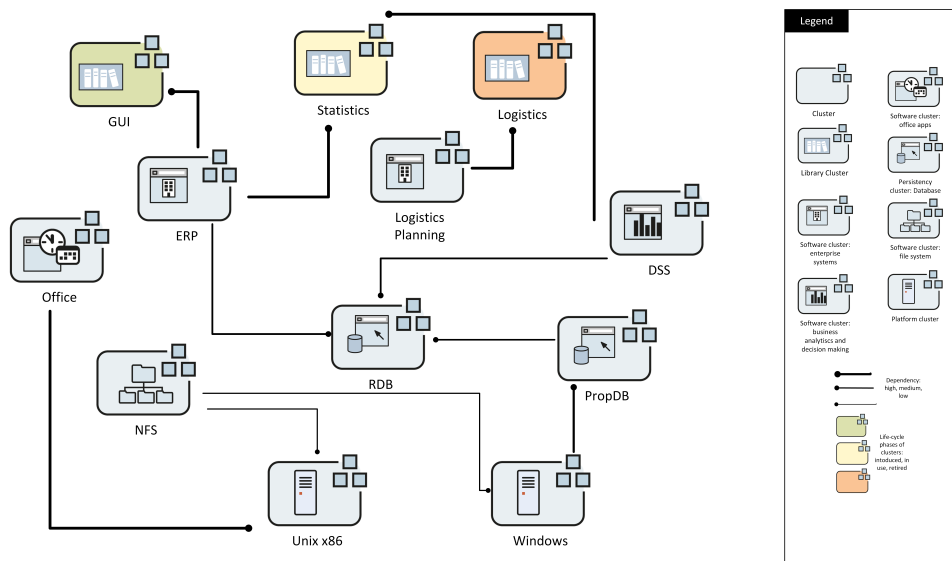


Figure 4.8: Exemplary, Conventional Architecture

Figure 4.8 shows a rather conventional IT architecture represented in line with the layered approach. Various types of application software clusters use clusters of software libraries and persistence systems. Software as well as persistence clusters are each more or less dependent on platform clusters. It needs to be stressed that through the aggregation to clusters the representation of the structure of IT infrastructure is made clearer, but at the cost of a loss of information. By using different background colors we may represent the phase in the life cycle of a cluster. If a more detailed view is desired, a cluster can be “decomposed” so that it individual elements can be investigated.

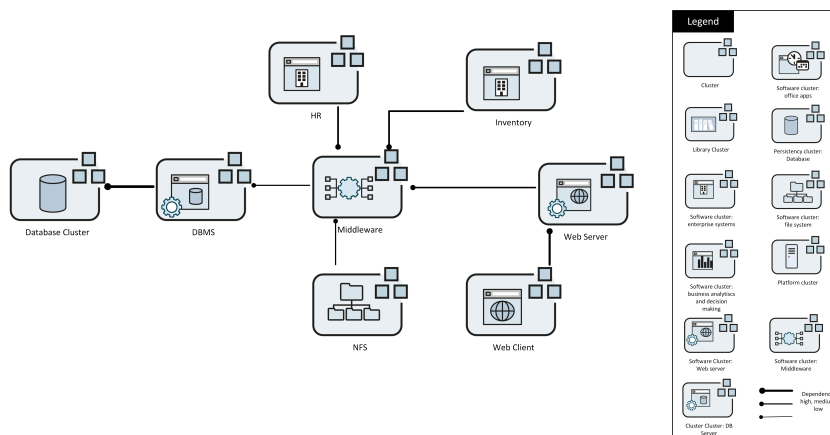


Figure 4.9: Architecture Example: Distribution and Integration

In turn, Fig. 4.9 represents an architecture focusing on distribution and integration aspects. The main focus is placed on a middleware cluster, which encompasses several middleware systems which, among other, include also application servers. Also here, the architecture diagram

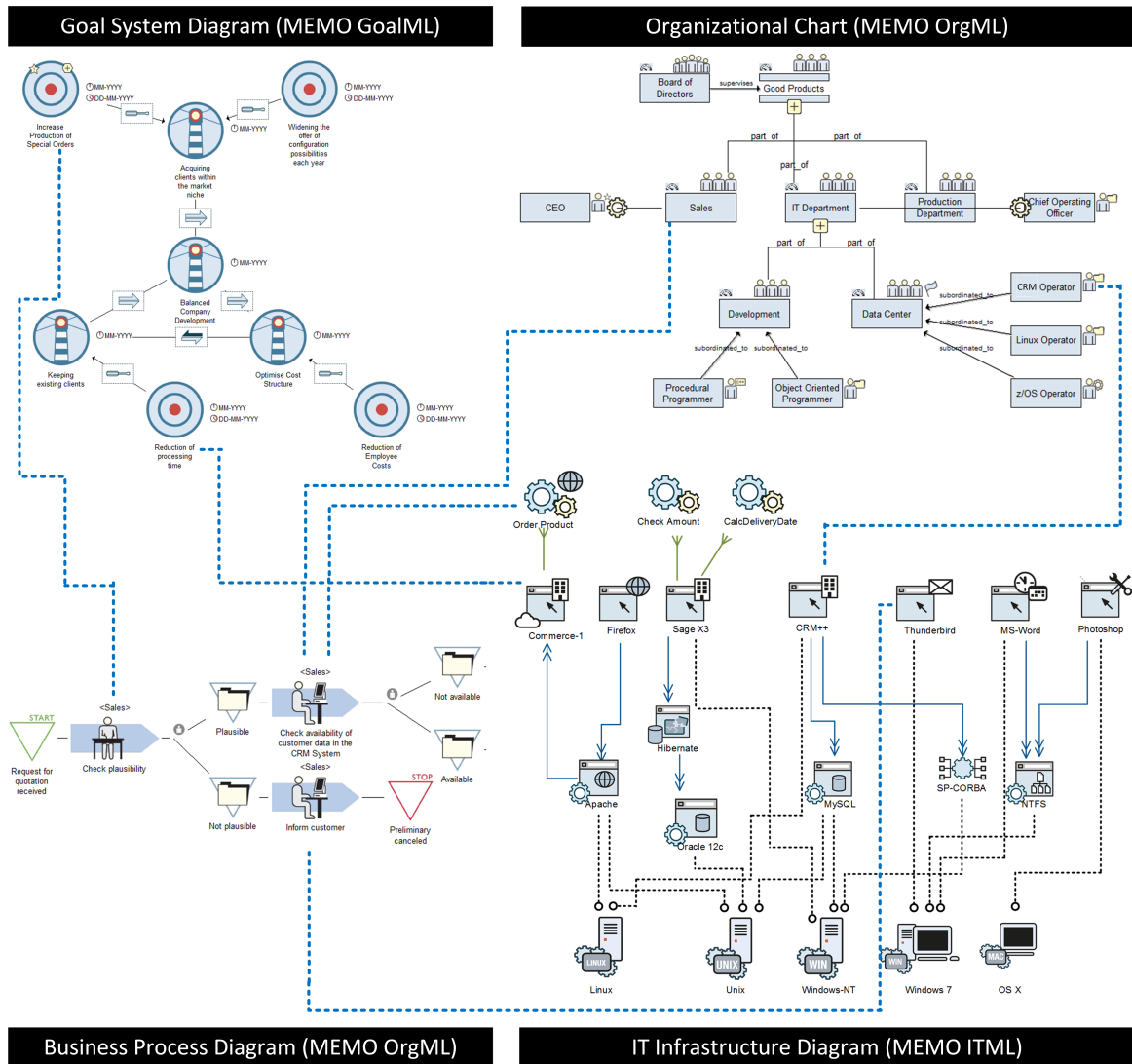


Figure 4.10: Integrated view: example

provides a high-level view on the IT landscape only and allows to obtain an overview of the most important elements and existing dependencies. Nevertheless, in order to conduct further analysis, there is a need to drill down in order to assess the relevant details.

4.3 Integration of ITML Diagram Types With Other MEMO Diagram Types

Next to individual diagram types that each express a different view on the IT landscape discussed so far, the ITML also offers concepts for establishing a relation to diagram types from other MEMO DSMLs, which each express a different view on the enterprise action system.

An example of such a concept is *AnyProcess*, cf. (Frank 2014b), which can be used to relate an ITML diagram type to a business process created in OrgML for business processes.

Establishing such relations between the ITML and diagram types from other MEMO DSMLs allows for analyses that cut across different views on an organization. Such cross cutting analyses can be illustrated on the basis of the integrated diagram in Figure 4.10. Here, we find a relationship between the ITML and *the organizational structure diagram*. In particular, a relation is established between the “CRM” application software and the role “CRM operator”. For a given element of the IT landscape, such a relation to the organizational structure can be used to assign, e.g., responsibilities or various access rights to roles or particular actors within the organizational structure.

Next, looking at a relationship between the ITML and *the business process diagram*, in Fig. 4.10 we see that a relation is established between the “Thunderbird” and the business process “Inform customer”, and the services “OrderProduct” and the business process “Check availability of customer data in the CRM system”. For a given element of the IT landscape, such a relation allows for, prominently, an impact of change analysis. For example, if the status of the software would be changed, either through a direct modification, or through a change in the platform on which the software runs, what business processes in the organization will be affected? Thus, we can answer questions such as: What is the characteristics of the support provided by some application/service types to some process type/process type element? Which process types in the organization are depending highly on the IT infrastructure?

Finally, looking at the established links with *the goal system diagram*, we may see, e.g., which elements of the IT infrastructure support organizational goals and what is the characteristics of such a support (e.g., “Commerce-1” supports realization of the goal “Reduction of processing time”, cf. Fig. 4.10).

The integration of ITML with other MEMO diagram types can also be used for other kinds of analysis. For example, inspired by Lankhorst (2013) the relation between an IT landscape diagram and a business process diagram can equally be used for quantitative analysis of the resources load on the IT landscape, given a certain amount of business processes executions.

4.4 Selected Design Principles

Although in this research report we focus mainly on the designed language, i.e., ITML, and do not explicitly address other elements of the modeling method (i.e., process models guiding the application of the proposed language), in what follows we discuss a few selected guidelines that should be followed while (re)designing IT infrastructures or describing and analyzing already existing ones.

While describing the existing IT infrastructure (e.g., by creating the IT Infrastructure Diagram), the following three guidelines should be considered.

Guideline 1: *Keeping focus and reducing complexity* – An IT infrastructure may be of great complexity, which manifests itself in a considerable heterogeneity of artifact types, a substantial number of hardware components and software installations, and diverse relationships among them. Therefore, an important goal of modeling of an existing IT infrastructure is to reduce this complexity in order to support the users of models in grasping the aspects that are essential for a particular problem. To do this, it is advisable to provide a list of the requirements for a specific problem, and use it to derive relevant questions. Against this background, a model should be designed in such a way that it is suitable to answer these questions or to support the answering of these questions. At the same time, modeling of additional aspects should be avoided.

Guideline 2: *Differentiation of perspectives* – IT infrastructure models can be aimed at different target groups, each having some specific professional perspective. IT managers, for example, tend to be less interested in technical details than IT architects or software developers. At the same time, the IT infrastructure models can also be used as a medium to promote communication between different target groups. For this purpose, a suitable intersection of the relevant aspects for the target groups involved should be determined.

Guideline 3: *Consideration of selected aspects of the organizational action system* – Even if there is a significant number of questions solely focusing on the IT infrastructure and its characteristics, the IT infrastructure should not be seen as an end in itself and considered in isolation from other aspects. Therefore, in many cases it is advisable to consider the relevant context, among others, models of the organizational action system, e.g., business process models or goal models. Here, too, one will focus on the questions that are important for certain analysis purposes (cf. Guideline 1) in order to deduce which aspects are to be included and in which way.

The following ten guidelines should be considered while (re-)designing and analysing IT infrastructure. Please note that we focus here on a few general guidelines only, which refrain from the fact that the scope for design is often considerably restricted.

Guideline 4: *Reuse* – With regard to the design of software systems, it is necessary to carefully analyze commonalities and map them using suitable abstractions. If ready-made software is used, reuse requires an open access to the functionalities that software offers and to the resources that it manages, so that they may be used by other systems. This concerns their syntactic representation and their semantics. The investigation of reuse possibilities requires consideration of already installed systems, as well as the possible use of ‘mediator’ systems, such as middleware or persistence frameworks.

Guideline 5: *Integration* – A lack of integration is associated with redundancy and complex, risky communication between software systems. It thus poses a threat to the efficiency and integrity of IT infrastructure. Next to the integration of data (static integration), the functional and dynamic (process) integration must also be taken into account. In any case, the integration of two software systems requires common concepts such as common data types or classes, common functional interfaces and common event types: this is the only way that an application can use data, operations or events from another application. In addition, it must be taken into account that common namespaces for the exchange of individual instances (more precisely: of references to these) exist or are created. This can be implemented, for example, by a database whose namespace is presented in the same way to the accessing software systems. In the case of objects, operations and events, however, this is associated with considerable challenges, which can usually only be partially overcome. One approach to meet these challenges is to plan an evolutionary approach, and, e.g., plan that initially only a few applications will be integrated through databases or middleware systems. Systems to be introduced in the future should then be selected in such a way that they can be connected via these databases and middleware systems.

Guideline 6: *Dynamic integration* – Two software systems are dynamically integrated, if they have common event types and can access corresponding event instances that were generated in the other system. Then, it becomes possible to run business processes supported by those integrated software systems. However, it will most likely make more sense to define and execute the process in a dedicated system, such as a workflow management system (WFMS). The dynamic integration can then be limited to the integration of the involved software systems with the WFMS. This results in the design guideline that, when selecting software systems, care must be taken to ensure that events that may be significant for the control of a process can be generated and stored in an event space that is also accessible to other software systems. Such an event space could for example be offered by a middleware system.

Guideline 7: *Synchronization* – Subsequent integration is usually only possible to a modest extent. There may also be reasons for deliberately keeping data redundant, such as high performance requirements. In such cases, the regular synchronization of data is a measure to keep the integrity-threatening effect of redundancy within limits.

Guideline 8: *Adaptability* – It can be assumed that the requirements for an IT infrastructure will change in time. It should therefore be designed in such a way that it does not hinder any necessary adjustments. That applies to the effort and risk associated with changes. To this end, it is advisable to first differentiate between the more likely invariant parts and the parts of the infrastructure that are likely to change over time. The architecture of an IT infrastructure should be designed in such a way that variant parts can be dependent on invariants, but invariants, conversely, are not dependent on variants. In view of the various imponderables that affect future technical developments, the competitive situation, the development of market

structures and much more, one must operate here with assumptions. In order to develop such assumptions, it is useful to consider questions such as “What are invariant aspects of our business model?”, “Will our product range change and how would this possibly affect the requirements for the IT infrastructure?”, “What will our service creation processes look like in the future?”, “With which actors will we work in the future?”. In addition, it must be assessed whether and to what extent individual systems can be adapted if required. The adaptability of an artifact, of a hardware or software system, depends on whether it is designed to address a set of very specific requirements only, i.e., how specialized it is and what is the range of its possible uses. For instance, application systems that address specific requirements can, under certain circumstances, cover a considerable range of uses, for example if they have powerful abstractions that support convenient and secure adaptations to changed requirements. Furthermore, if the common features of several software systems are bundled in a jointly usable artifact (which thus represents an abstraction from these systems), adaptability is promoted when a modification of these shareable artifacts allows to address the change in requirements. Finally, the adaptability of systems can also be considered with regard to the respective provider. This concerns questions about the ability and willingness to make adjustments if necessary, as well as the associated costs.

Guideline 9: *Integrity* – The benefit of an IT infrastructure depends largely on the integrity of the data it manages and the operations offered. Put simply, integrity is a measure of how much users can rely on the answers of the systems involved. On the one hand, this affects their consistency. For example, two simultaneous questions about a customer’s address should not lead to different answers. The integration of the application systems is an indicator of consistency. Low integration goes hand in hand with redundancy, which in turn jeopardizes consistency. It also depends on the extent to which the software systems involved correctly meet the respective requirements.

Guideline 10: *Investment protection* – The financial aspects associated with the creation or acquisition and implementation of software systems can be considerable. That is why protecting the investments made in terms of profitability is an important factor. In addition to the already considered adaptability of systems, the planned useful lifetime, the reputation and future prospects of the respective provider or, in the case of open software systems, the developers community, and the application of standards must be considered. The protection of investments in software systems that are maintained in-house depends on the availability of suitably qualified employees.

Guideline 11: *Tight vs. loose coupling* – To promote reuse and adaptability, the design principle of loose coupling is often recommended. Two systems are loosely coupled when they communicate via an interface that is relatively generic. An interface that only requires strings as input and also delivers the result as a string is less specific than an interface that requires objects of a class that has been specified for a certain context of use, e.g., a class such as “invoice”. Loose

coupling is therefore aimed at reducing the dependency between system parts. In this way, it should be easier to adapt the infrastructure by replacing individual components, compared to the case of tight coupling. On closer inspection, however, the recommendation to strive for loose coupling as a matter of principle turns out to be problematic. Loose coupling represents a threat to the efficiency and integrity of the software systems involved: Internal data must be transformed into the semantically “flattened” data of the interfaces and transformed back into internal data structures at the recipient, which is associated with effort and risk. At the same time, tight coupling does not pose a problem for the adaptability of an infrastructure, if the interface used for this has a high degree of invariance. In addition, tight coupling of several software systems to one system component can promote the adaptability of an infrastructure, if future changes are limited to these system components, i.e., the coupled software systems can abstract from such changes. This applies, for example, to the interfaces through which software systems access storage media or peripheral devices. If the technical access changes, the necessary adjustments are limited to the affected parts of the operating system. In general, the following rules should be observed: An artifact A1 should not come, if possible, from another artifact, that is less invariant than A1 over time. If this rule can be met, i.e., if there is reliable knowledge on both artifacts with regard to their changes, then tight coupling is, all other things being equal, a good option. Otherwise, a loose coupling is more likely. The appropriate degree of coupling must be checked on a case-by-case basis and should be documented, e.g., in architectural patterns.

Guideline 12: *Scalability* – If the load that an IT infrastructure has to cope with changes over time, the capacity (computing power, storage volume, number of computer workstations, bandwidth) must be adapted. The lower the effort involved, the better the scalability of the IT infrastructure. In order to support the scalability, the applications whose capacity requirements can be expected to change significantly over time must first be identified. The adjustment of the capacity should be possible on a stepwise basis, inline with the changing requirements; and should be monotonous, i.e., have no impact on the existing installations. However, adjustments that require the installation of additional hardware cannot be varied as desired. It may therefore be advisable to use external providers of platforms or software services that enable parts of the IT infrastructure to be encapsulated in the cloud.

Guideline 13: *Evolutionary development* – In many companies, an IT infrastructure has developed over time that suffers from considerable weaknesses. These include insufficient integration and adaptability, unsatisfactory functionality and cumbersome user interfaces. Against this background, the question emerges what options IT management has for such a situation. On the one hand, selective improvements are conceivable. They can be achieved by replacing individual legacy systems or by adding components that allow to hide the existing heterogeneity. The introduction of middleware systems or data warehouse systems fall into this category. Finally, a radical step is also conceivable, aiming at redesigning the entire IT

infrastructure. Even if the last option is most suitable to significantly improve the performance of an IT infrastructure, it is associated with considerable risks. A planned evolutionary development of the IT infrastructure offers the opportunity to improve the IT infrastructure in a targeted manner and to keep the risks within limits. In addition to a long-term vision, development stages must be modeled, each of which builds on one another, in order to protect the investments and further development of the IT infrastructure.

5 Conclusions

In this report, we present the ITML, a Domain-Specific Language for the modeling of IT infrastructures. The specification of the ITML reflects the professional language used for (the description and analysis of) IT infrastructures, such as various types of hardware, software, or networking infrastructure. To illustrate the use of the ITML, we present selected core diagram types. We also discuss typical IT infrastructure analysis scenarios, in terms of dedicated IT infrastructure scenarios, like the availability or performance of given IT infrastructure elements, and scenarios that cut across different perspectives on the enterprise action system, like IT-business alignment in terms of how the IT infrastructure supports business processes. In addition, from these scenarios we derive requirements that inform the specification of the ITML.

The presented ITML has been implemented with the meta modeling software environment ADOxx (Fill and Karagiannis 2013) as part of MEMO4ADO¹ (Bock and Frank 2016). However, due to the lack of support for intrinsic features and language level types in the ADOxx meta meta model (Fill and Karagiannis 2013)², in order to implement the ITML in ADOxx a redesign of the ITML meta model has been required, so that the desired domain aspects could all be modeled at exactly the same abstraction level, cf. (Bock and Frank 2016). Those modifications encompass, similarly to those already reported while implementing other MEMO languages, cf. (Bock and Frank 2016), among others the following aspects: (1) Omitting several concepts, e.g., those describing advanced domain details of interest only to a narrow group of domain stakeholders, e.g., middleware repository, (2) Simplifying concepts, e.g., subsuming some concerns in one central concept to foster accessibility and ease of use (e.g., aspects related to service contracts), (3) Modifying attributes to reduce complexity or to allow for holding references to elements modeled in other diagrams (attributes of type ‘Interref’) (e.g., when accounting for organizational assignment), (4) Adjusting abstraction levels in case of intrinsic attributes (or relations) or dropping them, in case it was not possible (e.g., `buildIn` association between a `Case` and a `HardwarePlatform`). Please note that dropping intrinsic elements by default would not be desirable, as in the ITML various important domain aspects are captured at that level. Finally, to foster reuse, in addition to the main diagram types mentioned in

¹The modeling tool can be downloaded from <http://www.omilab.org/memo4ado>

²The ADOxx meta meta model includes concepts to define meta classes, attributes, and relationships at level M_2 , which can be instantiated at type level M_1 in the ADOxx Modeling Toolkit (Fill and Karagiannis 2013, pp. 67). Yet, it does not support instantiating and managing instance populations at level M_0 that would represent instantiations of model elements from the level M_1 .

Chapter 4, additional diagram types have been defined, e.g., (1) ITML Languages Diagram, which allows to model different languages (and their characteristics). The defined concepts are referenced within the IT Infrastructure Diagram type; (2) ITML File Exchange Format Diagram, which allows to model different types of file exchange that specify the format of the files being exchanged, which may be then referenced from the level of the main diagram; or (3) ITML Library Diagram allows to define libraries used, which again are referenced from the main diagram.

As illustrated in this research report and in the proof-of-concept implementation as part of MEMO4ADO, we can use the created ITML models to conduct additional analyses compared to the original ITML. However, while designing a new DSML for modeling IT Infrastructure, we have faced modeling challenges and limitations, partly imposed by conventional meta modeling language architecture, we have been following, cf. (Frank 2014a; Kaczmarek-Heß and Kinderen 2017). Especially the following aspects we do not find satisfactorily addressed in the current version of the ITML.

Type or Instance: In the IT domain, differentiation between types and instances is not trivial, and it remains often unclear whether a real-world entity should be represented as a modeling concept (i.e., a type) or as its application (i.e., an instance of a modeling concept). Indeed, cf. also (Kaczmarek and Kinderen 2016; Kaczmarek-Heß and Kinderen 2017), it is not clear whether a software artifact like “Windows Operating System” should be treated as a type or as an instance. Similarly, it is unclear if a particular version of “Windows Operating System” is a type or an instance, or even a specific exemplar with the assigned license number installed on some piece of hardware. As in the MML language architecture there is no ‘perfect’ solution to the above mentioned challenge, therefore, in the proposed conceptualization we have decided to restrict the meta types to a few rather generic ones, which are necessary from the perspective of the goal pursued.

A fixed number of classification levels: IT artifacts exist in a remarkable variety of types, each of them possessing a variety of type-specific attributes and further hierarchies. In order to avoid conceptual redundancy, we are interested in making this hierarchy part of a language specification. Thus, we model, e.g., a Server and specialize it in, e.g., WebServer. However, by deciding to represent the refinement relations between software artifacts as specializations, we are dealing with a so called level mismatch problem (Atkinson and Kühne 2008), as various domain levels/hierarchy levels are mapped onto exactly the same model level. While it is certainly technically possible to overload the (M_2) level, by relying on specialization, this constitutes a workaround. Such workarounds are necessary since the ITML, being based on the traditional two-level paradigm natively does not offer constructs to mirror the hierarchies that naturally exist in the IT infrastructure domain.

Contingent Classification: As we aim at accounting for a variety of IT artifacts belonging to the enterprise IT landscape, a classification of different generic concepts like software, platform or hardware devices turned out to be necessary. However, all of those can be categorized in contingent ways, e.g., with regard to their primary purpose, architecture, visibility to a user. From a perspective of conventional modeling, as already mentioned, such differentiation can be modeled either by using relevant meta types, using generalization/specialization, using an enumeration attribute of the generic meta type, or as a role. Taking into account the pursued goals as well as, among others, considering avoiding conceptual redundancy, none of those options are fully satisfactory, cf. also (Kaczmarek-Heß and Kinderen 2017).

Problems with incorporating relevant information: As we use the specialization relation to model the variety of IT artifacts (cf. the software hierarchy), moving along it we are able to extend the definition of specialized meta types. However, we face problems when trying to incorporate relevant information, in particular when trying to assign values to attributes of meta types. This is because in conventional meta modeling meta types cannot have a state.

No associations between objects on different levels: Within conventional meta modeling the only relation allowed between different levels is the instantiation relation (Atkinson and Kühne 2008), and objects assigned to different classification levels cannot be jointly represented within one body of model. As a result, we cannot link a concept defined as part of language specification with a concept being part of language application. This leads often to redundancy in corresponding models, as the information not stated in the language specification needs to be added during the use of the language.

Summarizing, we see that the identified requirements for IT infrastructure modeling languages cannot be satisfactorily addressed with a conventional language architecture. A satisfactory solution is understood as a solution that would promote model integrity, would avoid conceptual redundancy, and allow to express all relevant knowledge at the relevant classification level. We argue that this is due to limitations imposed by (a) mainstream object-oriented programming languages used to implement the corresponding modeling tools, and (b) relying on the traditional two-level conceptual modeling, for details see (Kaczmarek-Heß and Kinderen 2017).






























Therefore, it seems reasonable to undertake an attempt to apply an alternative language architecture and use a multi-level modeling language like, e.g., Flexible Meta Modeling and Execution Language (FMML^x) (Frank 2014a). The term multi-level modeling (MLM) covers any modeling approach that aims to provide systematic support for representing multiple classification levels within a single body of model content (Atkinson and Kühne 2001). Multi-level modeling has been steadily increasing in importance, and works exist focusing on how, when, and why to use multi-level modeling. For instance, while (Lara, Guerra, and Cuadrado 2014) argue that, for different use cases, application of MLM may lead to a more ‘accurate’

and ‘simpler’ representation of a domain than conventional approaches, and thus, reduce so-called accidental complexity (Atkinson and Kühne 2008), Kinderen and Kaczmarek-Heß (2021) point to MLM having an ability to naturally mirror domain hierarchies, and, e.g., Frank (2014a, p. 335) argues that multi-level modeling fosters users’ empowerment by providing them with concepts they are familiar with.

Indeed, MLM approaches come with a promise that, by allowing to use multiple classification levels as well as relaxing the type/instance dichotomy, a closer correspondence of modeling language concepts to concepts of natural language may be reached. As indicated in our previous work, cf. (Frank 2016) and (Kaczmarek-Heß and Kinderen 2017), especially FMML^x, among others, offering a common representation of model and code, seems to be suited to account for the specifics of modeling IT infrastructures. Therefore, our future research will focus on a multi-level reconstruction of all MEMO modeling languages, including the ITML.

A Concrete Syntax – Concepts

Table A.1: *ITML Concrete Syntax – Concepts*

Concept	Symbol	Concept	Symbol
Case			
Rack		Rack mounted	
Mobile case: Laptop		Mobile case: Tablet	
Stationary Case: Tower		Stationary Case: Mini	
Stationary Case: Workstation		Stationary Case: All in One	
Platform			
Specific Platform		Virtual Specific Platform	
Mini Specific Platform		Virtual Mini Specific Platform	
Hardware Platform		Virtual Platform	
Location			
Hardware Device and Supporting Artefacts			
Router		External storage	
Scanner		Printer	
Software and Supporting Artefacts			
Software		Lifecycle: early stage	
Lifecycle: medium stage		Lifecycle: retirement	
Software: mission critical		Software: custom made	
Software: low performance		Software: medium performance	
Software: high performance		Software: low scalability	

Continued on next page

































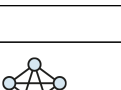



A Concrete Syntax – Concepts

Table A.1 – Continued from previous page

Concept	Symbol	Concept	Symbol
Software: medium scalability		Software: high scalability	
Software: source code available		Software: source code not available	
Operating System		Operating System Kernel	
Operating System: embedded		Operating System: real time	
Operating System: multiuser		Application System	
File System		Virtual Machine	
Component		Container Orchestrator	
Container		Container Group	
Container Engine		Web client	
Middleware		Workflow Management System	
Workflow Schema		Data Center	
Persistency Framework		Database Management System	
Firewall		Personal Firewall	
Network Firewall		Server	
Database management server		File server	
Web server		Application server	
Application Software		Application Software: Data Analysis	
Application Software: Email		Application Software: Enterprise Software	
Application Software: Office		Application Software: Tools	
Application Software: Web browser		Application Software: Runs in cloud	

Continued on next page

Table A.1 – Continued from previous page

Concept	Symbol	Concept	Symbol
Application Software: Low user satisfaction		Application Software: Medium user satisfaction	
Application Software: High user satisfaction		Data input	
Inductive reasoner		Distributed Ledger	
Data warehouse		IT Management Tool	
Database		Meta Repository	
OLAP Database		Database schema	
Extraction process		Extraction process: auto- mated	
Extraction process: executed, time span		Data structure	
File		Container Image File	
Executable File		Image File	
Image File: compressed		Archive File	
File Structure		File Exchange Format	
Library		Vendor	
Agreement		User Agreement	
Use Agreement		License	
Prop License		FOSS License	
Middleware Interface Repository		API	
Network			
Network		LAN	

Continued on next page



A Concrete Syntax – Concepts

Table A.1 – Continued from previous page

Concept	Symbol	Concept	Symbol
WLAN		WAN	
Service			
Service		Web Service	
Service: External Customer		Service: Internal Customer	
Service: External		Service: Runs in cloud	
Service: Supporting		Service Contract	
Service level			
Language			
Language		Specific language	
Model			
Conceptual model		Data model	
Object model		Process model	
Topic			
Data Topic		Function Topic	
Event Topic			
IT Architecture			
Cluster		Platform Cluster	
Persistency Cluster		Software Cluster	
Peripherals Cluster		Service Cluster	
IS Architecture		Scalability: low	
Scalability: medium		Scalability: high	
Expected changeability: low		Expected changeability: medium	
Expected changeability: high		Internal coupling: low	

Continued on next page

Table A.1 – Continued from previous page

Concept	Symbol	Concept	Symbol
Internal coupling: medium		Internal coupling: high	

Bibliography

- Atkinson, Colin and Thomas Kühne (2001). "The Essence of Multilevel Metamodeling". In: *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*. London, UK, UK: Springer-Verlag, pp. 19–33. ISBN: 3-540-42667-1.
- Atkinson, Colin and Thomas Kühne (2008). "Reducing accidental complexity in domain models". In: *SoSyM 7.3*, pp. 345–359. ISSN: 1619-1374. DOI: 10.1007/s10270-007-0061-0. URL: <http://dx.doi.org/10.1007/s10270-007-0061-0>.
- Barber, David (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- Bengtsson, PerOlof et al. (2004). "Architecture-level Modifiability Analysis (ALMA)". In: *J. Syst. Softw.* 69.1-2, pp. 129–147. ISSN: 0164-1212. DOI: 10.1016/S0164-1212(03)00080-3. URL: [http://dx.doi.org/10.1016/S0164-1212\(03\)00080-3](http://dx.doi.org/10.1016/S0164-1212(03)00080-3).
- Bock, Alexander (2015). "Beyond Narrow Decision Models: Toward Integrative Models of Organizational Decision Processes". In: *Proceedings of the 17th IEEE Conference on Business Informatics (CBI 2015)*. IEEE Computer Society, pp. 181–190. DOI: doi:10.1109/CBI.2015.34. URL: <http://ieeexplore.ieee.org/document/7264731>.
- Bock, Alexander and Ulrich Frank (2016). "Multi-perspective Enterprise Modeling—Conceptual Foundation and Implementation with ADOxx". In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*. Ed. by Dimitris Karagiannis, C. Heinrich Mayr, and John Mylopoulos. Cham: Springer International Publishing, pp. 241–267. ISBN: 978-3-319-39417-6. DOI: 10.1007/978-3-319-39417-6_11. URL: http://dx.doi.org/10.1007/978-3-319-39417-6_11.
- Bock, Alexander, Monika Kaczmarek, et al. (2014). "A Comparative Analysis of Selected Enterprise Modelling Approaches". In: *Proc. of PoEM 2014*. Ed. by Ulrich Frank and et al. Vol. 197. LNBIP. Berlin: Springer, pp. 148–163. URL: http://dx.doi.org/10.1007/978-3-662-45501-2_11.
- Bucher, Tobias et al. (2006). "Analysis and Application Scenarios of Enterprise Architecture: An Exploratory Study". In: *2006 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*, pp. 28–28. DOI: 10.1109/EDOCW.2006.22.

- Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong (2020). *Mathematics for Machine Learning*. The Cambridge University Press.
- Duncan, Nancy Bogucki (1995). "Capturing Flexibility of Information Technology Infrastructure: A Study of Resource Characteristics and Their Measure". In: *Journal of Management Information Systems* 12.2, pp. 37–57. DOI: 10.1080/07421222.1995.11518080. eprint: <https://doi.org/10.1080/07421222.1995.11518080>. URL: <https://doi.org/10.1080/07421222.1995.11518080>.
- Ernst, Dominik, David Bermbach, and Stefan Tai (2016). "Understanding the Container Ecosystem: A Taxonomy of Building Blocks for Container Lifecycle and Cluster Management". In: *Proceedings of the 2nd International Workshop on Container Technologies and Container Clouds (WoC 2016)*. IEEE.
- Feiler, Peter H, David P Gluch, and John J Hudak (2006). *The architecture analysis & design language (AADL): An introduction*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.
- Feltus, Christophe, Michael Petit, and Eric Dubois (2009). "Strengthening Employee's Responsibility to Enhance Governance of IT: COBIT RACI Chart Case Study". In: New York, NY, USA: Association for Computing Machinery, pp. 23–32. ISBN: 9781605587875.
- Fill, Hans-Georg and Dimitris Karagiannis (2013). "On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform". In: *EMISA* 8.1, pp. 4–25.
- Frank, Ulrich (2010). *Outline of a Method for Designing Domain-Specific Modelling Languages*. ICB Research Report 42. Essen: University of Duisburg-Essen.
- Frank, Ulrich (2011). *The MEMO Meta modeling Language (MML) and Language Architecture. 2nd Edition*. ICB-Research Report 43. Essen: University of Duisburg-Essen.
- Frank, Ulrich (2013). "Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines". In: *Domain Engineering*. Ed. by Iris Reinhartz-Berger et al. Berlin: Springer, pp. 133–157.
- Frank, Ulrich (2014a). "Multilevel Modeling". In: *Business and Information Systems Engineering* 6.6, pp. 319–337. ISSN: 1867-0202. DOI: 10.1007/s12599-014-0350-4. URL: <http://dx.doi.org/10.1007/s12599-014-0350-4>.
- Frank, Ulrich (2014b). "Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges". In: *SoSyM* 13.3, pp. 941–962.

Bibliography

- Frank, Ulrich (2016). "Designing Models and Systems to Support IT Management: A Case for Multilevel Modeling". In: *MULTI@MoDELS*. Ed. by Colin Atkinson, Georg Grossmann, and Tony Clark. ceur-ws.org, pp. 3–24.
- Frank, Ulrich et al. (2009). "ITML: A Domain-Specific Modeling Language for Supporting Business Driven IT Management". In: *In Proc. of the 9th OOPSLA workshop on domain-specific modeling (DSM, 2009)*.
- Goldstein, Anat and Ulrich Frank (2016). "Components of a multi-perspective modeling method for designing and managing IT security systems". In: *ISeB 14.1*, pp. 101–140. ISSN: 1617-9854. DOI: 10.1007/s10257-015-0276-5. URL: <http://dx.doi.org/10.1007/s10257-015-0276-5>.
- Hanschke, Inge (2010). *Strategic IT Management. A Toolkit for Enterprise Architecture Management*. Berlin: Springer. ISBN: 3-540-29169-5.
- Heise, David (2013). *Unternehmensmodell-basiertes IT-Kostenmanagement als Bestandteil eines integrativen IT-Controllings*. Berlin: Logos.
- ISO (2011). *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Geneva: ISO.
- Johansson, Erik and Pontus Johnson (2005). "Assessment of enterprise information security – an architecture theory diagram definition". In: *Proc. of Conference on Systems Engineering Research*, pp. 136–146.
- Kaczmarek, Monika and Sybren de Kinderen (2016). "A Conceptualization of IT Platform for the Needs of Enterprise IT Landscape Modeling". In: *18th IEEE Conference on Business Informatics, CBI 2016, 29th August - 1st September 2016, Paris, France, Volume 1 - Conference Papers*. Ed. by Elena Kornysheva et al. IEEE, pp. 74–83. DOI: 10.1109/CBI.2016.17. URL: <https://doi.org/10.1109/CBI.2016.17>.
- Kaczmarek-Heß, Monika and Sybren de Kinderen (2017). "A Multilevel Model of IT Platforms for the Needs of Enterprise IT Landscape Analyses". In: *Bus. Inf. Syst. Eng.* 59.5, pp. 315–329. DOI: 10.1007/s12599-017-0482-4. URL: <https://doi.org/10.1007/s12599-017-0482-4>.
- Kattenstroth, Heiko, Ulrich Frank, and David Heise (2013). "Towards a Modelling Method in Support of Evaluating Information Systems Integration". In: *Enterprise Modelling and Information Systems Architectures (EMISA 2013)*. Ed. by Reinhard Jung and Manfred Reichert. Bonn: Gesellschaft für Informatik e.V., pp. 85–99.

- Kinderen, Sybren de and Monika Kaczmarek-Heß (2018). "Enterprise Modeling in Support of SOA Migration Analysis". In: *EMISA* 13.1, pp. 1–36.
- Kinderen, Sybren de and Monika Kaczmarek-Heß (2021). "Making a Case for Multi-level Reference Modeling - A Comparison of Conventional and Multi-level Language Architectures for Reference Modeling Challenges". In: *Wirtschaftsinformatik 2021 Proceedings*. aisnet.
- Laan, Sjaak (2017). *IT Infrastructure Architecture – Infrastructure Building Blocks and Concepts*. Lulu Press Inc.
- Lankhorst, Marc (2013). *Enterprise Architecture at Work: Modelling, Communication and Analysis*. 3rd ed. The Enterprise Engineering Series. Heidelberg: Springer.
- Lara, Juan De, Esther Guerra, and Jesús Sánchez Cuadrado (2014). "When and How to Use Multilevel Modelling". In: *ACM Trans. Softw. Eng. Methodol.* 24.2, 12:1–12:46. ISSN: 1049-331X. DOI: 10.1145/2685615. URL: <http://doi.acm.org/10.1145/2685615>.
- Luftman, Jerry and C. Bullen (2004). *Managing the Information Technology Resource: Leadership in the Information Age*. Pearson Education. ISBN: 9780130351265.
- Luftman, Jerry, Hossein S Zadeh, et al. (2012). "Key Information Technology and Management Issues 2011-2012: An International Study". In: *Journal of Information Technology* 27.3, pp. 198–212. DOI: 10.1057/jit.2012.14.
- Mangiapane, Markus and Roman P. Buechler (2015). *Modernes IT-Management*. Springer. ISBN: ISBN 9783658034924.
- Moody, Daniel L. (2009). "The Physics of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering". In: *IEEE Trans. Software Eng.* 35.6, pp. 756–779.
- Newman, Sam (2015). *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc.
- Niemann, Klaus D. (2005). *Von der Unternehmensarchitektur zur IT-Governance: Bausteine für ein wirksames IT-Management*. Wiesbaden: Vieweg+Teubner Verlag.
- Nyrhinen, Mari (2006). *IT infrastructure: structure, properties and processes*. Working Papers. Helsinki School of Economics.
- Overbeek, Sietse, Ulrich Frank, and Christian Köhling (2015). "A language for multi-perspective goal modelling: Challenges, requirements and solutions". In: *CSI* 38, pp. 1–16. DOI: 10.1016/j.csi.2014.08.001.
- Papazoglou, Mike P and Willem-Jan Van Den Heuvel (2007). "Service oriented architectures: approaches, technologies and research issues". In: *The VLDB journal* 16.3, pp. 389–415.

Bibliography

- Parviainen, Paivi et al. (2017). "Tackling the digitalization challenge: How to benefit from digitalization in practice". In: *International Journal of Information Systems and Project Management* 5, pp. 63–77. DOI: 10.12821/ijispm050104.
- Scheer, August-Wilhelm (2001). *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. 4th ed. Heidelberg: Springer. ISBN: 3-540-41601-3.
- Sultan, Sari, Imtiaz Ahmad, and Tassos Dimitriou (2019). "Container Security: Issues, Challenges, and the Road Ahead". In: *IEEE Access* 7, pp. 52976–52996. DOI: 10.1109/ACCESS.2019.2911732.
- Syed, Madiha H. and Eduardo B. Fernandez (2018). "A Reference Architecture for the Container Ecosystem". In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ARES 2018. Hamburg, Germany: Association for Computing Machinery. ISBN: 9781450364485. DOI: 10.1145/3230833.3232854. URL: <https://doi.org/10.1145/3230833.3232854>.
- Tanenbaum, Andrew S. (2006). *Structured Computer Organization*. 5th. USA: Prentice Hall PTR.
- Tanenbaum, Andrew S. and Herbert Bos (2014). *Modern Operating Systems*. 4th ed. Boston, MA: Pearson. ISBN: 978-0-13-359162-0.
- The Open Group (2012). *ArchiMate 2.0 specification: Open Group Standard*. Zaltbommel: Van Haren. ISBN: 978-90-8753-692-3.

Previously Published ICB Research Reports

2021

No 71 (October 2021)

Schauer, Carola; Schwarz, Tobias: "Wie sollte man Studieninteressierte über Wirtschaftsinformatik informieren?: Ergebnisse einer Befragung von Oberstufenschülerinnen und -schülern in NRW zu präferierten Informationswegen und Vorstellungen über das Studieneinfach."

No 70 (December 2021)

Frank, Ulrich; Maier, Pierre; Bock, Alexander: "Low Code Platforms: Promises, Concepts and Prospects – A Comparative Study of Ten Systems."

No 69 (June 2021)

Schauer, Carola: "WirtschaftsinformatikStudiengänge an Universitäten in Deutschland – Analyse der Studienanfängerzahlen und Frauenanteile im Vergleich zur Informatik und zu Fachhochschulen."

2020

No 68 (December 2020)

Schauer, Carola: "Warum entscheiden sich Studienanfänger für Wirtschaftsinformatik? – Ergebnisse einer Umfrage unter Bachelorstudierenden im ersten Fachsemester Wirtschaftsinformatik an der UDE (Nov. 2018 und Nov. 2019)."

No 67 (November 2020)

Frank, Ulrich; Bock, Alexander: "Organisationsforschung und Wirtschaftsinformatik: Zeit für eine Annäherung?"

2018

No 66 (December 2019)

Frank, Ulrich: "The Flexible Multi-Level Modelling and Execution Language (FMML)^x. Version 2.0: Analysis of Requirements and Technical Terminology."

2015

No 65 (August 2015)

Schauer, Carola; Schauer, Hanno: "IT- und Medienbildung an Schulen. Ergebnisse einer empirischen Studie an einem rheinland-pfälzischen Gymnasium."

No 64 (January 2015)

Föcker, Felix; Houdek, Frank; Daun, Marian; Weyer, Thorsten: "Model-Based Engineering of an Automotive Adaptive Exterior Lighting System – Realistic Example Specifications of Behavioral Requirements and Functional Design."

No 63 (January 2015)

Schauer, Carola; Schauer, Hanno: "IT an allgemeinbildenden Schulen: Bildungsgegenstand und -infrastruktur – Auswertung internationaler empirischer Studien und Literaturanalyse."

2014

No 62 (October 2014)

Königer, Stephan; Heß, Michael: "Ein Software-Werkzeug zur multiperspektivischen Bewertung innovativer Produkte, Projekte und Dienstleistungen: Realisierung im Projekt Hospital Engineering."

No 61 (August 2014)

Schauer, Carola; Frank, Ulrich: "Wirtschaftsinformatik an Schulen – Status und Desiderata mit Fokus auf Nordrhein-Westfalen."

No 60 (May 2014)

Heß, Michael: "Multiperspektivische Dokumentation und Informationsbedarfsanalyse kardiologischer Prozesse sowie Konzeptualisierung ausgewählter medizinischer Ressourcentypen im Projekt Hospital Engineering"

No 59 (May 2014)

Goedicke, Michael; Kurt-Karaoglu, Filiz; Schwinning, Nils; Schypula, Melanie; Striewe, Michael: "Zweiter Jahresbericht zum Projekt 'Bildungsgerechtigkeit im Fokus' (Teilprojekt 1.2 – 'Blended Learning') an der Fakultät für Wirtschaftswissenschaften"

No 58 (March 2014)

Breitschwerdt, Rüdiger; Heß, Michael: "Konzeption eines Bezugsrahmens zur Analyse und Entwicklung von Geschäftsmodellen mobiler Gesundheitsdienstleistungen – Langfassung"

No 57 (March 2014)

Heß, Michael; Schlieter, Hannes (Hrsg.): "Modellierung im Gesundheitswesen – Tagungsband des Workshops im Rahmen der "Modellierung 2014""

2013

No 56 (July 2013)

Svensson, Richard; Berntsson; Berry, Daniel M.; Daneva, Maya; Doerr, Joerg; Espana, Sergio; Herrmann, Andrea; Herzwurm, Georg; Hoffmann, Anne; Pena, Raul Mazo; Opdahl, Andreas L.; Pastor, Oscar; Pietsch, Wolfram; Salinesi, Camille; Schneider, Kurt; Seyff, Norbert; van de Weerd, Inge; Wieringa, Roel; Wnuk, Krzysztof (Eds.): "19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2013). Proceedings of the REFSQ 2013 Workshops CreaRE, IWSPM, and RePriCo, the REFSQ 2013 Empirical Track (Empirical Live Experiment and Empirical Research Fair), the REFSQ 2013 Doctoral Symposium, and the REFSQ 2013 Poster Session""

No 55 (May 2013)

Daun, Marian; Focke, Markus; Holtmann, Jörg; Tenbergen, Bastian "Goal-Scenario-Oriented Requirements Engineering for Functional Decomposition with Bidirectional Transformation to Controlled Natural Language. Case Study "Body Control Module""

No 54 (March 2013)

Fischotter, Melanie; Goedicke, Michael; Kurt-Karaoglu, Filiz; Schwinning, Nils; Striewe, Michael "Erster Jahresbericht zum Projekt "Bildungsgerechtigkeit im Fokus" (Teilprojekt 1.2 – "Blended Learning") an der Fakultät für Wirtschaftswissenschaften"

2012

No 53 (December 2012)

Frank, Ulrich: "Thoughts on Classification / Instantiation and Generalisation / Specialisation"

No 52 (July 2012)

Berntsson-Svensson, Richard; Berry, Daniel; Daneva, Maya; Dörr, Jörg; Fricker, Samuel A; Herrmann, Andrea; Herzwurm, Georg; Kauppinen, Marjo; Madhavji, Nazim H; Mahaux, Martin; Paech, Barbara; Penzenstadler, Birgit; Pietsch, Wolfram; Salinesi, Camille; Schneider, Kurt; Seyff, Norbert; van de Weerd, Inge (Eds.): "18th International Working Conference on Requirements Engineering – Foundation for Software Quality. Proceedings of the Workshops RE4SuSy, REEW, CreaRE, RePriCo, IWSPM and the Conference Related Empirical Study, Empirical Fair and Doctoral Symposium"

No 51 (May 2012)

Frank, Ulrich: "Specialisation in Business Process Modelling – Motivation, Approaches and Limitations"

No 50 (March 2012)

Adelsberger, Heimo; Drechsler, Andreas; Herzig, Eric; Michaelis, Alexander; Schulz, Philipp ; Schütz, Stefan; Ulrich, Udo: "Qualitative und quantitative Analyse von SOA-Studien – Eine Metastudie zu serviceorientierten Architekturen"

2011

No 49 (December 2011)

Frank, Ulrich: "MEMO Organisation Modelling Language (2) – Focus on Business Processes"

No 48 (December 2011)

Frank, Ulrich: "MEMO Organisation Modelling Language (1) – Focus on Organisational Structure"

No 47 (December 2011)

Frank, Ulrich: "Multiperspective Enterprise Modelling – Requirements and Core Diagram Typs"

No 46 (December 2011)

Frank, Ulrich: "Multiperspective Enterprise Modelling – Background and Terminological Foundation"

No 45 (November 2011)

Frank, Ulrich; Strecker, Stefan; Heise, David; Kattenstroth, Heiko; Schauer, Carola: "Leitfaden zur Erstellung wissenschaftlicher Arbeiten in der Wirtschaftsinformatik"

No 44 (September 2011)

Berenbach, Brian; Daneva, Maya; Dörr, Jörg; Fricker, Samuel; Gervasi, Vincenzo; Glinz, Martin; Herrmann, Andrea; Krams, Benedikt; Madhavji, Nazim H; Paech, Barbara; Schockert, Sixten; Seyff, Norbert (Eds.): "17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2011) – Proceedings of the REFSQ 2011 Workshops REEW, EPICAL and RePriCo, the REFSQ 2011 Empirical Track (Empirical Live Experiment and Empirical Research Fair), and the REFSQ 2011 Doctoral Symposium"

No 43 (February 2011)

Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Language Architecture. 2nd Edition"

2010

No 42 (December 2010)

Frank, Ulrich: "Outline of a Method for Designing Domain-Specific Modelling Languages"

No 41 (December 2010)

Adelsberger, Heimo; Drechsler, Andreas (Hrsg.): "Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive – Cloud Governance, Cloud Security und Einsatz von Cloud Computing in jungen Unternehmen"

No 40 (October 2010)

Bürsner, Simone; Dörr, Jörg; Gehlert, Andreas; Herrmann, Andrea; Herzwurm, Georg; Janzen, Dirk; Merten, Thorsten; Pietschm, Wolfram; Schmid, Klaus; Schneider, Kurt; Thurimella, Anil Kumar: "16th International Working Conference on Requirements Engineering: Foundation for Software Quality – Proceedings of the Workshops CreaRE, PLREQ, RePriCo and RESC"

No 39 (May 2010)

Strecker, Stefan; Heise, David; Frank, Ulrich: "Entwurf einer Mentoring-Konzeption für den Studiengang M.Sc. Wirtschaftsinformatik an der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen"

No 38 (February 2010)

Schauer, Carola : "Wie praxisorientiert ist die Wirtschaftsinformatik? Einschätzungen von CIOs und WI-Professoren"

No 37 (January 2010)

Benavides, David; Batory, Don; Grunbacher, Paul (Eds.): "Fourth International Workshop on Variability Modelling of Software-intensive Systems"

2009

No 36 (December 2009)

Strecker, Stefan: "Ein Kommentar zur Diskussion um Begriff und Verständnis der IT-Governance – Anregungen zu einer kritischen Reflexion"

No 35 (August 2009)

Rüngeler, Irene; Tüxen, Michael; Rathgeb, Erwin P.: "Considerations on Handling Link Errors in SCTP"

No 34 (June 2009)

Karastoyanova, Dimka; Kazhamiakan, Raman; Metzger, Andreas; Pistore, Marco (Eds.): "Workshop on Service Monitoring, Adaptation and Beyond"

No 33 (May 2009)

Adelsberger, Heimo; Drechsler, Andreas; Bruckmann, Tobias; Kalvelage, Peter; Kinne, Sophia; Pellinger, Jan; Rosenberger, Marcel; Trepper, Tobias: "Einsatz von Social Software in Unternehmen - Studie über Umfang und Zweck der Nutzung"

No 32 (April 2009)

Barth, Manfred; Gadatsch, Andreas; Kutz, Martin; Ruding, Otto; Schauer, Hanno; Strecker, Stefan: "Leitbild IT-Controller/-in . Beitrag der Fachgruppe IT-Controlling der Gesellschaft für Informatik e. V."

No 31 (April 2009)

Frank, Ulrich; Strecker, Stefan: "Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems – Requirements, Conceptual Foundation and Design Options"

No 30 (February 2009)

Schauer, Hanno; Wolff, Frank: "Kriterien guter Wissensarbeit - Ein Vorschlag aus dem Blickwinkel der Wissenschaftstheorie (Langfassung)"

No 29 (January 2009)

Benavides, David; Metzger, Andreas; Eisenecker, Ulrich (Eds.): "Third International Workshop on Variability Modelling of Software-intensive Systems"

2008

No 28 (December 2008)

Goedicke, Michael; Striewe, Michael; Balz, Moritz: "Computer Aided Assessments and Programming Exercises with JACK"

No 27 (December 2008)

Schauer, Carola: "Größe und Ausrichtung der Disziplin Wirtschaftsinformatik an Universitäten im deutschsprachigen Raum – Aktueller Status und Entwicklung seit 1992"

No 26 (September 2008)

Milen, Tilev; Bruno Muller–Clostermann: "CapSys: A Tool for Macroscopic Capacity Planning"

No 25 (August 2008)

Eicker, Stefan; Spies, Thorsten; Tschersich, Markus: "Einsatz von Multi–Touch beim Softwaredesign am Beispiel der CRC Card–Methode"

No 24 (August 2008)

Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Language Architecture - Revised Version"

No 23 (January 2008)

Sprenger, Jonas; Jung, Jürgen: "Enterprise Modelling in the Context of Manufacturing - Outline of an Approach Supporting Production Planning"

No 22 (January 2008)

Heymans, Patrick; Kang, Kyo–Chul; Metzger, Andreas, Pohl, Klaus (Eds.): "Second International Workshop on Variability Modelling of Software–intensive Systems."

2007

No 21 (September 2007)

Eicker, Stefan; Annett Nagel; Peter M. Schuler: "Flexibilität im Geschäftsprozessmanagement-Kreislauf"

No 20 (August 2007)

Blau, Holger; Eicker, Stefan; Spies, Thorsten: "Reifegradüberwachung von Software"

No 19 (June 2007)

Schauer, Carola: "Relevance and Success of IS Teaching and Research: An Analysis of the Relevance Debate"

No 18 (May 2007)

Schauer, Carola: "Rekonstruktion der historischen Entwicklung der Wirtschaftsinformatik: Schritte der Institutionalisierung, Diskussion zum Status, Rahmenempfehlungen für die Lehre"

No 17 (May 2007)

Schauer, Carola; Schmeing, Tobias: "Development of IS Teaching in North-America: An Analysis of Model Curricula"

No 16 (May 2007)

Müller-Clostermann, Bruno; Tilev, Milen: "Using G/G/m-Models for Multi-Server and Mainframe Capacity Planning"

No 15 (April 2007)

Heise, David; Schauer, Carola; Strecker, Stefan: "Informationsquellen für IT-Professionals - Analyse und Bewertung der Fachpresse aus Sicht der Wirtschaftsinformatik"

No 14 (March 2007)

Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: "Auswahl von Bewertungsmethoden für Softwarearchitekturen"

No 13 (February 2007)

Eicker, Stefan; Spies, Thorsten; Kahl, Christian: "Softwarevisualisierung im Kontext serviceorientierter Architekturen"

No 12 (February 2007)

Brenner, Freimut: "Cumulative Measures of Absorbing Joint Markov Chains and an Application to Markovian Process Algebras"

No 11 (February 2007)

Kirchner, Lutz: "Entwurf einer Modellierungssprache zur Unterstützung der Aufgaben des IT Managements - Grundlagen, Anforderungen und Metamodell"

No 10 (February 2007)

Schauer, Carola; Strecker, Stefan: "Vergleichende Literaturstudie aktueller einführender Lehrbücher der Wirtschaftsinformatik: Bezugsrahmen und Auswertung"

No 9 (February 2007)

Strecker, Stefan; Kuckertz, Andreas; Pawlowski, Jan M.: "Überlegungen zur Qualifizierung des wissenschaftlichen Nachwuchses: Ein Diskussionsbeitrag zur (kumulativen) Habilitation"

No 8 (February 2007)

Frank, Ulrich; Strecker, Stefan; Koch, Stefan: "Open Model - Ein Vorschlag für ein Forschungsprogramm der Wirtschaftsinformatik (Langfassung)"

2006

No 7 (December 2006)

Frank, Ulrich: "Towards a Pluralistic Conception of Research Methods in Information Systems Research"

No 6 (April 2006)

Frank, Ulrich: "Evaluation von Forschung und Lehre an Universitäten - Ein Diskussionsbeitrag"

No 5 (April 2006)

Jung, Jürgen: "Supply Chains in the Context of Resource Modelling"

No 4 (February 2006)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part III - Results Wirtschaftsinformatik Discipline"

2005

No 3 (December 2005)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part II - Results Information Systems Discipline"

No 2 (December 2005)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part I - Research Objectives and Method"

No 1 (August 2005)

Lange, Carola: "Ein Bezugsrahmen zur Beschreibung von Forschungsgegenständen und -methoden in Wirtschaftsinformatik und Information Systems"

Research Group	Core Research Topics
Prof. Dr. F. Ahlemann Information Systems and Strategic Management	Strategic planning of IS, Enterprise Architecture Management, IT Vendor Management, Project Portfolio Management, IT Governance, Strategic IT Benchmarking
Prof. Dr. F. Beck Visualization Research Group	Information visualization, software visualization, visual analytics
Prof. Dr. T. Brinda Didactics of Informatics	Competence modelling and educational standards in Informatics, Students' conceptions in Informatics, Education in the digital world, Vocational education in Informatics
Prof. Dr. P. Chamoni MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
Prof. Dr.-Ing. L. Davi Research in Secure Software Systems	Software Security, Security of Smart Contracts, Trusted Computing, Hardware-assisted Security
Prof. Dr. K. Echte Dependability of Computing Systems	Dependability of Computing Systems
Prof. Dr. S. Eicker Information Systems and Software Engineering	Process Models, Software-Architectures
Prof. Dr. U. Frank Information Systems and Enterprise Modelling	Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management
Prof. Dr. M. Goedicke Specification of Software Systems	Distributed Systems, Software Components, CSCW
Prof. Dr. V. Gruhn Software Engineering	Design of Software Processes, Software Architecture, Usability, Mobile Applications, Component-based and Generative Software Development
Prof. Dr. T. Kollmann E-Business and E-Entrepreneurship	E-Business and Information Management, E-Entrepreneurship/E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing
Prof. Dr. J. Marrón Networked Embedded Systems	Sensor Networks, Adaptive Systems, System Software for embedded systems, Data Management in mobile environments, Hoarding / Caching, Ubiquitous/Pervasive Computing, Semi-structured databases
Prof. Dr. K. Pohl Software Systems Engineering	Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components
Prof. Dr. Ing. E. Rathgeb Computer Network Technology	Computer Network Technology
Prof. Dr. S. Schneegaß Human Computer Interaction	Mobile, wearable, and ubiquitous computing systems, Implicit Feedback, Usable Security, Smart Clothing, Interaction in Virtual and Augmented Worlds, Ubiquitous Interaction
Prof. Dr. R. Schütte Business Informatics and Integrated Information Systems	Enterprise Systems, IS-Architectures, Digitalization of organisations, Information modelling, Scientific theory problems of the Business Informatics field
Prof. Dr. S. Stieglitz Professional Communication in Electronic Media / Social Media	Digital Enterprise / Digital Innovation, Digital Society

DuEPublico

Duisburg-Essen Publications online

UNIVERSITÄT
DUISBURG
ESSEN

Offen im Denken

ub

universitäts
bibliothek

Dieser Text wird via DuEPublico, dem Dokumenten- und Publikationsserver der Universität Duisburg-Essen, zur Verfügung gestellt. Die hier veröffentlichte Version der E-Publikation kann von einer eventuell ebenfalls veröffentlichten Verlagsversion abweichen.

DOI: 10.17185/duepublico/75252

URN: urn:nbn:de:hbz:464-20220110-111514-6

Alle Rechte vorbehalten.