The MEMO Meta Modelling Language (MML) and Language Architecture

Frank, Ulrich

In: ICB Research Reports - Forschungsberichte des ICB / 2011

DOI: https://doi.org/10.17185/duepublico/47072

URN: urn:nbn:de:hbz:464-20180918-073405-8

Link: https://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet?id=47072

Source: ICB-Research Report No. 43, February 2011

ICB
Institut für Informatik und Wirtschaftsinformatik

Ulrich Frank

The MEMO Meta Modelling Language (MML) and Language Architecture

43

ICB-RESEARCH REPORT

2nd Edition

UNIVERSITÄT DUISBURG ESSEN

ICB-Research Report No. 43
February 2011

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb 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.

Authors' Address:

Ulrich Frank

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
D-45141 Essen

ulrich.frank@uni-due.de

Contact:

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen

Tel.: 0201-183-4041
Fax: 0201-183-4011
Email: icb@uni-duisburg-essen.de

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)

Preface to Revised Edition

A substantial amount of research and critical evaluation had gone into the MEMO meta modelling language when it was first published in 2007. This was for a good reason: A meta modelling language should be invariant for a longer time, since it is the foundation of an entire family of languages. Therefore, changing the meta modelling language jeopardizes the integrity of the respective modelling languages and the corresponding model editors. Nevertheless, our work on specifying modelling languages produced some additional requirements we were not aware of at first. It also revealed a few misconceptions. These discoveries led to a minor revision of the meta modelling language that was published in 2010. During the last months one further requirement evolved that is related to the very conception of modelling language. It also turned out that one concept in the meta meta model was specified on an inappropriate level of abstraction. Fortunately, this misconception did not result in erroneous implementations of corresponding tools because it was filtered out by an adequate interpretation of those who developed the software. The resulting revision of the meta modelling language may seem minor, because it affects mainly concepts in the meta meta model only. Nevertheless, it represents a major change, because it reflects an extended conception of modelling language. In addition to that, the meta modelling language was supplemented by an extensible set of "auxiliary" types. While these do not affect the semantics of the meta meta model, they promote the productivity of developing modelling languages and contribute to a higher level of consistency and coherence within a family of modelling languages. I also used the opportunity of a new edition for applying a few marginal changes. Among other things, they comprise the renaming of "deferredExternal" attributes to "obtainable".

The changes[1] are mainly motivated by the demand to provide an effective support for specifying domain-specific modelling languages. Unfortunately, they are accompanied by the unpleasant side-effect that the complexity of the meta meta model was increased – which compromises the reasonable requirement to keep a meta language simple. Since the research on modelling languages and their specification is far from having reached a mature state, it would be presumptuous to assume that the meta meta model will not require further modifications. I hope, however, that it serves us as a suitable foundation for developing languages and tools for the next years.

Essen, February 2011

Ulrich Frank

[1] Additional minor corrections in figures 6, 7, 9, and 18 have been made in November 2013.

Abstract

The family of languages that builds the foundation of the MEMO method is intended to feature a high degree of inter-language integration. For this purpose, the languages need to share common concepts. In order to define concepts that are semantically equivalent, it is recommendable to use the same meta modelling language for specifying the MEMO modelling languages. The previous version of the meta modelling language used for this purpose needed a revision. At the same time, there was need to account for alternative approaches to specifying modelling languages, especially those offered by the OMG or the Eclipse foundation. This report starts with an analysis of requirements that should be accounted for by a meta modelling language. Subsequently, the UML infrastructure library and meta object facility (MOF) are evaluated against these requirements. In addition to that, the report presents an evaluation of the Ecore model, which serves to represent meta models within the Eclipse Graphical Modeling Framework (GMF). The evaluation of both approaches shows that none of them is satisfactory as a meta modelling language for enterprise modelling. Then, the new version of the MEMO meta modelling language (MML) is presented. The language specification consists of a meta meta model that specifies that semantics and abstract syntax and a corresponding graphical notation (concrete syntax). The new version features a concept called *intrinsic features* that allows for differentiating between features that apply to types and those that apply to instances. It also includes a modified graphical notation that supports a clear distinction of meta models from models on other levels of abstraction. Finally, the report presents the outline of a tool that supports the creation and editing of MEMO meta models as well as their transformation into representations which can be used in the Eclipse modelling framework.

Table of Contents

PREFACE TO REVISED EDITION ..... **I**

1 INTRODUCTION **1**

2 META MODELLING LANGUAGES: REQUIREMENTS **2**

2.1 GENERAL REQUIREMENTS FOR META MODELLING LANGUAGES 4

2.1.1 Formal Requirements *4*

2.1.2 User-Oriented Requirements *5*

2.1.3 Application-Oriented Requirements *5*

3 META META MODELS: PREVALENT APPROACHES **8**

3.1 UML: INFRASTRUCTURE LIBRARY AND THE META OBJECT FACILITY 8

3.2 ECLIPSE FOUNDATION: ECORE 14

4 LANGUAGE SPECIFICATION **18**

4.1 BASIC DATA TYPES OR DOMAINS 18

4.2 INTRINSIC FEATURES 19

4.3 "LANGUAGE-LEVEL TYPES": CONCEPTS TO MODEL INSTANCES 23

4.4 THE META META MODEL 24

4.5 REFERENCE INSTANTIATIONS: AUXILIARY TYPES 28

4.6 THE GRAPHICAL NOTATION 31

4.7 EXAMPLES 34

4.8 PRELIMINARY EVALUATION 37

5 THE MEMO LANGUAGE ARCHITECTURE **39**

6 OUTLINE OF A MODELLING TOOL **41**

7 FUTURE RESEARCH **43**

8 REFERENCES **44**

Figures

FIGURE 1: SEMANTIC NET OF KEY TERMS AND CORRESPONDING LEVELS OF ABSTRACTION 3

FIGURE 2: EMOF OR PART OF THE INFRASTRUCTURE LIBRARY RESPECTIVELY ([OMG06A], P. 33; [OMG06B], P. 93) 10

FIGURE 3: REVISED VERSION OF EMOF 10

FIGURE 4: CMOF: "KEY CONCRETE CLASSES" ([OMG06A], P. 47) 12

FIGURE 5: ECORE 16

FIGURE 6: BASIC DATA TYPES USED WITHIN THE META META MODEL 19

FIGURE 7: EXEMPLARY USE OF A POWER TYPE – ADAPTED FROM [ODEL98] 20

FIGURE 8: EXAMPLE MODELLED WITH CLABJECTS – ACCORDING TO [ATKÜ07] 21

FIGURE 9: EXAMPLE MODELLED WITH INTRINSIC ATTRIBUTES, ASSOCIATIONS AND TYPES 23

FIGURE 10: COMPARISON OF INTRINSIC FEATURES AND MODELLING OF INSTANCES 24

FIGURE 11: THE MEMO META META MODEL 27

FIGURE 12: PLACEMENT OF AUXILIARY TYPES 29

FIGURE 13: ELEMENTS OF THE GRAPHICAL NOTATION 33

FIGURE 14: OPTIONS TO MARK THE ELEMENTS OF A META MODEL AS BELONGING TO A PARTICULAR LANGUAGE 34

FIGURE 15: A META MODEL OF THE ERM 34

FIGURE 16: DIFFERENTIATING TWO META MODELS THROUGH SPECIFIC SYMBOLS 35

FIGURE 17: THE USE OF INTRINSIC FEATURES 36

FIGURE 18: THE USE OF LANGUAGE-LEVEL TYPES 37

FIGURE 19: THE MEMO LANGUAGE LAYERS 39

FIGURE 20: THE MEMO LANGUAGE ARCHITECTURE AND CORRESPONDING CONCEPTUAL FOUNDATION FOR MODELLING TOOLS 40

FIGURE 21: THE MEMO META META MODEL AS AN ECORE INSTANCE 42

FIGURE 22: SIMPLIFIED WORKFLOW FOR DEVELOPING ADDITIONAL MODEL EDITORS WITHIN MEMO CENTER 42

Tables

TABLE 1: EVALUATION OF MOF AND THE UML INFRASTRUCTURE LIBRARY RESPECTIVELY 14

TABLE 2: EVALUATION VON ECORE 17

TABLE 3: PRELIMINARY SET OF GENERIC REFERENCE TYPES 29

TABLE 4: PRELIMINARY SET OF DOMAIN-SPECIFIC TYPES 30

TABLE 5: REPRESENTATION OF TEXTUAL ELEMENTS 31

TABLE 6: EVALUATION OF THE MEMO META MODELLING LANGUAGE 38

Typographical Conventions

If textual elements of meta (meta) models are referred to in the standard body text, they are printed in Courier italic, e.g. *`MetaEntity`*.

1 Introduction

Multi-Perspective Enterprise Modelling (MEMO), a method to guide the design and analysis of enterprise models, is based on a set of modelling languages that allow for creating conceptual models that represent various perspectives on an enterprise. These languages are specified through meta models. In order to foster the integration of these languages and – as a consequence – of the corresponding models, it is required that the language specifications, i.e. the meta models, make use of common concepts. This in turn recommends using common concepts for specifying the meta models. In other words: The MEMO languages should be specified using the concepts of a common meta meta model. Such a model was defined some time ago [Fran98a]. It has been successfully used for the specification of MEMO modelling languages. However, various developments of the previous years recommend rethinking the design of the meta meta model. The experiences we gathered with designing meta models resulted in additional requirements. Also, we were not satisfied any more with some decisions the first version of the meta meta model is based on. Furthermore, the remarkable relevance the UML has gained recommends taking into account its language architecture. Last but not least, it is useful to account for the development of modelling tools: Exploiting the potential of a modelling language will often recommend using a corresponding modelling tool. Since the implementation of a modelling tool implies a major investment, it will often be no option to develop a tool from scratch. A number of tools, especially so called meta modelling tools, promise to increase the productivity of developing modelling tools tremendously. Among these development environments, one has gained special relevance. The Eclipse Modeling Framework (EMF) as well as the Eclipse Graphical Modeling Framework (GMF) are subject of an open source project. They are supported by a large community of developers and users. The GMF targets the development of graphical modelling tools. To develop a specific modelling tool, the corresponding language specification has to be reconstructed using the meta model provided with the framework. This meta model, called Ecore, serves to generate Java classes which in turn represent language concepts. Hence, using GMF recommends analysing how the concepts of the intended meta meta model can be transformed to Ecore concepts. As an alternative, Ecore could be used directly as the meta meta model for specifying the MEMO languages. This requires evaluating whether Ecore could satisfy this purpose.

Against this background, we will first look at requirements a meta meta model for specifying modelling languages should satisfy. MOF and Ecore are then evaluated against these requirements – to come to the conclusions that none of them is a satisfactory candidate for serving as the MEMO meta meta model. Subsequently, the revised version of the meta meta model will be presented and evaluated. Finally, we will demonstrate how to map concepts of the meta meta model to Ecore concepts.

2 Meta Modelling Languages: Requirements

Designing a modelling language implies the analysis of the requirements it should satisfy. This is the case for meta modelling languages, too. As with any modelling language, the requirements depend crucially on the purpose the language should serve. There seem to be no publications that focus explicitly on requirements for meta meta models. However, there has been work on evaluating modelling languages that can be referred to, since meta meta models define the semantics and abstract syntax of meta model*ling languages*. Studies on general requirements for modelling languages do not account for the particularities of a specific language. Instead, they are aimed at generic requirements that apply to any language. There seem to be no *empirical studies* that target generic requirements. Instead, the few empirical studies that have been conducted so far, target particular kinds of languages, mainly data modelling languages. Also, they are not aimed directly at developing requirements, but rather at the empirical evaluation of certain modelling languages (see e.g. [GoSt90], [Hitc95]). In software engineering, the main focus is on *formal* requirements a modelling language should fulfil. A typical example of this perspective is given by [SüEb97] who demand for properties such as *completeness*, *simplicity*, and *correctness*. Completeness means that all language concepts should be precisely defined. This includes constraints that apply for their application. Simplicity recommends reducing the meta model to essential concepts, hence, avoiding redundant concepts. A meta model is correct, if it allows for generating all formally valid models and for deciding whether a model is formally correct. Apparently, these formal requirements suggest formalizing a meta model. They do not, however, indicate which concepts are required and how they should be presented. In addition to that, the analysis of languages in computer science is sometimes related to their expressive power, for instance by referring to a particular layer of the Chomsky hierarchy. However, since the Chomsky hierarchy is focussing on grammars and on automata, it is not directly applicable to meta models. Approaches that focus on ontologies as a theoretical foundation for modelling languages, such as [Webe97] or [OpSe99], suggest that a modelling language should be "ontologically" complete. This implies that it should include concepts for static, functional and dynamic abstractions. Apparently, such an approach neglects the fact that a modelling language will often emphasize a particular abstraction while leaving out others on purpose. Hence, it does not need to be "ontologically complete". With respect to the design of a meta language, the claim for ontological completeness seems to be more reasonable at first sight, since a meta language should allow for specifying a wide range of modelling languages. While the specification of requirements for modelling languages faces remarkable problems [Fran98b], defining requirements for meta modelling languages is even more challenging. Although we are able to reflect upon language, it is commonly regarded as a competence that we cannot entirely comprehend ([Lore96], p. 49). While this is demanding already for distinguishing between the type and meta level languages, a further level of abstraction takes us closer to ontological or semantic primitives, which determine our own thinking.

To encounter the confusion that is imminent to the distinction of language layers, it is important to strive for a differentiated terminology. The semantic net in Figure 1 shows key terms of this report and the corresponding levels of abstractions. The numbers used to identify the levels correspond to common conventions, starting with level 0 for representations of instances. A model (level M_1) is specified by a modelling language, which in turn is – partially – specified by a meta model (level M_2). At the same time, a model is an instance of a meta model, which in turn is an instance of a meta meta model. Note that the semantic net includes a simplification: Not only a modelling language on the M_2 level, but also all meta modelling languages are comprised of a specification of their semantics and syntax. The syntax can be differentiated into abstract syntax and concrete syntax (graphical notation). A meta model serves to specify the abstract syntax and semantics only.

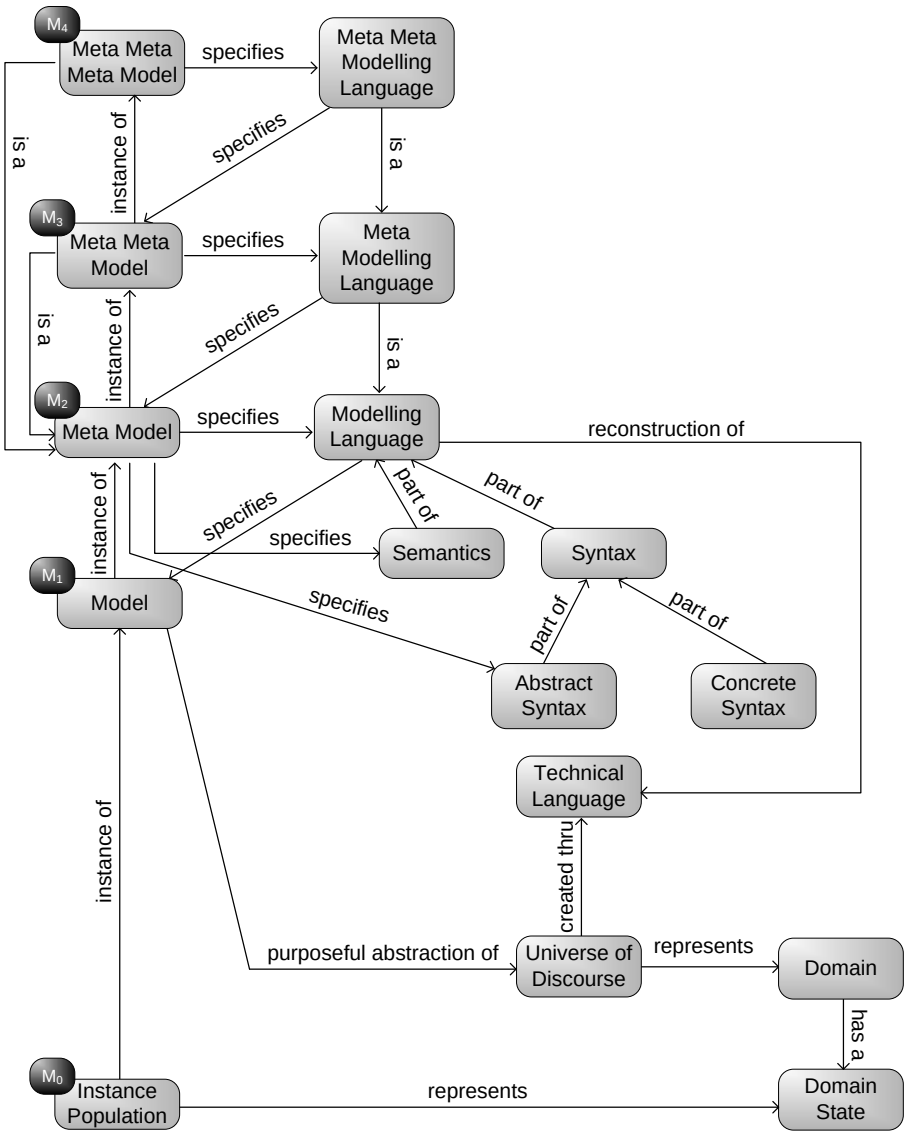


Figure 1: Semantic net of key terms and corresponding levels of abstraction

[FrLa03] present a framework for requirements of domain specific modelling languages. For analytical purposes, these criteria are differentiated into formal, user-oriented and applica-

tion-oriented requirements. Note that these are not orthogonal dimensions. These generic requirements need to be further refined for a specific language. Although the framework was designed for modelling languages (level M_2), its generic structure can be applied to meta modelling languages, too.

2.1 General Requirements for Meta Modelling Languages

Formal requirements are of special relevance for meta modelling languages, because they are a prerequisite for the (semi-) formal specification of modelling languages.

User-oriented requirements refer to the prospective users' perception of meta language concepts and their visualisation.

Application-oriented requirements are determined by the intended modelling domains and generic modelling purposes. They are related to the question whether a meta modelling language should be ontologically complete.

2.1.1 Formal Requirements

A meta modelling language should allow for the unambiguous specification of modelling languages. The resulting language specifications should also provide a foundation for the development of corresponding modelling tools. For these reasons, the abstract syntax of a meta modelling language itself needs to be specified precisely.

> **Requirement F1**: The specification of a meta modelling language should include a formal specification of its abstract syntax.

In order to foster appropriate interpretations of the modelling languages to be designed with a meta modelling language, the semantics of a meta modelling language should be defined precisely, too.

> **Requirement F2**: In the ideal case, there should be a formal specification of a meta modelling language's semantics. Hence, the specification should be complete and correct. Since a complete formalisation of semantics will sometimes imply too much of an effort, it may be sufficient to specify the semantics in a way that is regarded as unambiguous by expert users.
>
> **Requirement F3**: To foster formalisation and comprehensibility, a meta modelling language should satisfy the demand for simplicity (see also requirements A1, A2).

The specification of a meta modelling language requires a meta meta modelling language, which in turn needs to satisfy certain demands.

Requirement F4: To contribute to a precise or even formal semantics, the meta meta modelling language used to specify the meta modelling language should be a formal language. In order to avoid a further language to describe the concepts of a meta modelling language, it should feature a limited set of concepts only. This set of concepts is sufficient, if it allows for specifying all concepts required on the meta modelling language level. In

other words: The meta modelling language should be clearly simpler than the modelling languages it is supposed to describe.[1]

2.1.2 User-Oriented Requirements

Only very few people will use a meta modelling language. Designers of modelling languages are the main target group. Furthermore, designers of modelling tools might be interested as well. We assume that prospective users of a meta modelling language are experts for conceptual modelling.

> **Requirement U1**: The concepts of a meta modelling language should correspond to concepts modelling experts are familiar with. Since concepts used for creating static abstractions such as data models or class diagrams are well known within the group of prospective users, they seem to be especially suited for this purpose.

The concrete syntax of a modelling language should contribute to the comprehensibility of corresponding models. Since prospective users are expected to be familiar with the ERM or an object-oriented modelling language such as the UML, using a graphical notation that corresponds to one of these languages seems to be an adequate approach. On the other hand, there is need for distinguishing between different levels of abstraction.

> **Requirement U2**: The languages used on different levels of abstraction, such as a meta modelling language or a modelling language, should be clearly separated. Using one language for different levels of abstraction should be avoided.

Users of a meta modelling language will often deal with static modelling languages and corresponding models, e. g. with object models. This would suggest deploying a graphical notation that is different from those of languages for creating static abstractions. The following requirement reflects this conflict of goals:

> **Requirement U3**: The graphical notation of a meta modelling language should correspond to prevalent graphical notations, e.g. of data or object modelling languages. At the same time, the notation should include elements that allow for distinguishing a meta model from an object-level model at first sight (related to U1, U2).

2.1.3 Application-Oriented Requirements

A meta modelling language should be suited for specifying a wide range of modelling languages, if not any modelling language. Within our research, the focus is on languages for enterprise modelling. These include static abstractions such as object models or resource models, functional abstractions such as message flow diagrams or dynamic abstractions such as business process models. That does not imply, however, that a meta modelling language needs to offer specific concepts for creating functional or dynamic models: The purpose of a meta meta model is to model of a set of meta models. A meta model is essentially a static abstraction – even if it includes concepts that are intended for representing functional or dy-

[1] Note that this does not exclude that the metamodelling language is also used for the specification of less complex languages.

namic aspects. Therefore, a meta modelling language does not need to be ontologically complete. The claim for simplicity implies that a meta modelling language should not include concepts that are abstractions of machines, such as 'operation' or of human action, such as 'task'.

Requirement A1: A meta modelling language should offer all concepts required to specify languages in the scope of enterprise modelling.

Requirement A2: A meta modelling language should be restricted to concepts required for language design.

Requirement A3: A meta modelling language can be instantiated into meta models. Since meta models will often leave semantic gaps, the meta modelling language should also feature additional language elements that allow to express constraints on the interpretation of a meta model.

A meta modelling language is aimed at the specification of modelling languages, which will often be represented within corresponding modelling tools.

Requirement A4: In order to facilitate the development of tools, e.g. by generating object models from a meta model, the concepts offered by a meta modelling language should allow for a clear mapping to concepts used for software development. This suggests using a meta modelling language that already features such a mapping.

While a modelling language is usually focused on the description of concepts, e.g. types or classes, instead of particular instances, it is sometimes required to express characteristics that apply to all instances of a type. To give an example: The concept "process" within a language for modelling business processes serves to specify characteristics of a process *type*. While it is a well known fact that any process instance starts and terminates at a certain point in time, it is not possible to express this as an attribute of a process type. A process type may also have a certain lifetime. This is, however, clearly different from the lifetime of its instances.

Requirement A5: A meta modelling language should allow for distinguishing between different levels of abstractions. This includes especially the distinction between characteristics of types and of corresponding instances.

The elements of a conceptual model are supposed to represent concepts – or types respectively. This is for a good reason: The notion of a conceptual model implies abstraction or, to put it literally, focusing on concepts. This should foster analysing a subject with regard to its essential, invariant aspects without being distracted by features of specific instances. Hence, a conceptual model should not represent instances. Previous versions of the meta meta model were based on this assumption. As a consequence, it was not possible to specify instances as part of models. However, it turned out that this rule, although being perfectly plausible at first, is not satisfactory in all cases. For instance: A language for modelling logistic systems [Wied10] may serve to represent intermodal transportation networks. While such models should certainly abstract from particular transportation instances, it may be regarded as too much abstraction, if cities are abstracted to the concept "City". Instead, it may be preferable

to analyse a certain type of transportation net that includes particular cities. While a city such as Essen is certainly not a concept, it is not a typical instance either: Most of its relevant features such as its geographical location, its size or its transportation network will be widely invariant over a longer time period. At the same time, it serves as an abstraction over all particular locations within its geographical limits. Therefore, a meta modelling language should allow for modelling instances – even though this is a feature that should be used only after a thorough examination.

> **Requirement A6**: A meta modelling language should provide concepts that allow for representing instances.

For a more elaborate discussion of the preconditions for using instances within conceptual models see [Fran10]. Requirement A6 can be regarded as a supplement to requirement A5.

The value of a language depends on its dissemination: The more languages are specified through a meta modelling language, the better the chance to integrate these languages. Also, dissemination fosters the creation and reuse of tools that make use of a meta modelling language. In addition to dissemination, the standardization of a language contributes to protecting investments into corresponding tools and meta models. However, dissemination and standardization are orthogonal to the inherent quality of a language. It cannot be accomplished by designing a language. Instead, it requires economic and political processes. Hence, demanding for dissemination and standardization as a necessary feature would compromise the design of new meta modelling languages.

> **Requirement A7**: A meta modelling language should account for dissemination and standardization. If there are other languages for similar purposes that enjoy a higher dissemination and/or standardization, there should be a clearly defined mapping to the concepts of these languages.

Note that the requirements outlined above lack precision. In part, this is owed to the fact that one usually does not know in advance all the modelling languages that need to be specified with a meta modelling language. For this reason, it is required that any particular interpretation of the requirements should be elucidated.

3 Meta Meta Models: Prevalent Approaches

Only few meta meta models have been published so far. Some meta modelling tools, such as MetaEdit+ ([KeLy+96], http://www.metacase.com) or Cubetto (http://www.semture.de) feature meta meta concepts that allow for representing language specifications. However, these concepts are either not specified as meta meta models or not published as such. Besides, the main focus of these concepts would not be language specification, but support for tool development, which requires accounting for additional aspects such as versioning or user management. ADONIS, a further meta modelling tool, features a meta meta model. It is published, however, only in part ([JuKü+00], p. 395, translated in [Fill05], p. 4). IDEF (**I**ntegrated **Def**inition Methods) features a remarkable range of modelling languages. However, IDEF (for rationale and overview see [MaPa+92]) does not include a meta meta model. Furthermore, even the languages lack a specification through meta models. The language architecture, the UML is based on, features a meta meta model, the so called Meta Object Facility (MOF). With respect to dissemination and availability of corresponding tools, the UML is of outstanding relevance. For this reason, we will analyse whether the MOF could serve as a satisfactory meta meta model for the MEMO family of languages. In most cases, the efficient use of a modelling language recommends the use of a corresponding modelling tool. Therefore, it makes sense to account for approaches to reduce the effort required to build a tool. While meta modelling tools should offer clear advantages with respect to realizing model editors quickly, they lack a comprehensive framework that would support the implementation of additional functionality. In recent years, an open source software initiative – the Eclipse foundation – has achieved a set of tools and extensible software frameworks that have become the platform of choice for the development of modelling tools for many.

3.1 UML: Infrastructure Library and the Meta Object Facility

Obviously, the UML is the most important language for conceptual modelling. Its primary focus is on a family of modelling languages to support software systems modelling. The early versions of the UML suffered from a specification that lacked precision and consistency. With UML 2.0 the OMG aimed at overcoming these problems by providing a more elaborate specification. At the same time, the OMG launched its so called "Model-Driven Architecture" initiative (MDA), which is supposed to facilitate the generation of implementation level documents from conceptual models. This required accounting for mapping modelling concepts to implementation level concepts or for the peculiarities of implementation level artefacts, e.g. interfaces to middleware systems. These two streams of development resulted in the current structure of UML languages. Unfortunately, this structure or language architecture is all but easy to understand. On the one hand, the so called *infrastructure library* provides the basic linguistic concepts that are used to define the UML languages: "All of the

UML meta model is instantiated from meta-metaclasses that are defined in the *InfrastructureLibrary*." ([OMG06b], p. 15) While the infrastructure library is explicitly referred to as "metalanguage" or "meta metamodel" (e.g. [OMG06b], p. 11), it is called a "metamodel" at the same time. It serves to specify a basic subset of the UML that is used to define compliance level 0 (for tools that are certified by the OMG). Also, the infrastructure library is reused within the comprehensive UML specification, called *superstructure*. Hence, within the UML family of modelling languages, the infrastructure library acts both as a meta meta model and as a meta model: "The *InfrastructureLibrary* is in one capacity used as a meta-metamodel and in the other aspect as a metamodel, and is thus reused in two dimensions." ([OMG06b], p. 15) At the same time, the language definition is reflexive, since the infrastructure library is specified through a subset of UML class diagrams. Note that this overloading of a language with different levels of abstractions is a clear violation of requirement U2.

The confusion gets even worse with the introduction of the *Meta Object Facility* (MOF, [OMG06a]). MOF is intended to serve as a cornerstone of the MDA initiative. Following the idea of defining language packages, MOF is separated into the essential MOF (EMOF) and the complete MOF (CMOF). For this purpose, it allows to specify all UML languages. It also includes concepts that correspond to artefacts that are required for integration purposes, such as Interface Definition Languages, the Common Warehouse Model (CWM), the Enterprise Java Beans (EJB) model and XMI. Furthermore, it features transformation rules to these representations. These rules can be applied to any language that is specified through the MOF. Hence, MOF seems to be a meta modelling language (or at least a meta meta model). However, this is not clear. While the MOF is explicitly intended to act as a meta meta model for instantiating meta models ("… MOF is an example of a meta-metamodel." ([OMG06b], p. 16), there is a disclaimer in the documentation: „In the four-layer metamodel hierarchy, MOF is commonly referred to as a meta-metamodel, even though strictly speaking it is a metamodel." ([OMG06b], p. 16). The following excerpt from the MOF specification ([OMG06a], p. 11) illustrated the confusion caused be the UML language architecture (or rather: the lack of an architecture): "In particular, EMOF and CMOF are both described using CMOF, which is also used to describe UML2. EMOF is also completely described in EMOF by applying package import, and merge semantics from its CMOF description. As a result, EMOF and CMOF are described using themselves, and each is derived from, or reuses part of, the UML 2.0 Infrastructure Library." Figure 2 shows a central part of the EMOF ([OMG06a], p. 33). Exactly the same model is presented as the part of the infrastructure library that defines „the constructs for class-based modelling" ([OMG06b], p. 93).

It seems that the difference between the infrastructure library and the MOF is mainly related to their purposes. On the one hand, the infrastructure library serves to provide basic concepts needed for specifying more elaborate concepts of UML languages. On the other hand, the MOF – while serving to specify languages, too – is aimed at providing a framework that facilitates the integration of modelling tools with other systems used for the development of (distributed) systems. This includes the definition of transformation rules.

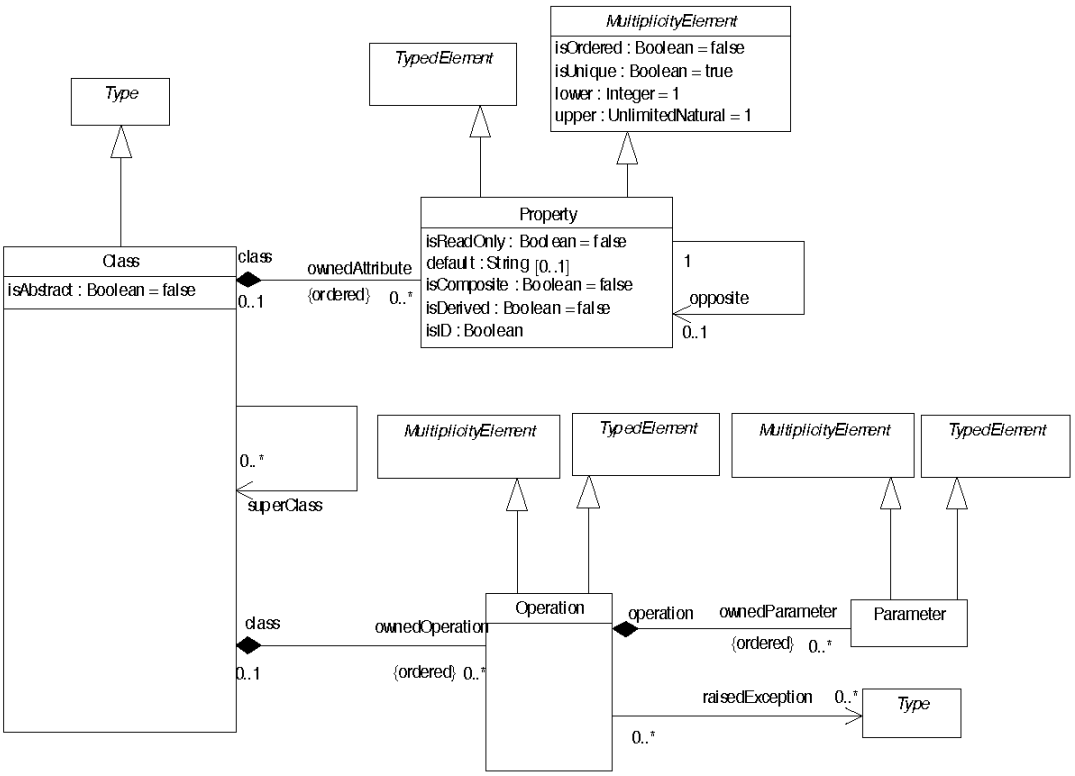


Figure 2: EMOF or part of the infrastructure library respectively ([OMG06a], p. 33; [OMG06b], p. 93)

A closer look at EMOF reveals some surprising features. Firstly, its representation includes multiple copies of classes. The semantics of an entity type (or a class) depends on its attributes and the associations it is involved in. For this reason, an entity type should be depicted only once within a model. Hence, multiple copies of an entity type make it difficult to catch its meaning. It is amazing that the OMG violates this well known principle of good modelling practice. Figure 3 shows a revised version of EMOF that avoids multiple copies of entity types.

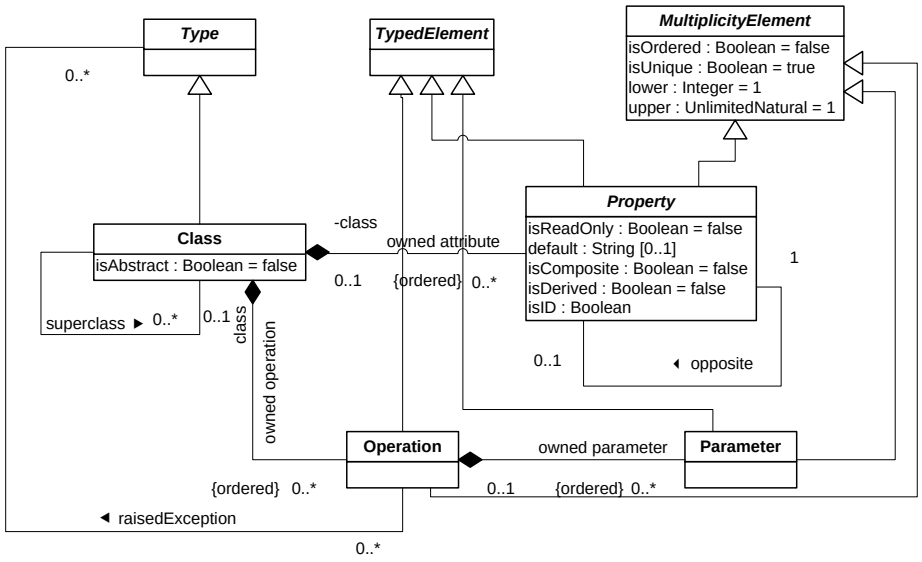


Figure 3: Revised version of EMOF

In addition to that, the EMOF specification suffers from unclear semantics. Supertypes such as *Type* or *TypedElement* remain unspecified. Concepts such as *Operation* or *Parameter* are apparently underspecified. To give a few examples: It is not explicated what the attributes mean that are assigned to *Property*. Nor does the reader get any support with understanding the meaning of the association named "opposite". Also, it is not clear what "default : String [0..1]" is supposed to mean. If EMOF is interpreted as a meta meta model, the pre-initialisation of attributes, such as "isReadOnly : Boolean = false", is confusing. Does that mean that an instantiation of the corresponding class would allow for this attribute having the value "false"? With respect to the purpose of a meta meta model, i.e. the definition of a modelling language, it seems beside the point to include concepts such as *Operation* or *Parameter*, since they imply the existence of software – a clear violation of requirement A2 and requirement F3. CMOF, which serves as the meta language to specify EMOF, is clearly more complex. This is a violation of requirement F4. It may be that these semantic gaps are filled somewhere in the jungle of cross-referencing UML specifications. However, the MOF specification itself [OMG06a] is not complete. CMOF is not only used to specify EMOF. It also serves for "more sophisticated metamodeling" ([OMG06a], p. 31). Figure 4 shows "key concrete" classes of CMOF. It seems that concepts used both in EMOF and CMOF do not need to share the same meaning. In EMOF, *Class* is specialized from *Type*. According to Figure 4, *Class* within CMOF is not specialized from *Type*, but from *Classifier*. The concept *Property* is not specified consistently either. *Association* is specialized from *Relationship*. However, the semantics of *Relationship* is not specified at all. This is the case for *StructuralFeature*, too.

While the CMOF is supposedly a comprehensive ("complete") model, it leaves semantic gaps as well. Superclasses such as *Relationship*, *Type* or *StructuralFeature* remain unspecified. While the might be specified somewhere else, this is not what one would expect from a document that is to specify MOF.

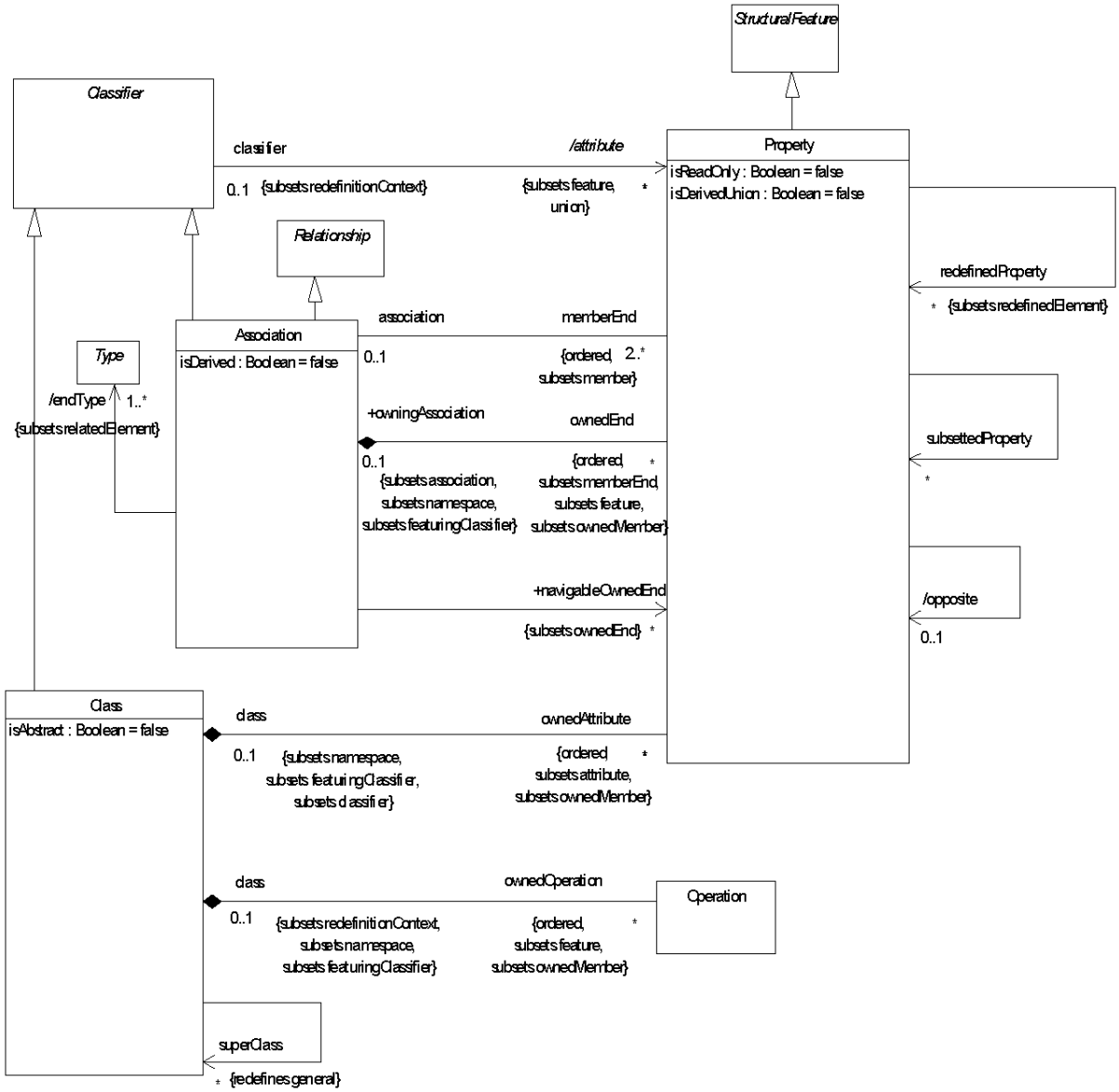


Figure 4: CMOF: "Key concrete classes" ([OMG06a], p. 47)

Evaluating the language architecture of the UML requires accounting for some interrelated peculiarities:

No clear differentiation between language specification and tool design: While the UML is primarily aimed at a standard for modelling languages, an essential purpose of this standard is to facilitate the certification of tools. Therefore the meta models include concepts such as operations or events, which are intended to guide the implementation of modelling tools (see example in Figure 4). As a consequence, the EMOF (as well as the infrastructure library) includes the concept `Operation`.

Not intended for specifying languages for enterprise modelling: The UML is primarily a family of modelling languages for software development. Therefore the focus is on concepts that allow for abstractions of software systems. As a consequence, the meta meta model includes specific concepts required for software system modelling.

Not directly intended for specifying modelling languages: While both EMOF and CMOF are explicitly intended to support the specification of meta models (see e.g. [OMG06a], p. 31), they are not directly used for specifying the UML itself. The UML is specified using the infrastructure library which is also reused in the MOF. It seems that the main purpose of the MOF is to define object models as a foundation for tool integration. Hence, the MOF is rather intended for defining meta models that define the concepts to be shared by a set of tools that are to be integrated. Nevertheless, the MOF can be regarded as a meta meta model, since it serves to describe meta models.

Evolutionary, pragmatic approach: The UML resulted from multiple contributions from industry and academia. This included accounting for specific interests and preferences, which compromised a concise and coherent language design. While numerous misconceptions and specification gaps were eliminated in the latest version (2.0), the UML still suffers from this burden of its evolution.

Table 1 shows the evaluation of the UML infrastructure library (or the MOF respectively) against the requirements for meta modelling language suggested above.

Despite the shortcomings that the evaluation reveals, the UML language specifications cannot be neglected for the specification of the MEMO meta modelling language. This is already implied by requirement A6. Also, the development of modelling tools requires modelling languages for software design. It is very likely that the UML will be the language of choice for this purpose. Therefore, the MEMO meta models need to be mapped to UML class diagrams. Furthermore, due to the dissemination of UML tools, it can be reasonable to replace the MEMO-OML [Fran98c] with the UML object modelling language. This would require integrating the corresponding UML concepts with MEMO modelling languages.

Table 1: Evaluation of MOF and the UML infrastructure library respectively

(-: not satisfactory; o: accounted for; +: good; ++: very good)

Req.	Eval.	Comment
F1	o	Apparently, the languages that serve as meta modelling languages make use of the infrastructure library. At the same time, the infrastructure library is used to specify the abstract syntax of UML class diagrams. While this is not convincing, the abstract syntax of UML class diagrams is defined (rather) precisely. Therefore, from a pragmatic point of view, it can be regarded as sufficiently specified.
F2	-	In the core specification document [OMG06a], the specification both of EMOF and CMOF is not complete and leaves the language designer with many questions concerning the semantics.
F3	-	Both EMOF and CMOF include concepts that are related to modelling tools. Therefore, both models are more complex than they needed to be, if they were intended for modelling language specification only.
F4	-	CMOF, which is used to specify the EMOF, is clearly more complex than the EMOF. At the same time, the MOF is defined using the infrastructure library which is not only more complex, but is also used for the same purpose as MOF, i.e. to specify meta models.
U1	+	The meta meta model is specified in the same notation as the UML itself. Hence, its representation can be expected to be comprehensible for many language designers.
U2	-	The same concepts are used on different levels of abstraction. The language architecture adds to the confusion.
U3	-	Different levels of the language architecture make use of the same notation.
A1	+	The UML meta language concepts should be sufficient for specifying enterprise modelling languages.
A2	-	The UML does not only serve as a language specification, but also as a reference for certifying tools. Therefore, the language concepts are not clearly separated from concepts that relate to tool specification only.
A3	+	The UML meta language includes the OCL, which can be used to add further constraints on language specifcations.
A4	+	Since the UML languages are specified with a subset of the UML object modelling language, the transformation into class diagrams needed for the development of modelling tools is very convenient (if it is required at all).
A5	o	The UML features powertypes. However, there is no precise specification of the concept (see Sect. 4.2).
A6	-	The UML allows for representing instance-level data, e.g. within interaction diagrams. However, the objects used in interaction diagrams are not representations of concrete instances. Instead, they are abstractions in the sense that they show prototypical instances to visualize behaviour. The MOF does not contain any specific concept for modelling instances.
A7	++	The UML is the outstanding standard in conceptual modelling for software design.

3.2 Eclipse Foundation: Ecore

The Eclipse initiative supports the development of model editors by providing a software framework that provides a generic architecture and generic functionality. Adapting the framework to develop a specific model editor starts with specifying a meta model of the cor-

responding modelling language. In order for the framework to interpret the meta model appropriately, it needs to be specified using predefined concepts. For this purpose, Eclipse includes a conceptual model, named Ecore. While Ecore is called a "metamodel"[2], a close look at it reveals two contrasting characteristics. On the one hand, it shows features of a meta meta model, because it serves to describe meta models. On the other hand, it is neither a meta nor a meta meta model, but an object model built as a conceptual foundation for modelling tools. The classes that constitute the model include operations that support introspection and transformation (see Figure 5). Furthermore, the classes include references to Java language constructs. Abstract classes are depicted as grey boxes.

Analysing Ecore reveals a number of surprising if not odd features. For instance: The abstract class `ETypedElement` includes the attributes `lowerBound` and `upperBound`, which serve to indicate the minimum and maximum number of values that must or may represent a feature such as an attribute. In addition to these, there are two other attributes, which are redundant: `many` indicates whether there may be multiple values; *required* serves to specify whether at least one value is mandatory. The attribute `container` of `EReference` is redundant, too: "A reference is a container if it has an opposite that is a containment."[3] Other features focus on particular implementation level aspects, which one would normally not include in a language specification, e.g. the attributes `containment` or `resolveProxies` in `EReference`.

However, evaluating Ecore as a meta meta model (or even as a meta model) would not do justice to its very purpose. Ecore is a model of an actual implementation. It guides users of the framework in representing the modelling language they want to build an editor for. The framework includes a plethora of generic functions to manipulate, navigate and transform graphical models that consist of interconnected modelling elements. To adapt the framework to the requirements of a specific modelling editor, the corresponding modelling language has to be reconstructed as a net of associated objects instantiated from the classes specified in Ecore. These objects are transformed into classes that represent the meta types within the meta model of the modelling language to be supported by the tool. The object states serve to define the semantics of these classes (see Figure 5). After that, the concrete syntax has to be defined by assigning graphical representations to the language concepts. The functionality of the resulting modelling tool can be further refined by selecting from options offered by the framework or by modifying/adding code. Table 2 shows the evaluation of Ecore according to the requirements suggested in 2.1.

Due to the remarkable productivity gains promised by Eclipse and its still growing dissemination, the specification of a meta meta modelling language recommends to account for

[2] http://www.eclipse.org/modeling/emf/?project=emf (accessed on July 8th 2008)

[3] http://download.eclipse.org/modeling/emf/emf/javadoc/2.5.0/org/eclipse/emf/ecore/EReference.html#isContainment() (accessed on July 8th 2008)

Ecore – not as a meta meta model or even a meta modelling language, but as a representation that is relevant with respect to building modelling tools. Hence, there should be a transformation of the concepts specified in a meta meta model – as well as of the concepts in corresponding meta models – to Ecore. Independent from that, one major concern remains: The documentation that is provided with Ecore is restricted to the description of the Java classes. This shortcoming includes the unusual terminology. Terms such as “instance class” or “meta object” are used without further explanation. This is definitely not satisfactory.

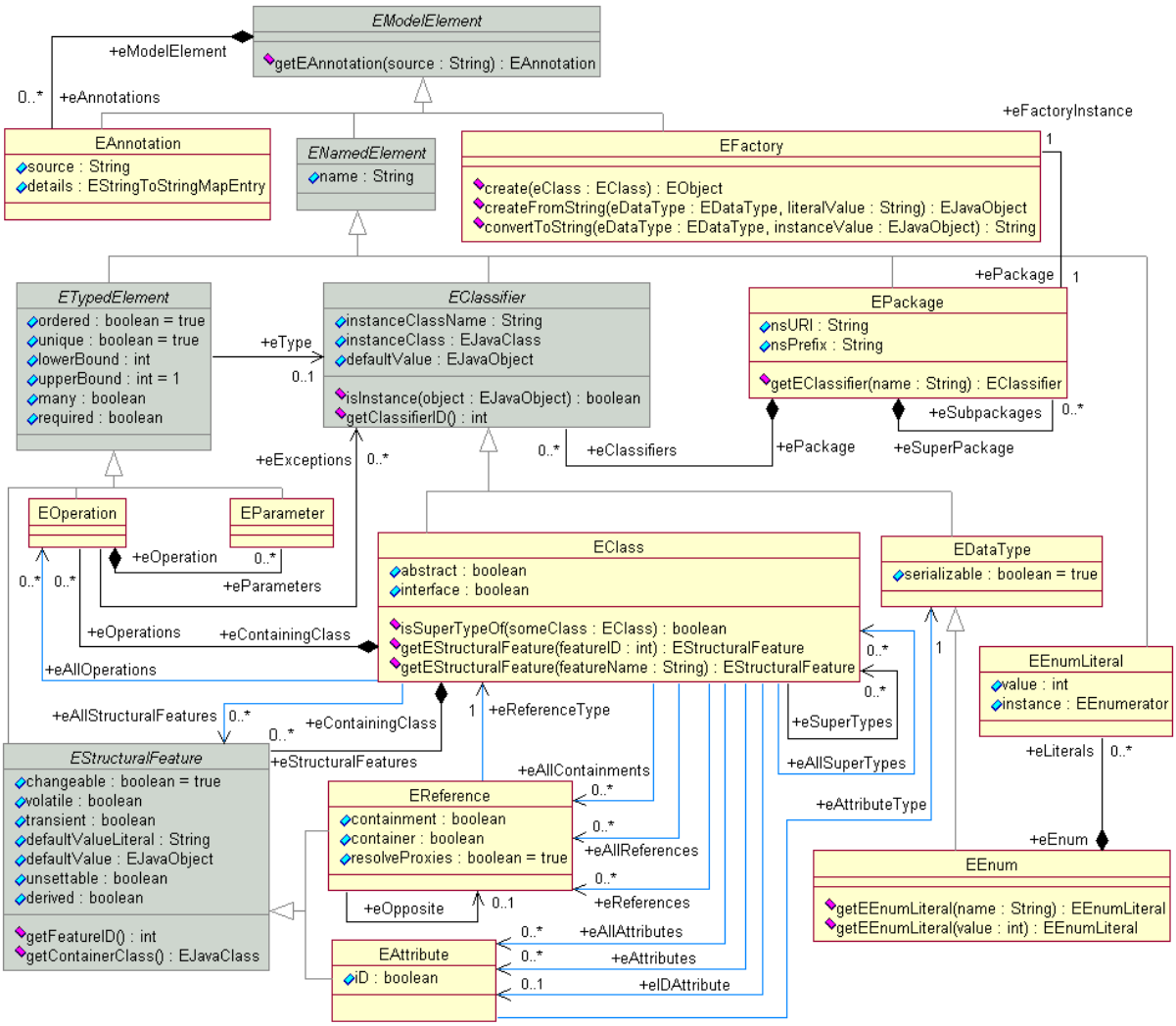


Figure 5: Ecore[4]

[4] http://download.eclipse.org/modeling/emf/emf/javadoc/2.4.0/org/eclipse/emf/ecore/package-summary.html (accessed on July 8th 2008)

Table 2: Evaluation von Ecore

Req.	Eval.	Comment
F1	+	Ecore is specified using a variant of UML class diagrams, the abstract syntax of which is formalized to a great extent.
F2	o	The language specification of the UML still includes some ambiguities. This is, e.g. the case for the semantics of specialisation/generalisation. However, by mapping Ecore to a programming language (Java) which is based on a formal specification (finally through the machine model it runs on), the Ecore models feature a precise semantics. Unfortunately, Ecore lacks concepts required to conveniently specify certain features of meta models.
F3	-	The UML is certainly not a language that satisfies the demand for simplicity.
F4	-	This criterion cannot be directly applied to Ecore since there is no explicit meta meta modelling language. Instead, Ecore is specified as a UML class diagram. Nevertheless, the UML is clearly more complex than Ecore itself.
U1	o	Ecore is presented through a variant of UML class diagrams. Hence, its syntax and (ostensible) semantics are easy to understand for those who are familiar with the UML.
U2	-	An appropriate interpretation is jeopardized through the fact that on the one hand, Ecore is represented as a class diagram, on the other hand an instance of Ecore is meant to be interpreted as a meta model. Hence, Ecore is an overloaded representation: It is located on the type (or class) level and at the same time it shows features of a meta meta model.
U3	o	Ecore uses the notation of UML class diagrams. This is for a good reason, because it is a UML class diagram. However, since it should be interpreted as a meta meta model, too, this notation is also confusing.
A1	o	On the one hand, Ecore is not intended to specify a modelling language. Instead, it serves to reconstruct a language specification for the purpose of developing a tool using an existing software framework. On the other hand, the object model that serves as a language reconstruction can be enhanced through additional specifications or code. Hene, Ecore provides a sufficient foundation for specifying tools for enterprise modelling.
A2	o	Since Ecore should not be regarded as a means to specify modelling languages, there should not be any confusion. However, it could be mistaken as such – in interpretation that is fostered by calling it a meta model.
A3	+	Ecore can be supplemented by OCL statements.
A4	++	This criterion marks a clear advantage of Ecore: As soon as a language is reconstructed using Ecore, a major step to develop a corresponding editor is accomplished.
A5	-	Ecore does include concepts that allow for such a differentiation. However, it could be modified using UML powertypes.
A6	o	Ecore is an object model that does not specify the level of abstraction (see A5). Instead, its semantics is overloaded by including meta-level and type-level data. Since the interpretation of the level of abstraction is – to a large extent – left to the software developer, it is possible to add an interpretation where an Ecore concepts such as “EClass” is instantiated into instances.
A7	++	The Eclipse initiative is a de facto standard for the development of modelling tools.

4 Language Specification

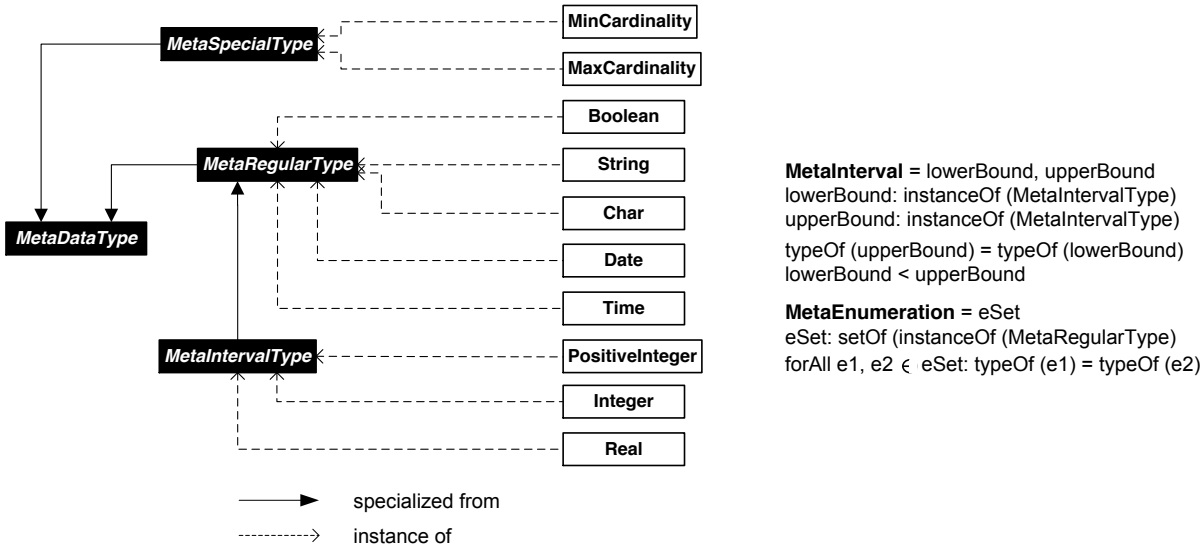
The evaluation of the UML language specification concepts and of Ecore has shown that neither one is satisfactory for the specification of modelling languages. Ecore is not a modelling language at all but only a class diagram that can be interpreted as the representation of a meta model. The UML infrastructure library or the MOF are not intended to serve especially as meta modelling languages. They are not introduced and used as pure meta meta models. Also, they do not feature a specific graphical notation. The main purpose of the MOF is to provide a foundation for tool interoperability. For this reason, we decided to further use our own meta modelling language. However, some revisions are required. On the one hand, they relate to shortcomings of the previous version. These include specification gaps (req. A5) and especially the lack of concepts that help with expressing different levels of abstraction (req. F3). On the other hand, they are concerned with the graphical notation. The revised version features a graphical notation that allows for clearly distinguishing meta models from models on the object level (req. U3).

All languages within MEMO are specified through this common meta language. It is specified through a meta meta model. While an explicit meta meta model is not mandatory for specifying meta models – as the bootstrapping approach used within the UML language architecture demonstrates – we decided for a clear separation of different language levels. Such a separation allows for defining a clearly more comprehensible language architecture. This is not only helpful for developers. We use MEMO for teaching purposes. The clear separation of language levels helps students to identify and understand the different levels of abstraction to account for. The use of a meta meta modelling language provides advantages over other approaches to language specification. Firstly, it makes use of the same paradigm. That should help prospective language users – modellers – with understanding the specification. Secondly, a meta model provides a good foundation for the implementation of modelling tools, because it can be reconstructed as an object model in a straightforward way. In order to foster the integration of the modelling languages and to support the construction of integrated modelling languages, MEMO features a language architecture.

4.1 Basic Data Types or Domains

The meta modelling language includes a set of basic data types. Their semantics is not specified any further. For this purpose, it is referred to the implementations of corresponding data types in prevalent programming languages. Note that we do not need an operational semantics for specifying meta models. Therefore, the data types can be regarded as domains that define sets of values. It is not possible to define a subset of a basic data type by specifying a range or an enumeration of values. It is assumed that there is no need for specifying subsets on the meta meta level. On the meta level it is possible to specify subsets which apply to the corresponding type level. This is prepared for by specializing *MetaDataType* into *Me-*

taRegularType and further on into *MetaIntervalType*. Types that are instances of *MetaIntervalType* allow for specifying subsets through intervals. *MetaInterval* serves to define the structure for initializing intervals. *MetaEnumeration* serves to instantiate a set of values of the same type that serves to specify attributes. *MetaInterval* and *MetaEnumeration* are specified in a formal pseudo-language (see Figure 6) In addition to data types featured by most programming languages, the types *Date* and *Time* are included. Furthermore, two more special types – to be instantiated from *MetaSpecialType* – are introduced, *MinCardinality* and *MaxCardinality*. They are defined as sets (see Figure 6). The basic data types or domains respectively used within the MEMO meta modelling language are depicted in Figure 6. Note that the instantiation relationships serve only the purpose to provide for using the abstraction *MetaDataType* (and its subtypes) within the meta meta model. It does not express a specific meaning apart from that.



$$MinCardinality = \{0\} \cup PositiveInteger$$

$$MaxCardinality = \{'*'\} \cup PositiveInteger$$

$$\forall\, x \in PositiveInteger \rightarrow x < '*'$$

Figure 6: Basic data types used within the meta meta model

4.2 Intrinsic Features

On the one hand, specifying a meta model requires reflecting upon the ontological essence of a term. On the other hand, it recommends taking into account that instances of a meta concept are types. Sometimes, this results in the problem that the essence of a term includes features that do not apply directly to the type level. Instead, they apply to the instances represented by a type. For example: A language for modelling product types includes a meta type "PhysicalProduct", which has attributes like "name" or "type" and further optional features. Within a particular model, it is instantiated to a certain product type, e.g. "TV Set", which includes the instantiation of attributes from corresponding meta types. While we know that every physical product has a weight, measurements or a serial number, these materialized

features do not apply to the corresponding product type, because product type is an abstraction. Since a meta type may only define features that can be instantiated to describe features of a type, it is not possible to express features that apply to the instances of this type only. Assigning these features to every instance would not only ignore an obvious abstraction, it would also result in redundancy. This problem is well known in conceptual modelling. One approach to deal with it is the conception of a so called "power type" (also referred to as "powertype"). According to Odell ([Odel98], p. 28) "a *power type* is an object type whose instances are subtypes of another object type." This is a confusing definition that needs further explanation. Figure 7 illustrates, how a powertype could be used to overcome the abstraction conflict between type and instance features.

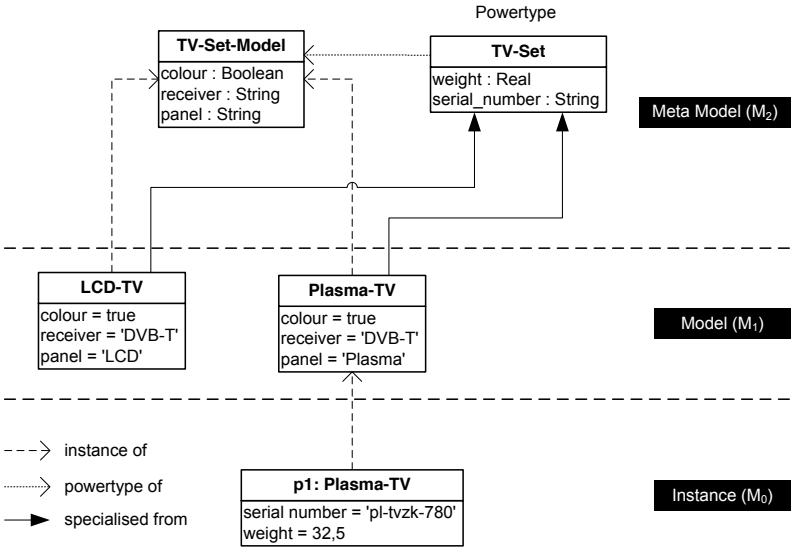


Figure 7: Exemplary use of a power type – adapted from [Odel98]

The UML includes the concept of a powertype as well ([OMG05], p. 223, p. 335). Drawing upon an example given by Odell, a power type is regarded as an additional classification schema: "For example, the metaclass TreeSpecies might be a power type for the subclasses of Tree that represent different species, such as AppleTree, BananaTree, and CherryTree." ([OMG05], p. 34). The specification of the current version of the UML provides a further example: "For example, a Bank Account Type classifier could have a powertype association with a GeneralizationSet. This GeneralizationSet could then associate with two Generalizations where the class (i.e., general Classifier) Bank Account has two specific subclasses (i.e., Classifiers): Checking Account and Savings Account. Checking Account and Savings Account, then, are instances of the power type: Bank Account Type. In other words, Checking Account and Savings Account are both: instances of Bank Account Type, as well as subclasses of Bank Account." ([OMG07], p. 57) While powertypes allow for coping with the problem outlined above, they come with a major disadvantage: There is no concept in natural language that would correspond to a powertype. Instead, the concept of a powertype is introduced only for providing a conceptual workaround. The concepts of a language for con-

ceptual modelling should correspond to concepts prospective language users are familiar with. This is certainly not the case with powertypes. In [Scha08] the concept of "class template" is presented. While it is similar to powertypes, it provides a more intuitive conception of the additional abstraction it allows for.

Similarly, Atkinson and Kühne criticize that the concept of a powertype seems artificial and thereby increases the complexity of a model, while compromising its comprehensibility. Therefore, they suggest a conception they call "deep instantiation" [AtKü07]. "Deep" refers to the possibility to define that a concept is supposed to be instantiated "deeper" in an instantiation hierarchy. It is based on a construct they call "clabject": "… we refer to such constructs as clabjects (class and object) and represent them using a combination of notational conventions from UML classes and objects." ([AtKü07], p. 10). A clabject can be specified using "fields" that either represent a meta type attribute – which is supposed to be instantiated and initialized on the type level – or a feature of instance of the type. These two meanings of a field are differentiated through so called "potencies". A potency indicates the number of instantiations of the corresponding meta types – and its instances respectively – to be taken before the field itself may be instantiated. A potency of 1 applies to the meta type attributes that are supposed to be instantiated on the type level. A potency of 2 means that the attribute applies only on one level further down the instantiation chain. A potency of 0 can be assigned to a (meta) type in order to mark it as abstract. The concept of a clabject is illustrated in Figure 8. The potency values – printed in red – that are assigned to two fields of the clabject "TV-Set" are supposed to be instantiated only on the instance level.

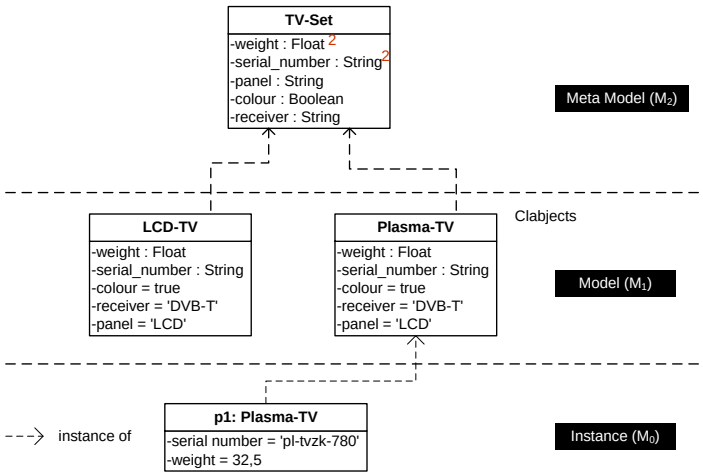


Figure 8: Example modelled with clabjects – according to [AtKü07]

Compared to powertypes, clabjects have the clear advantage that they generate less complexity. A clabject corresponds to the common (overloaded) concept of a class in natural language. It forces the modeller to explicitly clarify the level of abstraction intended with each feature of the class. However, the concept of a clabject has some shortcomings, too. While differentiating "fields" through "potencies" is a powerful instrument for expressing different levels of instantiation, it is still difficult to understand because it is an artificial conception.

Sometimes, not only attributes (or "fields") are subject of delayed instantiation, but also associations. The additional challenge generated by accounting for associations is illustrated in Figure 9. While one could associate (meta) classes and define when their fields are supposed to be instantiated, the question remains how to express multiplicities for the deeper layers. Consider the following example: We assume that every class of "TV-Set" can be assigned one particular receiver type (instance of "Receiver") only. This would be expressed through corresponding multiplicities on the M2 layer. Further on we assume that a particular TV (instance of instance of "TV-Set") can be assigned one to many different particular receivers (instance of instance of "Receiver"). In this case, there would be need to specify these multiplicities somehow. The concept of a clabject, as it is presented in [AtKü07], does not include a solution to this problem. While potencies allow for expressing multi-level instantiation chains, it is disputable whether potencies > 2 are required in modelling practice. Doing without potencies would then reduce the complexity of a language.

Against this background, the concept of a clabject is slightly modified for its representation in the MEMO meta meta model. Firstly, we do not use potencies. This decision is based on the assumption that – at least for the purpose of specifying modelling languages – potencies > 2 are not needed. Also, we do not speak of "fields". Instead, a (meta) type may have (regular) attributes that apply to its instances or "intrinsic attributes" that can be instantiated only with the instances of its instances. Intrinsic attributes correspond to fields with a potency value of 2. Furthermore, our concept includes associations: An association that gets effective only with the instances of the entity types it connects is called an "intrinsic association". An entity type that must not be instantiated directly, but only on the level below the one it is presented on, is called an "intrinsic type". Note that all attributes of an intrinsic type are intrinsic by default for the entire lifecycle of that type. Also, all associations an intrinsic type is involved in must be intrinsic, too.

Figure 9 shows the representation of a modified example, where *Receiver* is modelled as an associated type with regular attributes and an intrinsic attribute. Defining attributes of associated types as intrinsic has the following implications: The association is implicitly defined for each level of abstraction that is covered by the attributes or intrinsic attributes respectively. In the example, this means that the meta type *Receiver* is associated to the meta type *TV-Set*. Its instance is associated to instances of *TV-Set* etc. In the example shown in Figure 9, intrinsic features (attributes, associations or entity types) are marked by grey boxes.

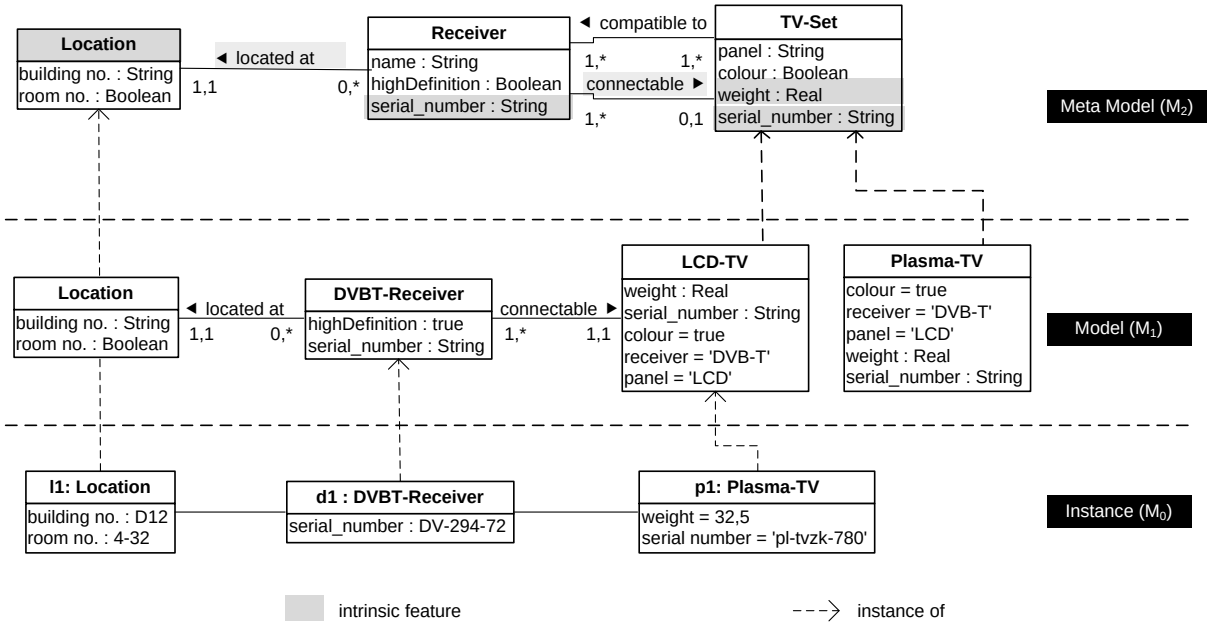


Figure 9: Example modelled with intrinsic attributes, associations and types

4.3 "Language-Level Types": Concepts to Model Instances

While the specific purposes of conceptual models vary to a large extent, they have in common that they are aimed at abstractions. Hence, they should not represent particular instances, the state and even the existence of which may change over time. However, sometimes it can make sense to include representations of instances into a model. This is the case, if instances in a targeted domain satisfy the following conditions (for a more elaborate discussion of this subject see [Fran10]):

- The purpose of a model recommends accounting for instances.
- Abstracting instances to the type level would not fit the intended applications of a model anymore.
- The existence and the relevant state of an instance are stable throughout the intended lifetime of a model.

Possible examples of instances that could be included into models are cities, countries, organisational units (e.g. "Marketing Department") or organisations (e.g. a particular company). With respect to specifying modelling languages, this consideration leaves two choices. On the one hand, the possibility to model instances could be ignored since it is required in exceptional cases only. This would help to keep the meta modelling language simpler and yet easier to apply. On the other hand, one could provide a meta modelling concept, which in fact serves to model types and not meta types. This would result in overloading the meta modelling language, which comes with the challenge to specify additional constraints that prevent ambiguity – and make the language more difficult to understand and use at the same time. I have hesitated for long to decide for the second option – mainly for the reason

that the experience with designing modelling languages that we gained during the last years suggests that requirement 6 can hardly be ignored.

At first sight, it may appear that a concept that allows for specifying types which are instantiated into instances on the model level is not required because intrinsic features or intrinsic types could serve the same purpose. However, this is not the case. An intrinsic feature or type serves to defer instantiation of meta types. Hence, on the model level, intrinsic features are not instantiated. For this reason they are not suited to represent instances within a model. Figure 10 illustrates the difference between intrinsic features and "language-level types".

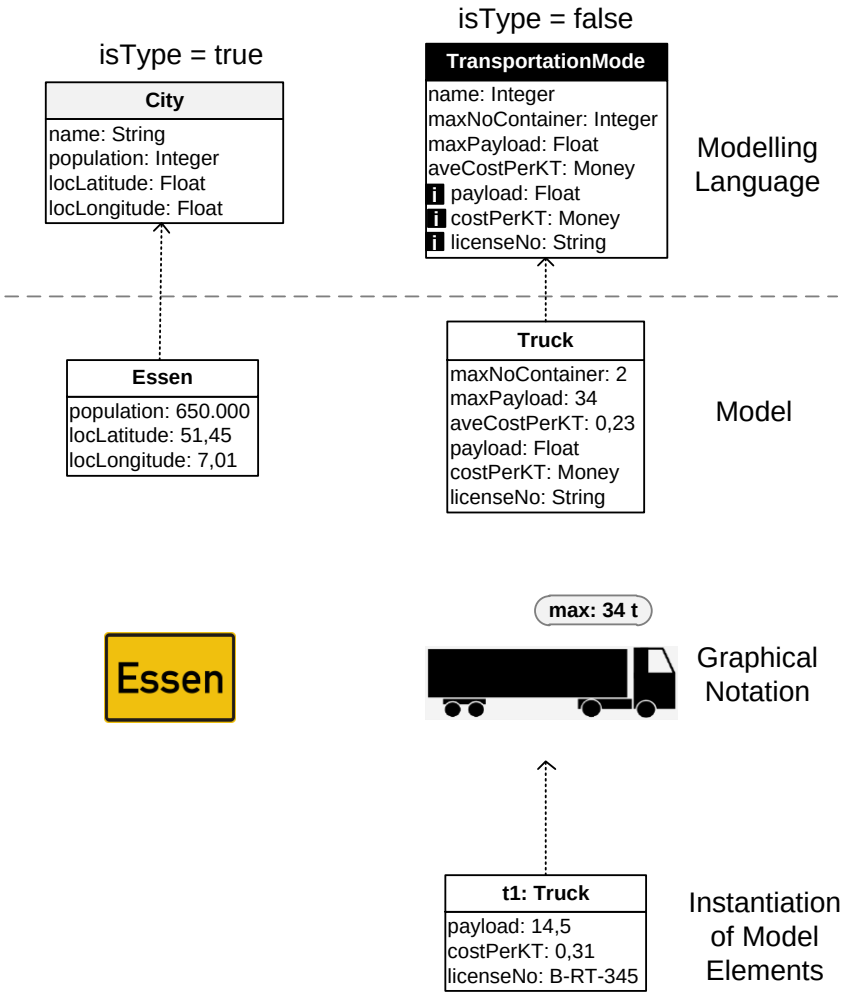


Figure 10: Comparison of intrinsic features and modelling of instances

4.4 The Meta Meta Model

The concepts used to specify the meta meta model as well as the graphical notation correspond to the Entity Relationship Model (ERM). Unfortunately, the requirements for specifying modelling languages result in further concepts that considerably increase the complexity of the meta meta model. Among other things, they include concepts to specify intrinsic and obtainable features, specialisation relationships, and multiplicities that can be assigned to attributes. In addition to that, the meta meta model is overloaded in the sense that it includes

two levels of abstraction: At the core of the meta meta model is the abstraction *MetaEntity*. Usually, its instances are meta types. However, in rare cases, it may be instantiated into types. The level of abstraction represented by a particular instance of *MetaEntity* is indicated by the state of the attribute *isType*. Note that this kind of overloading is certainly not an elegant choice. Instead, it reflects the ambiguity of the subject. *MetaEntity* is associated with concepts that are used to define the semantics of an instantiated meta type (or, in exceptional cases, types) – such as *MetaAttribute* or *MetaAssociationLink*. Note that they are instantiated into types. Most concepts defined through the meta meta model are well known from meta modelling languages. To support a clear distinction of the meta meta model from models on other levels of abstraction (in correspondence to requirement U3), the concepts of the meta meta model are represented as rectangles with a grey background. In order to further specify the semantics of a meta model and to comment on its concepts, the meta meta model includes the concepts *Comment* and *Constraint*. To allow for an unambiguous identification of comments and constraints, they can be assigned identifiers. While a comment is written in natural language, a constraint should be specified in a formal language in order to foster precision and to allow for machine interpretation. The OCL [OMG06c] is a good choice for this purpose, because it is supported by various tools. While both *Comment* and *Constraint* apply to the meta type level, they are not instantiated into meta types (or types) but into instances, which are assigned to a meta model. Hence, they are on a different level of abstraction as compared to other concepts of the meta meta model. This is expressed through a white background, which corresponds to the representation of object or data models.

MetaAssociationLink serves the specification of associations between instances of *MetaEntity*. Each instance of *MetaAssociationLink* can be specified through a name, a role, a minimum cardinality and a maximum cardinality. Each instance is associated to exactly one further instance of *MetaAssociationLink*. Both instances are associated to exactly one instance of *MetaEntity*. Hence, only binary assocations are supported. The name that can be assigned to an instance of *MetaAssociationLink* serves as a designator of the corresponding association. Each one of the two names is supposed to be read in the direction towards the associated instance of *MetaAssociationLink*. Usually, one designator will be sufficient. The attribute *role* allows for assigning a role to an association end (see below). The attribute *predecessor* within *MetaAssociationLink* serves the specification of modelling languages that support dynamic abstractions. If *predecessor* is set to true, the corresponding concept is supposed to occur before the one it is linked to through the opposite instance of *MetaAssociationLink*. Note that there is no specific semantics specified for it. It might seem appropriate to exclude cyclic associations. However, a cycle on the type level may make sense in case of multiple instances. Hence, this type of association merely serves to make corresponding meta models more comprehensible.

The semantics of specialisation – which is restricted to single generalisation (single inheritance) – corresponds to that of object-oriented programming languages: A *MetaEntity*

instance ME1 that is specialized from the *MetaEntity* instance ME2 inherits all features from ME2. However, different from logical subsumption – and the prevalent notion of specialisation in natural language – instances of ME2 would not be instances of ME1. Instead, every instance of an instance of *MetaEntity* is specified through exactly one (meta) type. This restriction is a tribute to the semantics of specialisation in programming languages. Although this concept of specialisation is the source of misinterpretations and problems (see e.g. [Fran03]), it was chosen to foster the transformation of meta models to object models used for developing corresponding modelling tools. The attribute *isSingleton* of *MetaEntity* serves to express whether a *MetaEntity* may be instantiated into one type only. Note that this constraint should be used only after thorough considerations. Optionally, multiplicities can be assigned to attributes – represented through the attributes *minCard* and *maxCard* of *MetaAttribute*. Within the meta meta model this is expressed through the multiplicity [0..1]. Particular instances of *MetaEntity* or attributes or associations can be specified as intrinsic. If an instance of *MetaEntity* is specified as intrinsic (attribute *isIntrinsic* = true), all its attributes during its entire lifecycle as well as all associations it is part of are intrinsic, too. In the case of attributes, the boolean attribute *isIntrinsic* within *MetaAttribute* serves to define whether an attribute is intrinsic. The Boolean attribute *isIntrinsic* within *MetaAssociationLink* can be used to mark an association as intrinsic. The boolean attribute *derivable* within *MetaAttribute* serves to specify whether the value of an attribute may be deferred from other parts of a meta model. It reflects the fact that the level of detail used for specifying a meta model may vary. For instance: A meta type such as "Organisational Unit" may include the attribute "numberOfPositions". The corresponding value may be assigned directly to the type that was instantiated from "Organisational Unit". It could, however, be calculated from the position types and the corresponding numbers of instances – provided, these details were represented in the model. The attribute *simulation* within *MetaAttribute* allows for indicating that an attribute is introduced for simulation purposes. This could be, for instance, the case with attributes such as "averageAvailabilityPerDay" of a certain resource type. Sometimes, it may be possible that the value of an attribute can be obtained from external sources, e.g. a database. For example: A business process type could include the attribute "averageRevenues", which would serve to represent the average revenues generated by an instance of this type. If this value can be obtained from an external information system, this can be expressed by setting the attribute *obtainable* within *MetaAttribute* to true.

The constraints that apply to the meta meta model are defined through OCL expressions in order to foster the creation of a tool for editing meta models (see chapter 6). Figure 11 shows the MEMO meta meta model. An instance of *MetaModel* is composed of any elements that are instantiated from concrete subtypes of *MetaConcept*. It defines the namespace for all named entities. Note that it is not exactly a language concept. It can be instantiated into a particolar meta model, which could be instantiated into its models. However, specific features of models, such as the times they were created or modified, are not accounted for – e.g.

through associating `MetaModel` with `MetaAttribute`. Instead, this is regarded as a feature that is relevant for the development of corresponding tools (see chapter 7). The concept of role is rather overloaded within conceptual modelling (for a comprehensive analysis of the role concept in conceptual modelling see [Stei00], especially p. 61 ff.). In the meta meta model it is accounted for only for one pragmatic reason: Sometimes, it is not possible to unambiguously identify a particular end of an association, which may be required to specify a constraint. In this case, it is possible to assign a role to an entity type that forms the end of an association. A role can support the identification of an association end only, if its name is unique within the associations that end at the corresponding instance of `MetaEntity` (Constraint 10). The meta meta model itself includes two roles that are assigned to `MetaEntity`.

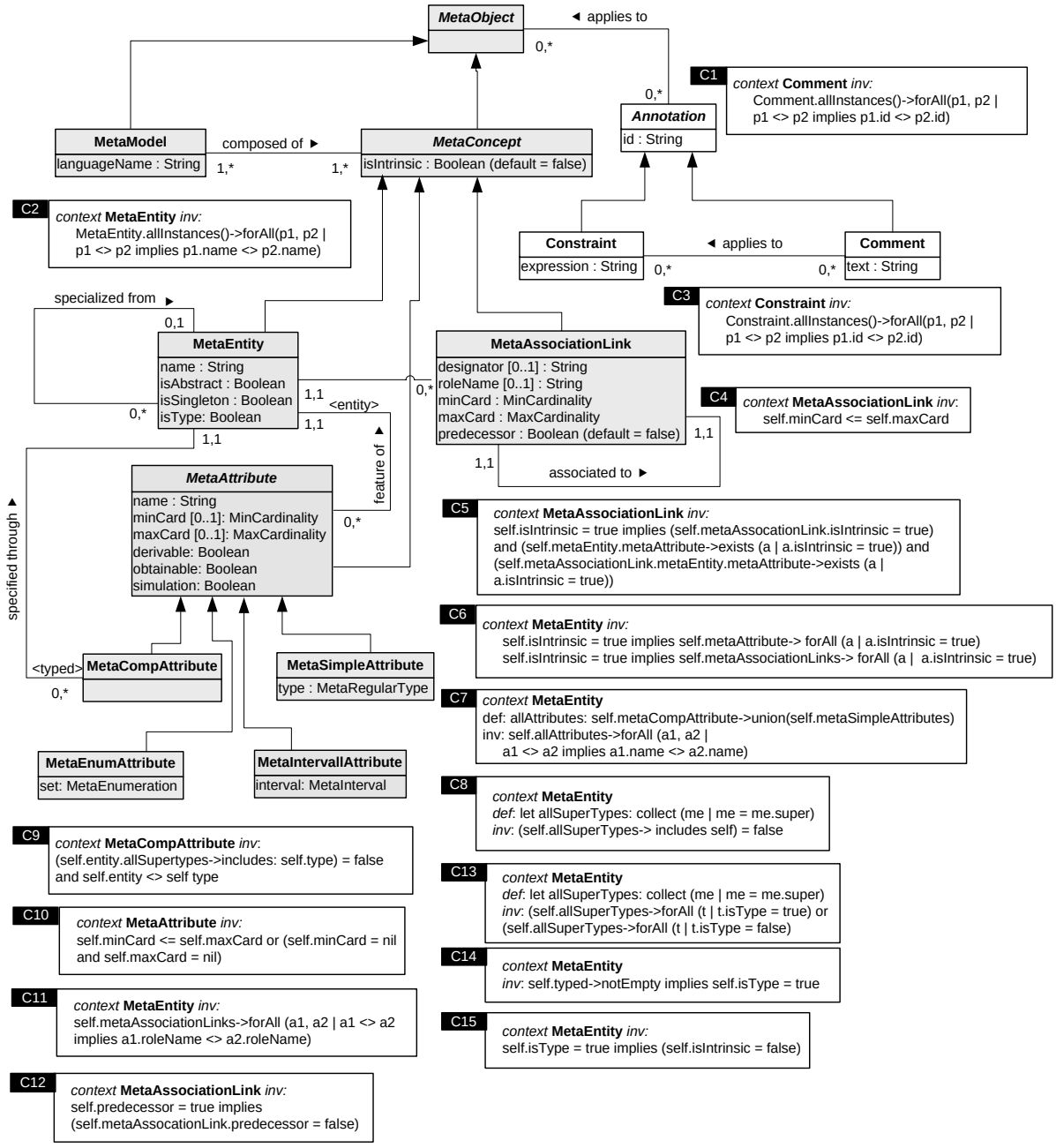


Figure 11: The MEMO meta meta model

Constraints C1 and C3 express that identifiers of constraints and comments have to be unique. Constraint C2 defines that names of instances of *MetaEntity* have to be unique, too. Constraint C7 specifies that names of attributes (either instances of *MetaCompAttribute*, *MetaIntervalAttribute* or *MetaSimpleAttribute*) have to be unique within the scope of the entity type they are assigned to. Constraint C3 expresses that the minimum cardinality has to be less or equal to the corresponding maximum cardinality. If an instance of *MetaEntity* is marked as intrinsic (through the attribute *isIntrinsic*), then all its attributes and all associations it is involved in must be marked as intrinsic, too (constraint C6). Specialisations of instances of *MetaEntity* must not be cyclic (constraint C8). Constraint C9 serves to avoid cyclic specifications, which could result in non-terminating initialisation procedures: A *MetaCompAttribute* must not be specified through the *MetaEntity* it is a feature of, nor through one of the *MetaEntities*, the associated *MetaEntity* is specialized from.[5] In addition to that, the *MetaEntity*, a *MetaCompAttribute* is specified through, must represent a type, i.e. its attribute *isType* must be set to true. An association is either intrinsic or not. Therefore, if the attribute *isIntrinsic* within an instance of *MetaAssociationLink* is initialised as intrinsic, the corresponding instance of *MetaAssociationLink* has to be intrinsic, too. Furthermore, the associated entity types must be intrinsic or at least one of their respective attributes must be intrinsic. This is expressed through constraint C5. Multiplicities are optional for attributes. If they are use, the minimum cardinality must be smaller or equal the max cardinality (constraint C10). Constraint C11 specifies that the name of a role must be unique within the set of associations the corresponding entity type is part of. Constraint 12 prevents two associated *MetaAssociationLinks* from both having set their attributes *predecessor* to true at the same time. Constraint 13 expresses that within a specialisation hierarchy of instances of *MetaEntity* all elements have to be either on the type level or on the meta type level. Constraint 14 serves to assure that an instance of *MetaCompAttribute* is specified by an instance of *MetaEntity* that represents a type. Constraint 15 prevents that an instance of *MetaEntity* that is specified as type can be specified as intrinsic at the same time.

4.5 Reference Instantiations: Auxiliary Types

In order to promote the integrity of conceptual models, the design of domain-specific modelling languages recommends the use of types that include more semantics than basic types to specify attributes of meta types. These types can be instantiated from *MetaEntity* – with "isType" set to true. Hence, they are not part of the meta modelling language in a strict sense. Instead, they are specified by the meta modelling language. Nevertheless they are concepts that are used for specifying meta models. The semantic net in Figure 12 illustrates the relationship between meta modelling language, auxiliary types and meta models.

[5] For a thorough analysis of OCL concepts to specify transitive closures see [Baar03].

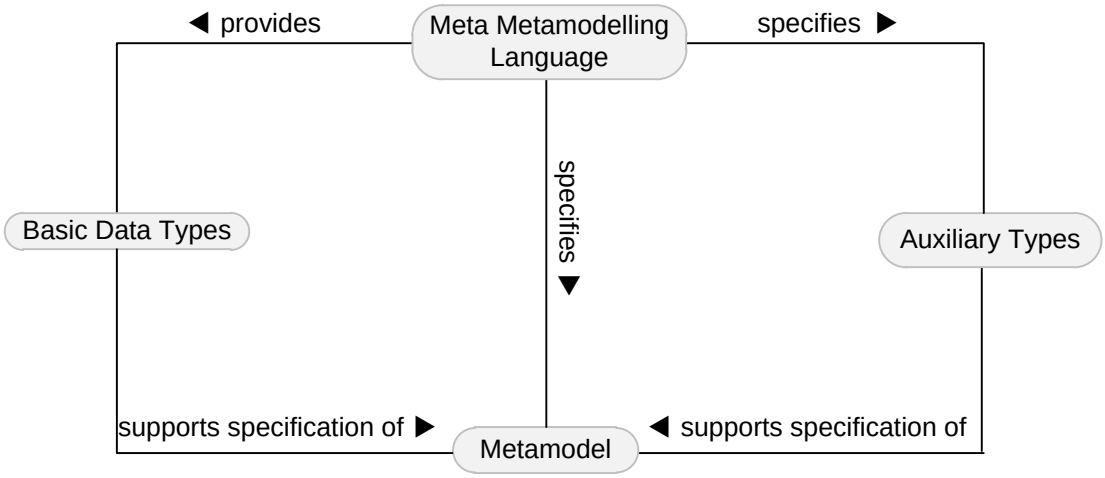


Figure 12: Placement of auxiliary types

While it is possible to define a set of more specific types for each modelling languages, the quest for reuse and integration suggests the specification of types as a common reference for a set of modelling languages. This is especially the case for types that are generic in the sense that they do not reflect requirements of particular domains. But with respect to the specification of languages for enterprise modelling is it useful, too, to define domain-level types as common references: The set of languages for enterprise modelling target similar domains with a substantial amount of overlapping and a distinctive need for integrating models designed in different languages.

The excerpt of a set of reference types shown in Table 4 evolved from the specification of the MEMO OrgML. Note that the present set should not be regarded as complete. Instead, it rather serves as a common, further growing repository. With every new modelling language and with every modification of existing languages new domain-specific further types may be added. Reference types may also serve as abstractions that allow for refinements at a later time. The type `Mission` in Table 3, for instance, serves to specify the mission of organisational units or projects. At present, its specification remains on a high level of abstraction. Later on, more semantics may be added – without compromising the semantics of attributes that were specified with a previous version.

Table 3: Preliminary set of generic reference types

TimeUnit unit: {#second, #minute, #hour}	This generic type serves to specify the unit that is implicitly referenced by a corresponding value that specifies the number of units.
Duration unit: TimeUnit dur: Float	`Duration` allows for specifying attributes that represent a time interval.

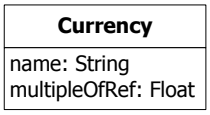
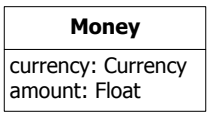
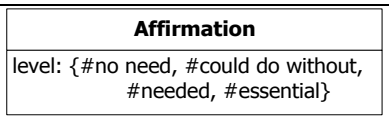
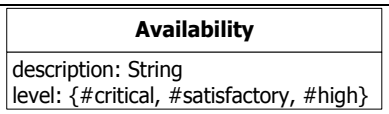
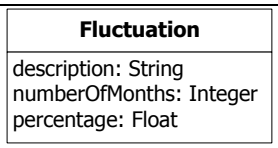
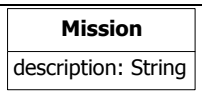
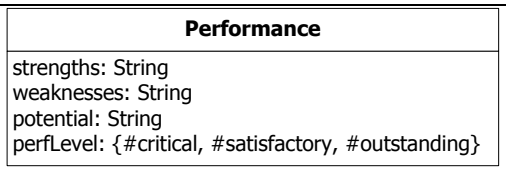
	This type allows for representing currencies. An instance c of `Currency` is defined by its name and a factor a unit of a reference currency has to be multiplied by to produce a unit of c.
	Money allows for representing an amount of money on a higher level of abstraction (and semantics) by including the corresponding currency as an instance of `Currency`.

Table 4: Preliminary set of domain-specific types

	Whenever an attribute represents an evaluation, `Affirmation` can be used to express the corresponding judgement. For instance: An organisational unit could include the attribute "subjectOfOutsourcing" to indicate whether outsourcing this type of organisational unit is a useful option. Specifying it with `Affirmation` would contribute to reuse and model coherence. In addition to that, it would allow for convenient and safe revisions at a later time.
	This type serves to specify attributes that represent an availability – of a resource or a product.
	`Fluctuation` is primarily intended to represent the fluctuation of employees within a certain organisational position or role. It could be applied to resources in general, too.
	An organisational unit, a project etc. may be characterized by a mission. The type `Mission` serves to specify respective attributes. In its current state, this auxiliary type does not provide an elaborate specification.
	Various types of analysis require accounting for the performance of subjects such as organisational units, products etc. `Performance` defines a concept that does not only allow for defining a performance level on an ordinal scale, but to also describe strengths, weaknesses and potential.

4.6 The Graphical Notation

The concrete syntax or graphical notation of the meta modelling language is much like the one already used for drawing the meta meta model itself. For the specification of textual designators/annotations we use a Bachus-Naur form (see Table 5). The non-terminal symbols are used within the graphical illustration of the notation (see Figure 13 and Figure 14). Notice that we do not bother with specifying a few basic non-terminal symbols – like *LowercaseLetter*, *UppercaseLetter*, *LineFeed* etc. or *String*.

Table 5: Representation of textual elements

Basic Symbols & Composites	<digit> ::= 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
	<positiveInteger> ::= {< digit >}
	<infiniteNumber> ::= '*'
	<separator> ::= '..'
	<lowerString> ::= <LowercaseLetter> <String>
	<upperString> ::= <UppercaseLetter> <String>
Multiplicity	<maxCardinality> ::= <PositiveInteger> | <infiniteNumber>
	<minCardinality> ::= <PositiveInteger>
	<multiplicity> ::= '(' <minCardinality> separator <maxCardinality> ')'
Names & Designators	<EntityName> ::= <upperString>
	<AttributeName> ::= <lowerString>
	<backwardArrow> ::= '◀'
	<forwardArrow> ::= '▶'
	<designator> ::= <lowerString>
	<backwardDesignator> ::= <backwardArrow> <designator>
	<forwardDesignator> ::= <designator> <forwardArrow>
	<roleName> ::= <lowerString>
	<constraintkey> ::= 'C' <number>
	<commentkey> ::= 'C' <number>

To satisfy the demand for a clear visual distinction between meta models and models on the object level (req. U3), instances of `MetaEntity` are represented in a different layout: Instead of a black font on a white (or grey respectively) background, a white font on a black background is used to depict the name of the instance. If a `MetaEntity` is instantiated into a type (indicated by a respective value of its attribute `isType`), the name of the resulting type is printed in black on a grey background. Specialisation relationships are depicted using a common notation: an arrow that is directed towards the generalized concept. In order to foster the distinction from UML class diagrams, the arrowhead is filled in black. This is the

same notation as the one used in the meta meta model already. Usually, one will not use more than one designator for an association. However, it is possible to assign one designator for each direction. Comments and constraints are represented through specific boxes with attached identifiers. As an option, they can be linked to a selected model element through a dotted line. They are expressed through strings. In the case of constraints it is recommended to use OCL expressions. Roles within associations are depicted as grey, rounded boxes with their names printed in white. Intrinsic features are a concept that is specific to the MEMO meta modelling language. Their semantics is substantially different from ordinary modelling concepts. Therefore they need to be marked clearly. This is accomplished through a white "i", which is printed in a black box. The box is attached to the names of attributes and entity types or to the designators of associations. If an association carries two designators, both should be marked accordingly. In the case of intrinsic entities, the box has a white frame to make its shape visible. If an association is not assigned a designator, the box is placed next to the edge that represents the association. Abstract entity types are marked by printing their names in italic.

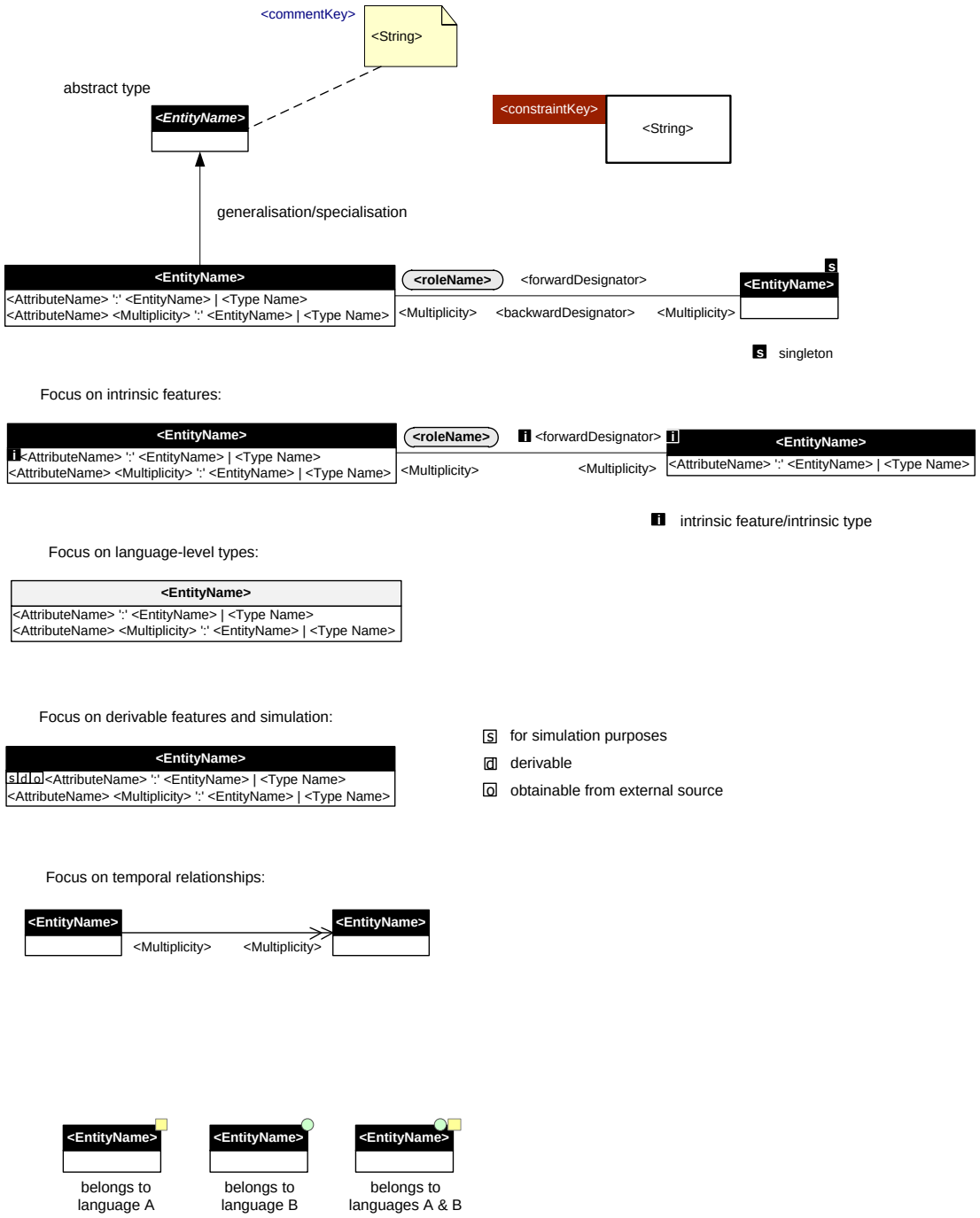


Figure 13: Elements of the graphical notation

Enterprise models require the use of various languages that need to be integrated. For this purpose, the corresponding meta models have to be merged. In the case of complex meta models, this constitutes a substantial challenge even for experienced language designers. In order to contribute to a more transparent representation, the elements of a meta model can be marked by a symbol that indicates the modelling language they belong to. Since the set of languages that can be specified using the MEMO meta modelling language is not determined, it is not possible to define symbols in advance. Instead, the language designers have

to cater for that. Figure 14 shows possible options for marking entity (meta) types that are part of the MEMO OrgML meta model.

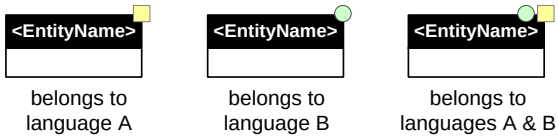


Figure 14: Options to mark the elements of a meta model as belonging to a particular language

4.7 Examples

The application of the MEMO meta modelling language allows for constructing a wide range of meta models. The following examples serve to illustrate the use of both basic concepts that will be required for most meta models as well as the use of more sophisticated or rarely required concepts. The first example, depicted in Figure 15 shows a meta model of the ERM. This is certainly not a typical application, since the MEMO meta modelling language is supposed to be used for the specification of more complex meta models.

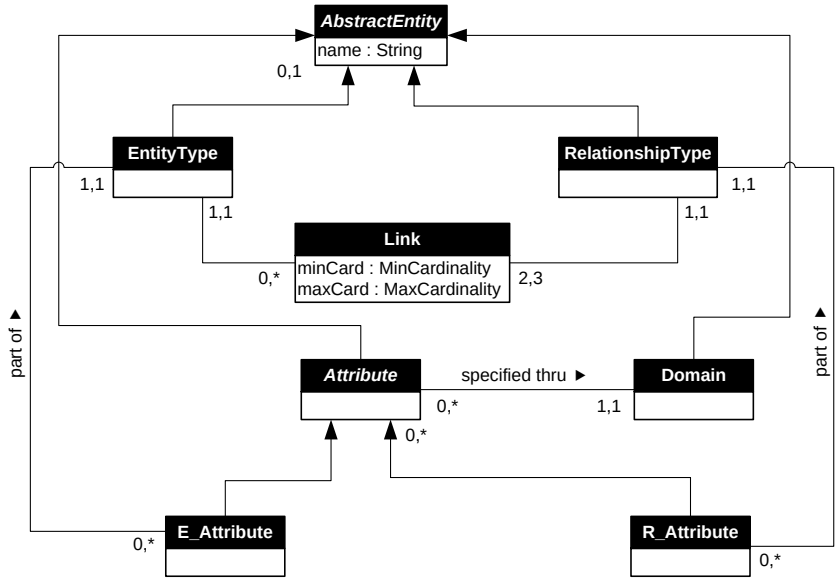


Figure 15: A meta model of the ERM

If modelling languages need to be integrated, the corresponding meta models will usually be placed side by side in order to look for common concepts. The example in Figure 16 shows the integration of the ERM with the DFD. The symbols used to distinguish both languages make use of different colours only. The example illustrates the use of roles and constraints, too.

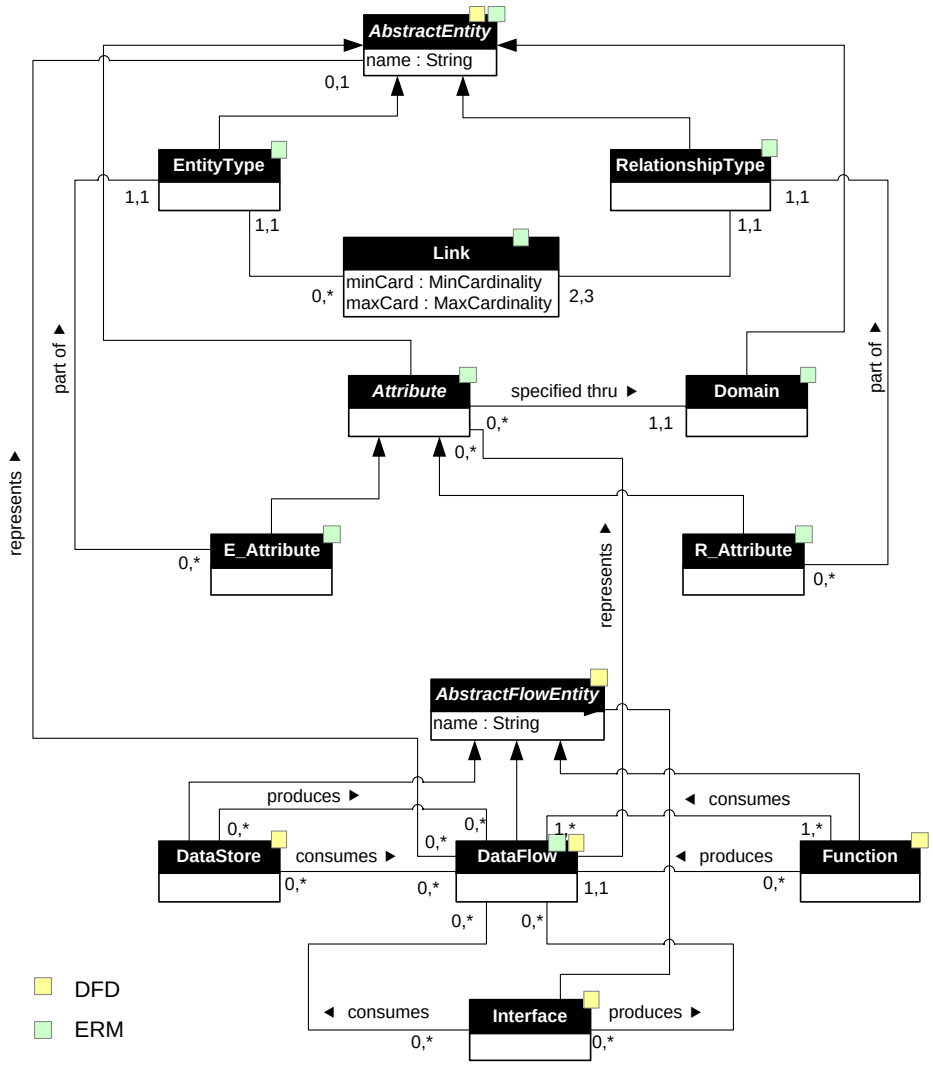


Figure 16: Differentiating two meta models through specific symbols

The use of intrinsic features is a more sophisticated option offered by the MEMO meta modelling language. The example in Figure 17 shows all concepts that can be used to express intrinsic features: intrinsic entity types, intrinsic attributes and intrinsic associations. The example shows a simplified application of the MEMO OrgML. In order to illustrate the meta model's semantic, the type and instance level are represented, too. The meta type `Process` is associated to the meta type `OrgUnit`. To specify a particular organisation model, `Process` is instantiated into `OrderManagement` and `OrgUnit` into `MarketingDepartment`. Both meta types contain intrinsic attributes that are not instantiated on the type level, but only on the instance level. The time a process is started or terminated is not a feature of a type, but of a particular instance. This differentiation is not that obvious for the instantiation of `OrgUnit`. This is because `MarketingDepartment` is defined as singleton (indicated through the little box with an 'S' on top of the box that represents the type). The type does not have a particular number of employees, nor was it founded at a certain date. Instead, these features belong to the single instance of `MarketingDepartment`. Note that `Market-`

ingDepartment does not have to be defined as singleton. If, for example, a multinational corporation specifies a reference organisation structure for all its national subsidiaries, then there would be multiple instances. To express that every organisational unit, no matter of what type it is, is headed by one employee, the type *Employee* could be associated with *OrgUnit*. However, *Employee* does not apply to the meta level. Therefore, it is specified as intrinsic. Note that one should be very careful with using this option, because normally a meta model should not include types.

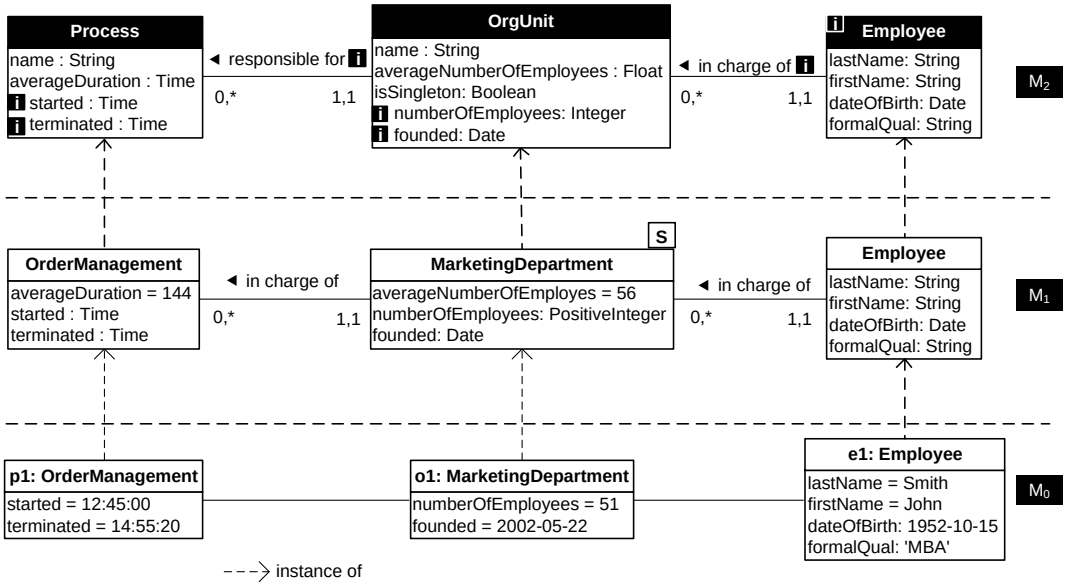


Figure 17: The use of intrinsic features

Figure 18 illustrates the use of language-level types, which are instantiated to instances on the model level. The concepts shown on the M_2 level could be part of a language for modelling logistic networks. *RegularService* serves to specify types of regular services provided by a shipper. In a corresponding model, a type of a regular service would be described by the cities it serves. The cities as well as the respective countries are – for plausible reasons – modelled as instances. The meta type *RegularService* includes intrinsic feature to allow for describing particular instances.

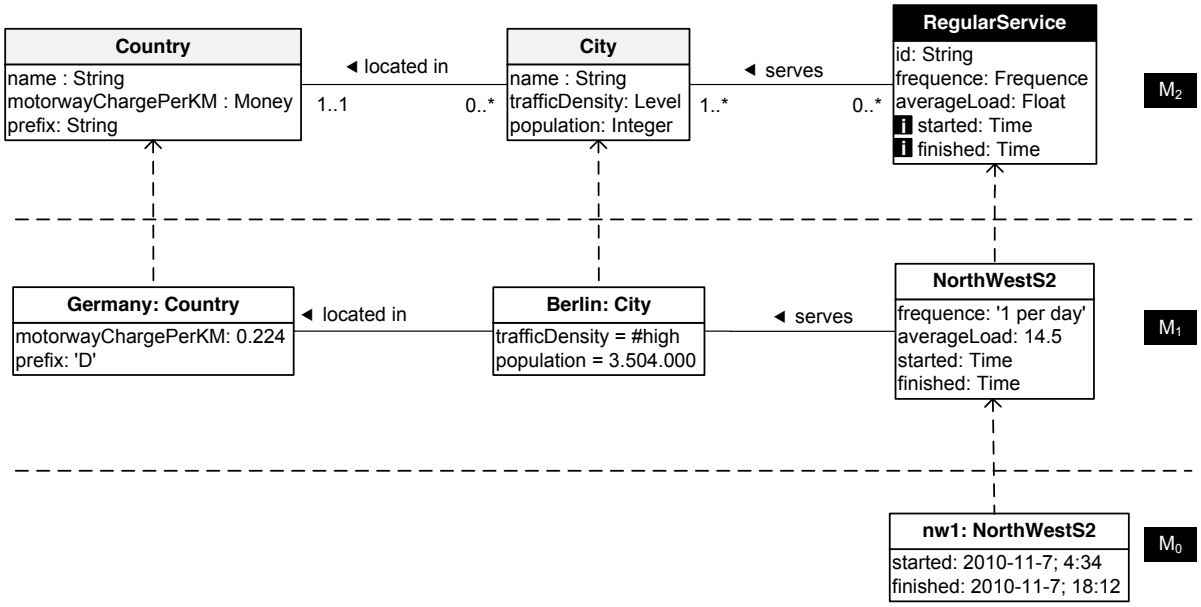


Figure 18: The use of language-level types

4.8 Preliminary Evaluation

The MEMO meta modelling language was designed to meet the requirements presented in 2.1. Table 6 gives an overview of how well the requirements are satisfied. With respect to some criteria (e.g. U1 or U3), such an assessment suggests to involve a larger number of language designers. This has not happened yet.

Table 6: Evaluation of the MEMO meta modelling language

Req.	Eval.	Comment
F1	+	The abstract syntax of the MML is formalized.
F2	+	The semantics of the MML is formalized to a large extent.
F3	o	Although the MML includes a few specific concepts, such as intrinsic features, it is restricted to a small set of concepts. Unfortunately, the complexity of the meta meta model has grown over time with the emergence of additional requirements for the specification of modelling languages. As a consequence, it is not as simple as originally intended. Nevertheless, the additional concepts are regarded as necessary to account for requirements A2 and A3.
F4	o	The MML does not make use of an explicit meta meta modelling language. The language concepts used to specify it correspond to the ERM, which is enhanced by a few concepts only – such as specialisation and abstract entity types. While more than a dozen OCL constraints counter inappropriate interpretations, they are not sufficient for a comprehensive formalisation.
U1	+	Modelling experts should be familiar with most concepts offered by the MML, because they correspond to the ERM. However, many prospective users will probably not know intrinsic features.
U2	+	The MEMO language architecture provides a clear differentiation of levels of abstraction.
U3	+	The specific graphical notation of the MML promotes a clear differentiation of meta models from models on other levels of abstraction.
A1	o	The MML was specifically designed for specifying languages for enterprise modelling. Its core concepts have been successfully used for this purpose for several years. Nevertheless, it cannot be excluded that in future times requirements will occur, the MML does not account for.
A2	+	The MML's sole purpose it the specification of meta models.
A3	+	The MML makes use of the OCL, which can be applied to add further constraints on language specifications.
A4	+	The MML supports a clear mapping to object-oriented implementation languages. It also supports a transformation of meta models into Ecore representations (see 6).
A5	+	The MML features intrinsic features, the semantics of which is precisely defined. Intrinsic features are also accounted for by specific notation elements.
A6	+	The MML allows for specifying a concept of a meta model as type. Hence, it is possible to specify modelling languages that offer concepts to model instances.
A7	o	The MML is clearly not a standard. However, its instances (meta models) can be transformed into Ecore representations or other standard representations such as XMI – which, however, may cause the loss of semantics.

5 The MEMO Language Architecture

MEMO consists of an extensible set of modelling languages. They are integrated through shared concepts, which in turn are specified through the common meta modelling language. This construction allows for a coherent integration of new languages that supplement the existing set of languages. It provides a foundation for designing a corresponding set of integrated modelling tools, too. Figure 19 shows the two levels of the language architecture and the corresponding models on the type level: The common meta meta model specifies the abstract syntax and semantics of the MEMO meta modelling language. It is instantiated into the meta models specify the abstract syntax and semantics of the MEMO modelling languages, such as the Object Modelling Language (OML, [Fran98c], [Fran98d]), the Organisation Modelling Language (OrgML), the Strategy Modelling Language (SML) or the IT Modelling Language [Kirc08]. Further MEMO languages target modelling of resources [Jung08] or various aspects of corporate knowledge management [Scha08]. Note that it may be required to reconstruct the architecture occasionally. If, for instance, two languages share a growing number of concepts, merging them into one language will improve the architecture's transparency. The bottom layer represents the models that are created by the modelling languages.

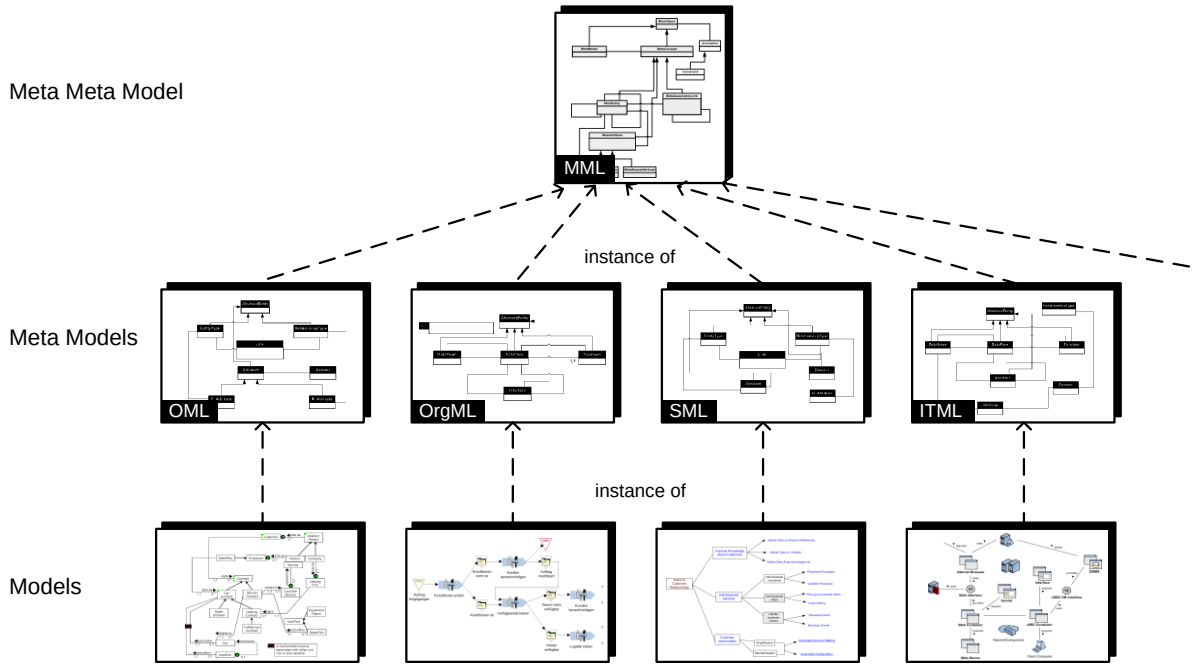


Figure 19: The MEMO language layers

In addition to providing for an integrated set of modelling languages, the architecture should also account for the construction of a tool environment: While the meta models can be regarded as a conceptual foundation for the design of a corresponding modelling tool, they cannot be used directly for this purpose. Instead, they need to be reconstructed as object

models. These object models do not only represent the meta models, they need to be enhanced with tool specific features, e.g. features that relate to versioning, to user management or to analysing and transforming models. In case a tool is supposed to support collaborative modelling in a distributed setting, there is need to include concepts that allow for model locking on various levels of detail. In order to provide a conceptual foundation for a tool suite that allows for integrating various modelling editors, the object models that correspond to particular meta models are merged into an integrated object model (see Figure 20). The various editor of an integrated tool provide particular views on instances of this object model.

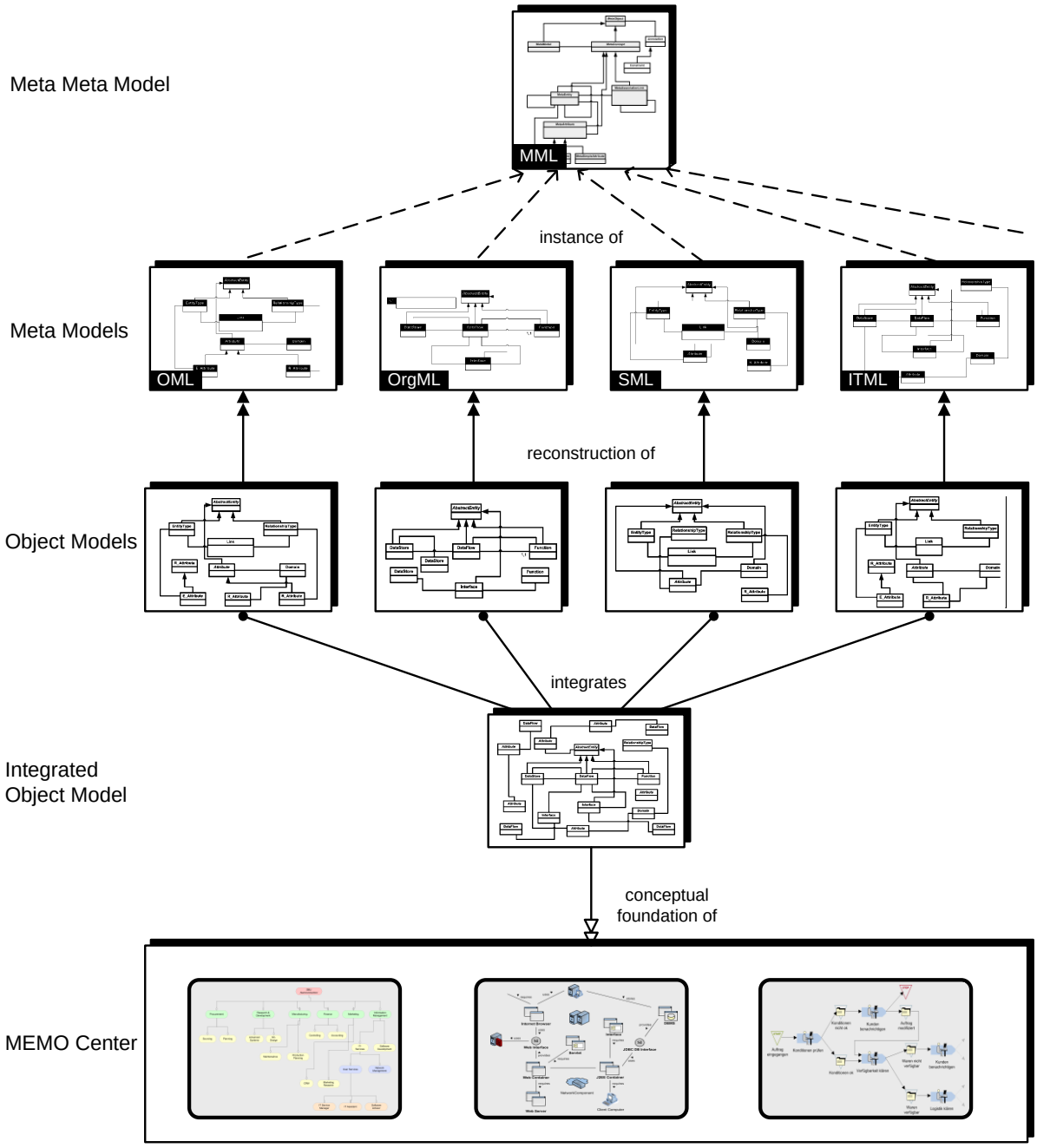


Figure 20: The MEMO language architecture and corresponding conceptual foundation for modelling tools

6 Outline of a Modelling Tool

The meta models specified through the MEMO MML can be used as a conceptual foundation for the development of modelling tools. This requires reconstructing them as object models (see chapter 5). With respect to the remarkable gain in productivity provided by the GMF, we decided to use it as a foundation for the development of MEMO Center. MEMO Center a is modelling environment that allows for creating various models, which are all integrated. For this reason, it provides cross-model integrity checks. If, for instance, a business process model includes a reference to an IT resource with an ITML model, the tool would prevent deleting this resource or would – on explicit user demand – perform a consistent delete operation in all related models. Furthermore, the tool allows for transforming models of various kinds into other representations. For example, a business process model that is integrated with an ITML model could be transformed into the schema of a workflow management system – for the description of a prototype, see [Jung04]. The set of MEMO modelling languages is supposed to be extensible, which implies the development of further model editors. For this reason, the creation and integration of new model editors as well as the maintenance of editors should be supported by an efficient tool. The tool – which is currently under construction – is built using the GMF. For this purpose, the meta meta model was reconstructed as an instance of Ecore.

Figure 21 shows a simplified version of the Ecore instance that was created with the GMF. Note that this model is represented as an instance of Ecore, while its presentation within the model editor gives the impression that it is a class diagram. However, its semantics is different from a class diagram. The connectors between two instances of *EClass* – such as *MetaEntity*, *MetaAttribute* etc. – do not represent associations as they are known from class diagrams. Instead, they represent references as they are used on the implementation level. Therefore, each association in the MEMO meta meta model is represented by two links in the Ecore instance. In addition to that, further peculiarities of Ecore have to be accounted for. For this reason, creating a meta (meta) model in the GMF is certainly more demanding (and confusing) than using a specialized editor – like the MML editor that is illustrated in Figure 22.

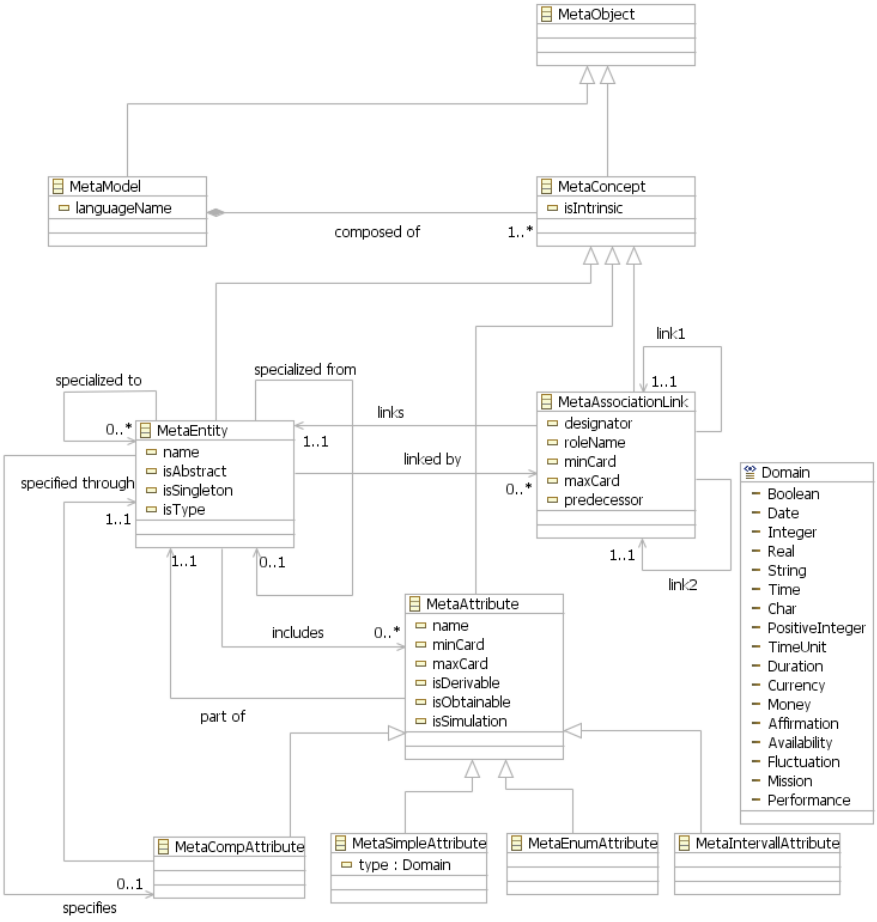


Figure 21: The MEMO meta meta model as an Ecore instance

The MEMO meta modelling editor allows for specifying MEMO meta models. As soon as a meta model is finalized, the editor transforms it into a corresponding Ecore instance. This includes the transformation of OCL statements. Subsequently, further specifications, such as the concrete syntax, have to be added. This still requires remarkable expertise and effort. Nevertheless, the MEMO meta modelling editor and the GMF, it is part of, facilitate the construction of additional model editors to a great extent. Figure 22 illustrates through a simplified workflow how to develop an editor for a new MEMO modelling language.

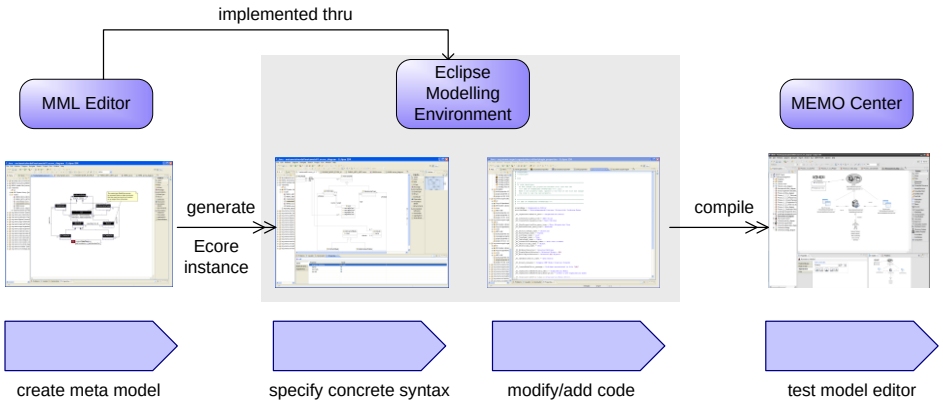


Figure 22: Simplified workflow for developing additional model editors within MEMO Center

7 Future Research

The new version of the MEMO MML reflects more than ten years of experience with designing languages for enterprise modelling. Hence, it is promising a relatively mature foundation for specifying meta models. Nevertheless, new requirements may evolve that suggest modifying the MML. Hence, we regard the MML as an instrument, but also as an ongoing subject of our research. This is the case with the language architecture, too. Focussing on new domains motivates the design of new modelling languages. The corresponding meta models are then added to the language architecture. In order to keep the architecture consistent, commonalities of the languages need to be analyzed from time to time. This may result in redesigning the language architecture by merging languages.

MEMO is a method for enterprise modelling. A modelling method does not only consist of one or more modelling languages, but also of one or more corresponding process models that guide the application of the languages. A process model is comprised of the control flow of phases that need to be completed. It also specifies the roles that are required for staffing a corresponding project. In order to support the individual configuration of process models, a specific language for designing process models can be applied. This can either be an adapted version of a business process modelling language or a dedicated language for modelling project phases, such as the one specified by Schauer as an extension of the MEMO language family ([Scha08], p. 245 f.). A meta modelling language like the MML and a language for modelling process models provide the foundation for designing methods that satisfy particular requirements. However, for many prospective users of a customized method designing it from scratch would be too much effort. Therefore, our future research on method engineering will target approaches to reuse and adapt existing modelling languages and process models.

A method that is specified through meta models for the language(s) and process model(s) it includes, provides an excellent conceptual foundation for elaborate project management tools. A process model – as an instance of a corresponding meta model – would represent a certain type of managing projects. Its phases would be related to role types, types of models and – as a prescriptive reference – to states of models that are supposed to be accomplished. A particular project would then be represented through representations of models and a corresponding instance of the selected process model. Such a representation could be used to generate the static structure of an information system that would manage all aspects of a project that were specified in the method, e.g. states (or versions) of models accomplished (or not) in any phase.

8 References

[AtKü07] Atkinson, C.; Kühne, T.: **Reducing accidental complexity in domain models**. In: Software and Systems Modeling. Online First, June 2007

[Baar03] Baar, T.: **The Definition of Transitive Closure with OCL – Limitations and Applications**. In: Broy, M.; Zamulin, A.V. (Eds.): Perspectives of System Informatics. Springer: Berlin, Heidelberg etc. 2003, p. 358-365

[Fill05] Fill, H.-G.: **UML Statechart Diagrams on the ADONIS Metamodeling Platform**, Proceedings of the International Workshop on Graph-Based Tools (GraBaTs 2004), Electronic Notes in Theoretical Computer Science, Vol.127, No. 1, 2005, pp. 27-36

[Fran98a] Frank, U.: **The MEMO Meta-Metamodel**. Research Report No. 9, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau 1998

[Fran98b] Frank, U.: **Evaluating Modelling Languages: Relevant Issues, Epistemological Challenges and a Preliminary Research Framework**. Research Report No. 15, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau 1998

[Fran98c] Frank, U.: **The Memo Object Modelling Language (MEMO-OML)**, Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 10, Koblenz 1998

[Fran98d] Frank, U.: **Applying the MEMO-OML: Guidelines and Examples**. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 11, Koblenz 1998

[Fran01] Frank, U.: **Organising the Corporation: Research Perspectives, Concepts and Diagrams**. Research Report No. 25, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau 2001

[Fran03] Frank, U.: **Ebenen der Abstraktion und ihre Abbildung auf konzeptionelle Modelle - oder: Anmerkungen zur Semantik von Spezialisierungs- und Instanzierungsbeziehungen**. In: EMISA Forum, Band 23, Nr. 2, 2003, pp. 14-18

[Fran10] Frank, U.: **Outline of a Method for Designing Domain-Specific Modelling Languages**. ICB-Research Report, Institut für Informatik und Wirtschaftsinformatik, Universität Duisburg-Essen, No. 42, 2010

[FrLa03] Frank, U.; Laak, B. van: **Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen**. Research Report No. 34, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau 2003

[GoSt90] Goldstein, R.C.; Storey, V.: **Some findings on the intuitiveness of entity-relationship constructs**. In: Lochovsky, F.H. (Ed.), Entity-Relationship Approach to Database Design and Querying. Elsevier Science: Amsterdam 1990, pp. 9-23

[Hitc95] Hitchman, S.: **Practitioner perceptions on the use of some semantic concepts in the entity-relationship model**. In European Journal of Information Systems, vol. 4, 1995, pp. 31-40

[Jung07] Jung, J.: **Entwurf einer Sprache für die Modellierung von Ressourcen im Kontext der Geschäftsprozessmodellierung**. Logos: Berlin

[Jung04] Jung, J.: **Mapping of Business Process Models to Workflow Schemata. An Example Using MEMO-OrgML and XPDL**. Arbeitsberichte des Instituts für Wirtschafts- und Verwaltungsinformatik, Universität Koblenz-Landau, Nr. 47, 2004

[JuKü+00] Junginger, S.; Kühn, H.; Strobl, R.; Karagiannis, D.: **Ein Geschäftsprozessmanagement-Werkzeug der nächsten Generation - ADONIS: Konzeption und Anwendungen**. In: Wirtschaftsinformatik, vol. 42, no. 5, 2000, pp. 392-401

[KeLy+96] Kelly, S., Lyytinen, K., Rossi, M.: **MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment**, in Proceedings of the 8th International Conference on Advanced Information Systems Engineering, CAiSE'96, Heraklion, Crete, Greece, May 1996, ed. by Constantopoulos et al., Lecture Notes in Computer Science No. 1080, Springer: Heidelberg 1996, pp. 1-21

[Kirc08] Kirchner, L.: **Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung**. Logos Verlag: Berlin 2008

[Lore96] Lorenz, K: Sprache. In: **Enzyklopädie Philosophie und Wissenschaftstheorie**. Ed. by J. Mittelstraß. Vol. 4, Metzler: Stuttgart, Weimar 1996, pp. 49-53

[MaPa+92] Mayer, R.J.; Painter, M.K.; deWitte, P.S.: **IDEF Family of Methods for Concurrent Engineering and Business Re-Engineering Applications**. Knowledge Based Systems: College Station 1992

[Odel98] Odell, J.: Power Types. In: Odell, J. (Ed.): **Advanced Object-Oriented Analysis and Design Using UML**. , Cambridge University Press: Cambridge 1998, pp. 23-33 (revised version of: Odell, J.: Power Types. In: Journal of Object-Oriented Programming, Vol. 7, No. 2, 1994, pp. 8-12

[OpHe99] Opdahl, A .L.; Henderson-Sellers, B.: **Evaluating and Improving OO Modelling Languages Using the BWW-Model**. In Proceedings of the Information Systems Foundations Workshop (Ontology, Semiotics and Practice), (digital publication), Sydney 1999

[OMG05] OMG: **Unified Modeling Language Specification**. Version 1.4.2, 2005

[OMG06a] OMG: **Meta Object Facility (MOF) Core Specification**. Version 2.0, 2006

[OMG06b] OMG: **Unified Modeling Language: Infrastructure**. Version 2.1.1, 2006

[OMG06c] OMG: **Object Constraint Language**. OMG Available Specification. Version 2.0, 2006

[OMG07] OMG: **OMG Unified Modeling Language (OMG UML)**, Superstructure. Version 2.1.2, 2007

[Scha08] Schauer, H.: **Unternehmensmodellierung für das Wissensmanagement. Eine multi-perspektivische Methode zur ganzheitlichen Analyse und Planung**. Dissertation, University Duisburg-Essen 2008

[SüEb97] Süttenbach, R.; Ebert, J.: **A Booch Metamodel**. Fachberichte Informatik, 5/97, Universität Koblenz-Landau 1997

[Stei00] Steimann, F.: **Formale Modellierung mit Rollen**. Habilitationsschrift. Universität Hannover, Hannover 2000

[Webe97] Weber, R.: **Ontological Foundations of Information Systems**. Coopers&Lybrand: Melbourne 1997

[Wied10] Wiedenbruch, A.: **A Method for Modeling Container-based Intermodal Transportation Networks**. Master Thesis, University Duisburg-Essen 2010

Previously published ICB - Research Reports

2010

No 42 (December)

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

No 41 (December)

Adelsberger,Heimo; Drechsler, Andreas (Eds): "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; Pietsch, Wolfram; Schmid, Klaus; Schneider, Kurt; Thurimella, Anil Kumar (Eds): "16th International Working Conference on Requirements Engineering: Foundation for Software Quality. Proceedings oft he 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 STCP"

No 34 (June 2009)

Karastoyanova, Dimka; Kazhamiakan, Raman; Metzger, Andreas; Pistore, Marco (Eds.): "Workshop on Service Monitoring, Adaption 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; Kütz, Martin; Rüding, 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 Müller-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 Geschäftsprozess-management-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. H. H. Adelsberger Information Systems for Production and Operations Management	E-Learning, Knowledge Management, Skill-Management, Simulation, Artificial Intelligence
Prof. Dr. P. Chamoni MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
Prof. Dr. F.-D. Dorloff Procurement, Logistics and Information Management	E-Business, E-Procurement, E-Government
Prof. Dr. K. Echtle 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. B. Müller-Clostermann Systems Modelling	Performance Evaluation of Computer and Communication Systems, Modelling and Simulation
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 Networking Technology	Computer Networking Technology
Prof. Dr. E. Rukzio Mobile Human Computer Interaction	Novel Interaction Technologies, Personal Projectors, Pervasive User Interfaces, Ubiquitous Computing
Prof. Dr. A. Schmidt Pervasive Computing	Pervasive Computing, Uniquitous Computing, Automotive User Interfaces, Novel Interaction Technologies, Context-Aware Computing
Prof. Dr. R. Unland Data Management Systems and Knowledge Representation	Data Management, Artificial Intelligence, Software Engineering, Internet Based Teaching
Prof. Dr. S. Zelewski Institute of Production and Industrial Information Management	Industrial Business Processes, Innovation Management, Information Management, Economic Analyses

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)