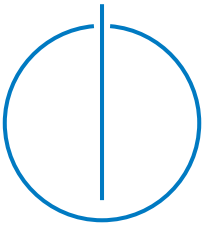
Fakultät für Informatik
Der Technischen Universität München

Bachelor's Thesis in Wirtschaftsinformatik

Specifying and classifying generic enterprise model repository services

Florian Balke



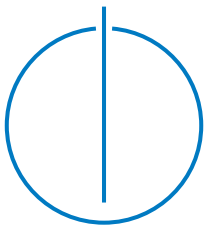
Fakultät für Informatik
Der Technischen Universität München

Bachelor's Thesis in Wirtschaftsinformatik

Specifying and classifying generic enterprise model repository services

Spezifikation und Klassifikation generischer Dienste eines Repositories für Unternehmensmodelle

Author:	Florian Balke
Supervisor:	Prof. Dr. rer. nat. Florian Matthes
Advisor:	Dipl.-Inf. Christian M. Schweda
Date of submission:	15.09.2010



I assure the single handed composition of this bachelor's thesis only supported by declared resources.

München, den 15. Oktober 2010

Florian Balke

Acknowledgments

Abstract

Enterprises increasingly resort to enterprise architecture (EA) management to cope with today's challenging environmental conditions. Thereby, EA management is gaining importance as a central means of supporting enterprises both to adapt to changing market conditions and to take new business opportunities enabled by innovative technologies. The benefit of EA management is directly coupled with documenting and analyzing the EA, so as to be able to provide a holistic view on the enterprise, describing the current state as well as future states thereof. Hence, devising an information model to structure the information of such descriptions constitutes one of the major tasks of EA management.

Although the plurality advocates the idea that EA information models are organization-specific design artifacts (cf. [BMS10b]), it is to assumed that an underlying meta-model exists, in which the different organization-specific information models are founded. Therefore, this thesis sets the goal of eliciting requirements for EA information modeling, while assessing whether state-of-the-art modeling concepts, such as those of the UML [OM10], are sufficient to satisfy the requirements. Subsequently, a set of tools is evaluated against these requirements to get a general idea of the capabilities of their generic repository services for EA information models.

Zusammenfassung

Unternehmen greifen vermehrt auf das Management der Unternehmensarchitektur (UA) zurück, um mit den herausfordernden Umweltbedingungen fertig zu werden. Dabei gewinnt das UA Management, als zentrales Mittel zur Unterstützung bei sowohl der Anpassung an sich ändernde Marktbedingungen, als auch dem Ergreifen neuer Geschäftsgelegenheiten durch innovative Technologien, zunehmend an Bedeutung. Der Nutzen des UA Managements ist direkt mit der Dokumentation und Analyse der UA verbunden, um in der Lage zu sein eine ganzheitliche Sicht auf das Unternehmen bereitzustellen, welche den aktuellen, sowie zukünftige Zustände davon beschreibt. Daher stellt die Gestaltung von Informationsmodellen zur Strukturierung dazu benötigter Informationen eine der Hauptaufgaben des UA Managements dar.

Obwohl die Mehrheit die Meinung vertritt, dass UA Informationsmodelle organisationsspezifische Design-Artefakte sind (cf. [BMS10b]), ist davon auszugehen, dass ein zugrundeliegendes Meta-Modell existiert, in dem die verschiedenen Informationsmodelle begründet sind. Deshalb, setzt sich diese Thesis zum Ziel, Anforderungen an die UA Informationsmodellierung zu erheben und darüber hinaus zu beurteilen, ob Modellierungskonzepte nach dem Stand der Technik, wie die aus der UML [OM10], genügen, um den Anforderungen gerecht zu werden. Darauffolgend wird einen Reihe von Tools evaluiert, um einen Überblick der Fähigkeiten ihrer generischen Repository Dienste für Unternehmensmodelle zu erhalten.

Contents

List of Acronyms

EA

EAM

EAMPC

IDE

IT

OCL

sebis

UML

XML

XML Schema

List of Figures

List of Tables

1 Introduction

Today's enterprises increasingly have to cope with challenging environmental conditions. On the one hand the enterprises have to adapt to a changing market environment, e.g. demand and buying behavior of customers or competition with other companies. On the other hand technological innovations, enabling new business opportunities for strengthening competitiveness, have to be taken into account. All of these factors inevitably call for an increased alignment of business and information technology (IT). At this point the management of the enterprise architecture (EA) comes into play, steadily gaining importance for companies (cf. [Bu08a, BMS10a, BMS10b]).

In accordance with Matthes et al. in [Ma09], enterprise architecture management (EAM) seeks to control and enhance the existing and planned IT support for an organization, doing this in a continuous and iterative process. EAM addresses the challenges of organizational change by providing a holistic view on the enterprise, considering IT and business-related aspects as well as environmental factors. The documentation of the EA plays a major role thereby, since the holistic description of the management subject is a prerequisite to any management endeavor. In order to build such a description, enterprises have to gather the needed information and to devise an *information model*[1], which defines and structures the necessary information. Since companies normally differ in their organization, in their perception of what EAM is, and in the goals they are pursuing thereby, procedures for gathering information as well as the underlying information models are regarded as organization-specific artifacts that have to be designed to fit the corresponding organizational environment.

The work presented in this thesis is based on the foundations of the EAM Pattern Catalog[2] (cf. [se10b]), which serves as source of investigation and as means of valida-

[1] Information models may also be referred to as meta-models (cf. [Ai08]) or conceptual models (cf. [Gu05]), but in order to avoid ambiguity and confusing, it is abstained from using other therms than information model in the remainder of the thesis

[2] Therefore, this work is based on the research on System Cartography at the chair for Software Engineering for Business Information Systems (sebis) of Prof. Dr. Florian Matthes at the Technische Universität München.

tion of the findings. The motivation and goals of the thesis are described in Section 1.1 followed by an overview of its structure in Section 1.2.

1.1 Motivation

As alluded to above, EAM is a topic that increasingly attracts the attention of today's enterprises, since it tackles the important challenge of aligning IT and business. Thereby, EAM is strongly connected to the documentation and analysis of the EA (cf. [Ai08, BMS10b]), so as to be able to provide a holistic view on the enterprise, for which reason the conceptualization of the enterprise makes up one of its major tasks. Albeit information models being organization-specific, it is to assume that different information models may be founded in a common meta-model or to put it into other terms may be documented by the same modeling concepts. Staying with this idea, requirements that have to be fulfilled by the modeling concepts provided for information modeling have to be elaborated. Object-oriented languages, such as the most notorious example, the UML [OM10], are used to provide different state-of-the-art information models (cf. [GWS04, Gu05, Bu07, se10b, BMS10a, BMS10b]). Due to the almost arbitrary complexity of information models, it is furthermore to investigate whether pure object-oriented approaches are appropriate or more sophisticated concepts of conceptualization are needed.

Due to the presumably convergence of required EA information modeling concepts, the thesis sets the goal of eliciting requirements for EA information modeling. The subsequent goal is to evaluate a set of tools providing *generic repository services* for enterprise models based on the beforehand elicited requirements.

1.2 Structure of the thesis

The conduction of the thesis started with a comprehensive analysis of the EAM Pattern Catalog [se10b], related literature of EAM as well as literature concerned with conceptual modeling based on well-founded ontologies. Theses scientific foundations for EA information modeling and the applied modeling concepts are briefly summarized in Chapter 2.

The elicitation of requirements for EA information modeling takes place in Chapter 3, which resulted in a set of representative scenarios reflecting requirements for EA information modeling rather single requirements. The scenarios are either directly derived from the EAM Pattern Catalog [se10b] or stem from related literature and

are as far as possible validated by experiences of the EAM Pattern Catalog [se10b]. For each scenario, a set of questions is devised that summarizes the requirements thereof and serves the purpose of a fingerpost for evaluating the scenario.

The scenarios elaborated in Chapter 3 compose the foundation for evaluating a set of tools providing repository services in Chapter 4. Each tool is evaluated against each scenario at the same criteria derived from the overall composition of the scenarios.

Subsequent to the evaluation of repository services, the findings of the thesis are critically reflected giving an outlook on research questions that have to be investigated more in-depth or appeared due to findings of the thesis.

2 Scientific foundations for EA information modeling

The EAM Pattern Catalog [se10b] serves as foundation for eliciting requirements for EA information modeling. Besides this knowledge base, overall experiences of the field of EAM in general and EAM information modeling in particular (cf. [Ma09]) exert influence on the conduction of this thesis. Furthermore, the perception of EA information modeling is driven by the notion of an ontological well-founded conceptualization of information models (cf. [Gu05]), since such a notion is regarded closer to the perception of a user without technical background, as presumably encountered while establishing an organization-specific EAM function. The scientific foundations resorted to over the thesis as well as global assumptions made therein are devised throughout this section.

Irrespective of the steadily increasing importance of EAM, there is no definition of EAM that achieved overall acceptance in research and practice. In the remainder of this thesis, the definition devised by Matthes et al. in [Ma09] is resorted to as follows:

> *Enterprise architecture management* is a continuous and iterative process controlling and improving the existing and planned IT support for an organization. The process not only considers the information technology (IT) of the enterprise, also business processes, business goals, strategies etc. are considered in order to build a holistic and integrated view on the enterprise.
>
> Goal is a common vision regarding the status quo of business and IT as well as of opportunities and problems arising from theses fields, used as a basis for a continually aligned steering of IT and business.

Out of this definition, Matthes et al. emphasise in [Ma09] the major role of EAM to plan and manage the evolution of the EA, targeting an enhanced alignment of IT and business. Thereby, alignment of IT and business means to adjust the IT support to the business strategy, but as well to incorporate emerging innovative IT for enabling new business strategies and goals.

2.1 EAM Pattern Catalog

The EAM Pattern Catalog as presented among others in [Bu08a, Ma09, se10b] is a collection of best practice EAM patterns, that can be utilized to introduce an EAM function tailored to the organization-specific problems and context. The EAM Pattern Catalog can be applied to a new EAM endeavor from scratch, as well as the enhancement of an already existing EAM function. The EAM Pattern Catalog provides guidance for systematically establishing an organization-specific EAM function and thus provides an holistic view on the enterprise. For that purpose, the collection of proven best-practice patterns is subdivided into the three types *methodology*, *viewpoint* and *information model patterns*, which all may be adapted to the organizational context and the specific problem. Figure 2.1 provides an overview of the different EAM patterns and their relationships.

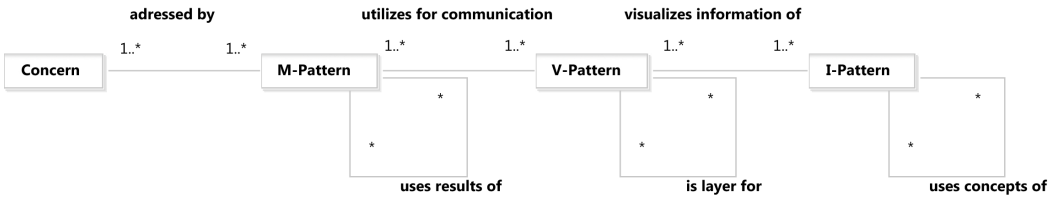


Figure 2.1: UML class diagram describing the relationship between Concerns (cf. [Bu08a])

- *Concerns* represent the pain points of a company, that have to be identified before starting an EAM endeavor.
- *Methodology Patterns* (M-Pattern) are selected subject to the beforehand identified concerns, helping to construct the set of necessary tasks to address the specific problems of a company.
- *Viewpoint Patterns* (V-Pattern) provide ways of visualizing information that support the chosen tasks of the M-Patterns.
- *Information model Patterns* (I-Pattern) provide the information needed by visualizations of V-Patterns. For that, an I-Pattern contains an information model fragment defining and structuring this information.

For the conduction of this thesis, particularly I-Patterns are put under investigation in order to derive requirements for EA information modeling and to validate elicited requirements by example.

2.2 EA management layers and cross functions

In accordance to Matthes et al. in [Ma09], there is a basic structure behind an EA information model giving an abstract overview of the general aspects that have to be taken into account. Matthes et al. refine in [Ma09] these general aspects into different architectural layers and cross functions capturing the elementary domains of an EA information model, as shown in Figure 2.2.

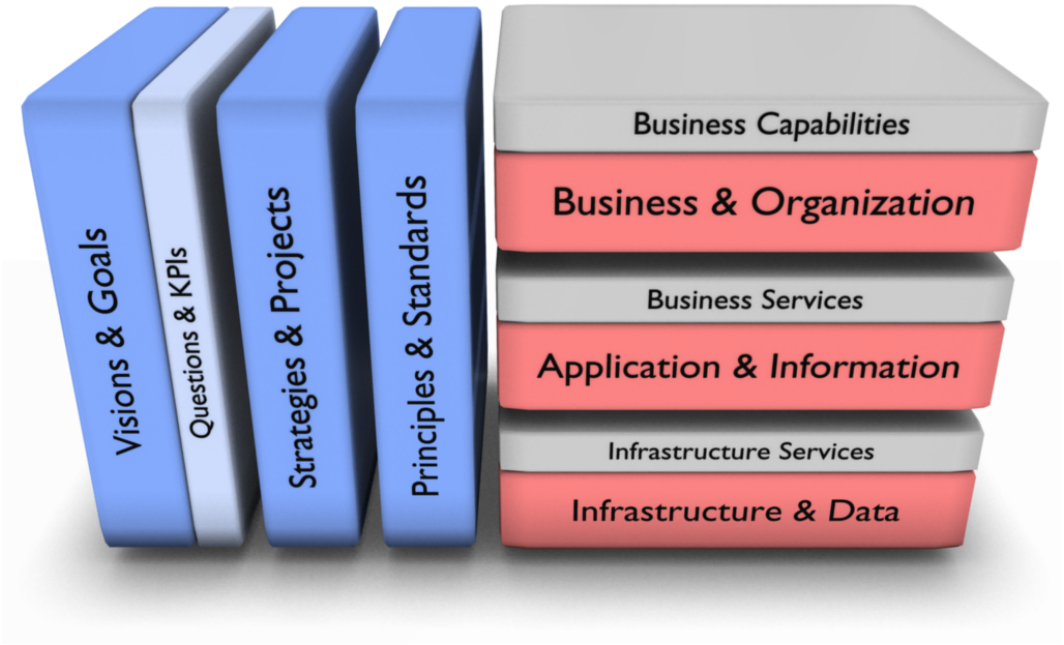


Figure 2.2: Layers and cross-cutting aspects of an EA (cf. [se10a])

The architectural layers of Figure 2.2 mirror the composition of a company's EA from business and organization related aspects, comprising logical concept that are independent of their technical realizaiton, over application and information related aspect, that describe the realization of the business related concepts, to aspects directly concerned with the IT infrastructure. The architectural layers are respectively extended by an abstraction layer, providing a customer-oriented perspective on the corresponding architecture layer, that thus suppresses details of the actual realization to focus on the provided functionality (cf. [se10a]). Cross functions are introduced to complement the layered structure with concepts, which do not exclusively belong to any of the layers and so are not directly part of the static EA, but may exert influence on any elements organized therein (cf. [Ma09]).

2.3 Ontological foundations for structural conceptual models

According to Guizzardi in [Gu07], the main success factor of using a modeling language lies in its ability to provide a set of primitive types that can directly express relevant domain concepts comprising what he calls a *domain conceptualization*. An abstraction of a certain state of affairs in reality articulated by elements constituting the domain conceptualization is in virtue of Guizzardi in [Gu07] a *domain abstraction*. Furthermore, he highlights the need of a language for representing domain conceptualizations and domain abstractions in a concise, complete and unambiguous way. Thereby, *domain appropriateness* is a measure of the truthfulness of a language to a given domain in reality and the *comprehensibility* states the conceptual clarity of the specifications produced in a language.

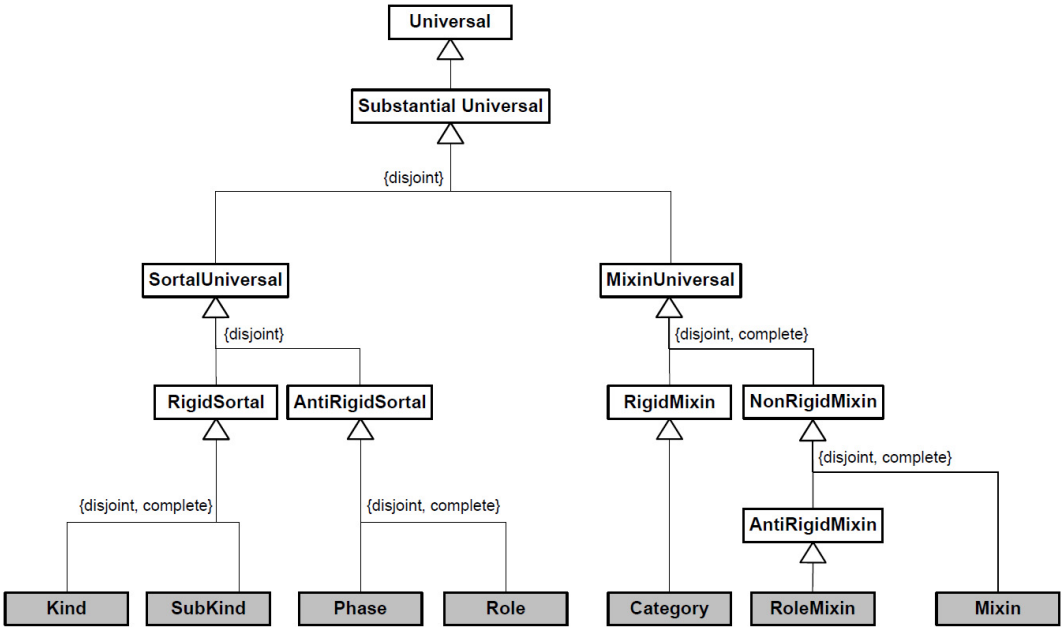


Figure 2.3: Typology of substantial universals, according to Guizzardi in [Gu05]

Guizzardi develops in [Gu05] a foundational ontology, named *Unified Foundational Ontology* describing a *typology of universals* that he uses in turn for re-designing the proportion of the UML [OM10] dealing with classifiers for the purpose of conceptual modeling and ontological representation. In order to achieve a concise, clear and unambiguous modeling throughout the scenarios in the succeeding sections, it is resorted to this foundational ontology for conceptual modeling. In [Gu05], the author introduces two ontological categories for *substantial universals*, namely *sortal* and *mixin universals*, as illustrated in the typology of substantial universals in Figure

2.3. Whereas a mixin universal only states whether a general term applies to a particular, a sortal universal additionally supplies a *principle of identity* enabling to decide upon the equality of two particulars. In terms of Guizzardi (cf. [GWS04, Gu07]) universals and particulars are roughly the ontological counterpart to classifiers and object instances in object-oriented modeling. He recommends a lightweight stereotype approach using the extension mechanism of UML to specialize modeling elements.

In line with Guizzardi in [GWS04, Gu07], sortal universals describe types that supply a principle of identity, whereas mixin universals constitute dispersive types that cover entities with different principles of identity. According to Guizzardi in [Gu05], there has to be always exactly one ultimate sortal inherited by each sortal universal, that supplies the principle of identity. Hence, if a sortal inherits from more than one other sortal, there has to be an ultimate sortal, which all of these inherited sortals may be traced back to. Slightly deviating from Guizzardi in [Gu05], rigid sortal universals are subsequently not annotated by an UML stereotype (cf. [OM10]), since kinds and subkinds may be compared to the native UML [OM10] concepts class and subclass. Thus, a kind of compatibility to pure UML information models is achieved, since kinds and subkinds are utilized, if not otherwise stated.

All types deviating from rigid sortal universals have to be annotated with a stereotype. Thereby, the utilization of stereotypes deviates again from Guizzardi's proposals and so the stereotype «mixin» denotes *categories* instead of *mixins* in conformance with Guizzardi's notion, since mixins in Guizzardi's notion are not utilized in the following and mixin is regarded as the more appropriate name for the most common mixin universal subtype. The further types of the typology of substantial universals utilized throughout the thesis are introduced later on by example in an appropriate context.

3 Scenarios for EA information modeling

The elicitation of requirements for EA information modeling primarily serves the purpose of covering the modeling issues from the point of view of the enterprise architect who takes the role of an information model designer or even the user. Hence, technical requirements as derivable for each modeling issue are not the focal point in the following, but come into play when evaluating by which technical concepts the repository services realize the elicited modeling requirements. Connected thereto is the request of reflecting the "true" ontological nature of the used types and their relationships in order to ensure clarity of the information model. The introduced concepts that are not provided by state-of-the-art object-oriented modeling languages by default pursue the goal of creating a domain appropriate and comprehensible information model.

In order to address the requirements, repository services are supposed to provide facilities for creating, maintaining and refining the information model, as well as for managing and storing corresponding information about the specific EA. Besides creating, updating and deleting of specific information, the management thereof includes providing access to repository information by queries or similar functionalities, as well. In doing so, the requirements elicited in the following have to be fulfilled by the repository services.

Instead of single requirements, a set of scenarios is devised in order to reflect the requirements for EA information modeling, that are based on current, related literature and particularly on the EAM Pattern Catalog (cf. [se10b]). The patterns of the EAM Pattern Catalog, constituting best-practices that have stood the test at various practitioners, make up the basis for deriving relevant scenarios. The selection of the analyzed patterns has taken place in an iterative investigation process, so as to incorporate all insights gained throughout the elicitation of requirements. Hence, on the one hand scenarios are directly derived from the EAM Pattern Catalog and on the other hand stem from further investigation in the field of EA information modeling, backed and validated by examples of the EAM Pattern Catalog, as far as possible. The devised scenarios are, where possible, complemented by illustrative information models, reflecting a typical situation, in which the requirements of a scenario apply.

Such typical situations are directly or indirectly derived from patterns of the EAM Pattern Catalog, so resorting to a well-founded basis of experiences therefor. The information models are realized utilizing the UML[3] [OM10] extended by a lightweight stereotype approach, as outlined in Section 2.3.

Scenario	Relevant I-Patterns
General architecture aspects	
Hierarchy modeling	I-12, I-18, I-30, I-84, I-85
Temporal and variant modeling	I-24, I-32, I-33, I-40, I-44, I-93
Non-rigid typing and concept of identity	I-26
Multi-level modeling	I-26, I-66, I-69
Cross-cutting aspects	
Lifecycle	I-26, I-44, I-89
Project affects	I-33, I-35, I-36, I-38, I-39, I-44, I-57, I-70, I-89, I-94
Standardization	I-6, I-23, I-41, I-67
Goals	I-59, I-86, I-87
Responsibilities	-
Service aspects	
Role-based access control	-
Queries	-
Information model changes	-

Table 3.1: Scenarios for EA information modeling

Throughout the requirements elicitation the scenarios in Table 3.1 were identified. These scenarios are roughly subdivided into *general architecture aspects*, *cross-cutting aspects* and *service aspects*. In doing so, general architecture aspects make up important aspects of EA information modeling that particularly highlight the demand for further concepts than those of the UML [OM10]. Cross-cutting aspects deal with specific issues that may influence other concepts on different layers of the EA (cf. Section 2.2). Service aspects constitute functionalities that may be regarded mandatory for repository tools.

The results of the analysis contribute to the *Building Blocks for Enterprise Architecture Management Solutions* (BEAMS) as devised in [se10a], since relevant aspects for

[3] The utilization of UML as meta-language does not mean that UML is the most appropriate one for EA information modeling. Nevertheless, UML is commonly used and understood, as well as a fairly convenient language for describing objects-oriented models of any kind (cf. [BMS10b]).

the creation of re-usable information model building blocks (IBB) are investigated and coherent building blocks are derived thereby.

3.1 General architecture aspects

Whether dealing with hierarchies, temporality or one of the other aspects within this section, general architecture aspects are needed to achieve a concise and clear description of the EA information model. These aspects address architecture concepts needed on the one hand to realize the basics of the EA in different subject areas, which would cause reinventing the wheel if not available. On the other hand, they are requisite for appropriately realizing cross-cutting aspects in the succeeding Section 3.2.

3.1.1 Hierarchy modeling

Hierarchical concepts are pervasive throughout organizational structures, e.g. structures of authority to decide, as well as the composition of business-relevant objects, such as organizational units, business processes, component-based applications and so forth. In the context of EA modeling, this means to deal with relationships of subordination and different kinds of part-whole relationships. Extending hierarchies to a concept of organizing and structuring elements, further relationships depending on the underlying structure, such as linear order consisting of predecessors and successors, come into play. Thereby, several challenges occur which are not covered by pure UML [OM10] necessitating the utilization of further concepts, such as constraints in the Object Constraint Language (OCL) [OM06b].

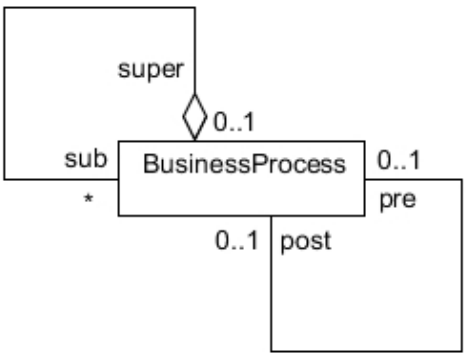


Figure 3.1: Object-oriented model of a hierarchy

Figure 3.1 shows an excerpt of the I-Pattern I-12 of the EAM Pattern Catalog [se10b] illustrating a typical hierarchical concept of a business process. This pure UML-based model does not ensure the subordinating part-whole relationship and its transitive closure to be acyclic. Furthermore, it is assumed that the ordering relationship is not independent of the hierarchical structure, since a business process of a lower hierarchy-level cannot be contained in the ordering of a higher hierarchy-level. Using OCL [OM06b] can solve abovementioned problems, but also impairs the readability of the model, as shown in Figure 3.2.

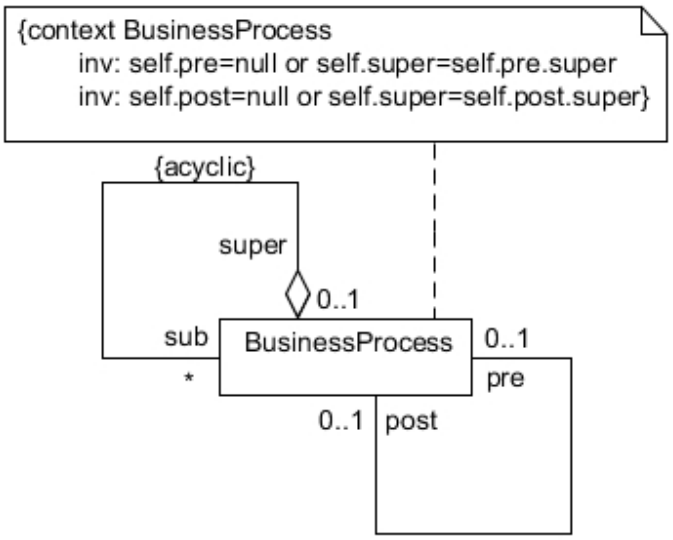


Figure 3.2: Hierarchy extended by constraints

The part-whole, respectively the parent-child relationship is annotated with a constraint `{acyclic}` in Figure 3.2 in order to ensure that no business process can be superordinate to its own superordinate business processes. This constraint can also be expressed using the OCL [OM06b], but would in this case require a very complex recursively defined condition, unnecessarily distracting from the example. Under the condition of an acyclic hierarchy, the second constraint, formulated in OCL, introduces two invariants to check whether the predecessor or successor process is either `null` or belongs to the same superordinate business process.

Albeit OCL [OM06b] is expressive enough for the little example, Guizzardi devises in [Gu06] a foundational ontology for conceptual modeling of part-whole relationships. He does not directly address problems of hierarchy modeling, but ones closely connected thereto. To support conceptual modelers, he defines a typology of four

different sorts of part-whole relationships and investigates which combinations of them are transitive. In contrast to specifying relationships by adding constraints in the OCL [OM06b], Guizzardi proposes an ontologically founded theory of different types of part-whole relationships considering inherent properties thereof, which can directly be utilized for creating the information model.

Several aspects may be relevant for hierarchies that are summarized in the following questions:

- How can (self-) relationships be annotated as being hierarchic, i.e. acyclic, one-to-many?
- How can sub-elements of a hierarchy be ordered, i.e. in a linear order?
- Can the validity of other relationships for lower or higher hierarchy levels be defined?

3.1.2 Temporal and variant modeling

Committed to the goal of documenting and analyzing both the currently implemented and the target EA, as well as planned intermediate evolutions, temporal aspects in the models are relevant. Staying with this idea, the evolution of the EA is supposed to be managed by the EAM function. Furthermore it is necessary to plan several states or variants for one point in future which may be decided upon later. That allows to support the decision making process with appropriate models and to reflect management decisions taken at an appropriate point in future, as well as a validation whether the plans pursue the strategic goals of the enterprise.

Besides the perspective of prospective states, the traceability backwards using historic data makes up another temporal aspect. This, for example, means that the evolution of plans over the time is made transparent and that the decision making may be reassessed by comparing documented alternatives and their rationales. This goal can be achieved by additionally modeling the time when a plan or a decision was made, respectively.

Conforming to Buckl et al. in [Bu09a] requirements for temporal modeling of the EA and its evolution may be deduced:

- Facilitating of target landscape planning by intermediate landscape plans in order to support the landscape evolution
- Traceability of management decisions by the documentation of historic data and rationales

- Support of different variants for a certain point in time
- Support of the life-cycle of architecture elements and their various relationships depending on the life-cycle phase

Especially the last point is not only concerned with temporal and variant modeling, but relates to additional modeling aspects, the non-rigidity of types and the life-cycle modeling of architecture elements, that exert influence on several other concepts. In line with Buckl et al. in [BMS10a] life-cycle modeling may be alluded to as cross-cutting aspect. These concepts are treated in a separate section due to their importance along with the management of EAs.

In order to model different states of an architecture element, it becomes necessary to model the period of time in which these states and versions are valid. To put it into other terms, properties and associations are supposed to be tracked in how they have changed or are expected to change by assigning a period of validity. For exemplifying the problem we regard the I-Pattern I-24 of the EAM Pattern Catalog [se10b] as shown in Figure 3.3, illustrating which business applications are hosted by which organizational unit.

Figure 3.3: I-Pattern I-24 – OrganizationalUnit-hosts-BusinessApplication

Analyzing the contained association for a certain point in time reveals that a specific business application is hosted by exactly one organizational unit. But extending the considered point in time to a period also affects the understanding of the hosts-relationship, since over the time it may be possible that a specific business application is hosted by different organizational units. In line with Carlson et al. in [CEF99], this problem may be addressed by the *temporal association pattern*. Thereby, the HOSTS-relationship is converted into the value class ApplicationHost augmented with two additional properties specifying the start- and end-time of validity, as depicted in Figure 3.4. This additional class is necessary, since organizational units are not only associated with a single business application and so a period of time stating their validity may not consistently reflect the validity a specific HOSTS-relationship. The problem of documenting passed through changes and envisioned ones, is solved thereby, but for the cost of introducing additional classes, which leads to further problems, such as lost clarity of the model. In particular, the restriction of the

multiplicities that a business application is only hosted by one organizational unit at the same time is instantly not ensured, necessitating the distinction between a long-term, i.e. change-aware, observation and a single point in time. This restriction may be ensured by adding a constraint, such as the one realized in the OCL [OM06b] in Figure 3.4.

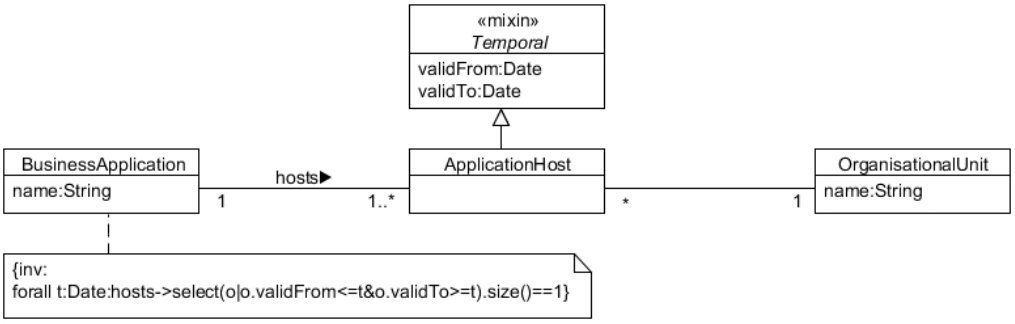


Figure 3.4: I-Pattern I-24 extended by the temporal association pattern

To address the loss of clarity throughout temporal modeling, Buckl et al. in [Bu09a] recommend in line with Carlson et al. in [CEF99] the utilization of an UML stereotype (cf. [OM10]) `«temporal»` for realizing the temporal property pattern that generally speaking expresses the same model as in Figure 3.4, even considering restrictions of multiplicities. The stereotype adds a period of validity to the HOSTS-relationship that has not to be explicitly stated by attributes and retains multiplicity dependent restrictions without introducing additional classes or constraints to the information model. The utilization of the stereotype combines the advantages of the original and the extended information model fragment. Thus, the clarity of the original model fragment is preserved and the expressiveness is extended by temporality, in doing so the stereotype `«temporal»` enables to model the true ontological nature of the hosts-relationship.

Figure 3.5: Temporal association structure applied on I-Pattern I-24 [se10b]

In Figure 3.5, the temporal association pattern stereotype structure is applied on the I-Pattern I-24 (cf. Figure 3.3) achieving the tracking of historic states and prospective ones, but traceability of changes asking for bitemporality and rationales is not ensured yet. Bitemporality calls for the capability of statements about states

and prospective changes of the EA at a point of view in the past, e.g. the current state is supposed to be compared to the assumptions on this present state of two weeks ago and these of a month ago. Therefore, a second dimension of time has to be considered, additionally modeling the time when certain plans were valid or have changed, respectively.

The diagram in Figure 3.6 exemplifies the three dimensions to deal with for temporal, bitemporal and variant modeling and thereby clears the distinctions between them. Issues of temporal modeling, concerning changes of the EA over the time, are ascribed to the x-axis of the diagram, realized in the examples by properties describing the period of validity, namely VALIDFROM and VALIDTO. The y-axis makes up the second dimension needed for extending to bitemporality, showing the time at which the valid historic, current or planned EA is supposed to be regarded. This dimension is realized by simply adding a property defining the point in time, at which the model fragment was instantiated, in the succeeding model in Figure 3.7. Finally, the z-axis of the diagram depicts the different variants that may exist at a certain point in time, which is not addressed by the model fragments in Figure 3.7 and Figure 3.8 yet.

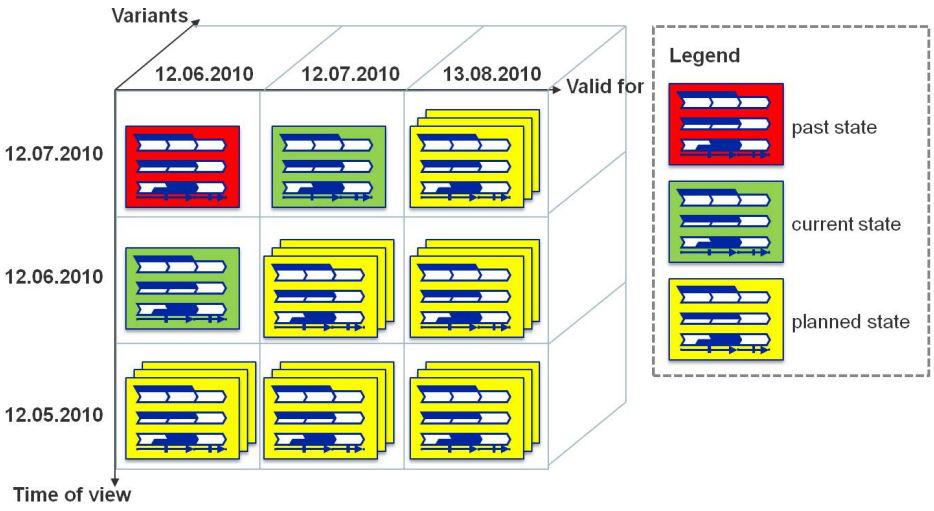


Figure 3.6: Schematic illustration of bitemporal and variant modeling of the EA

The model fragment in Figure 3.7 extends ORGANIZATIONALUNIT-HOSTS-BUSINESSAPPLICATION to bitemporality. That is realized by refining the mixin TEMPORAL to the mixin BITEMPORAL by adding an additional attribute defining the time of instantiation. Thus, the second dimension of time is introduced to the model fragment of Figure 3.4, but again with the important drawback of losing clarity due to the complex model structure. Since the resulting structure can be regarded as a

recurring model fragment again, realizing bitemporality independently of a specific application, an adapted UML stereotype (cf. [OM10]) `«bitemporal»` is proposed, as shown in Figure 3.8. In order to complement the traceability by respective rationales, the mixin BITEMPORAL might be related to a type RATIONALE, documenting such reasons.

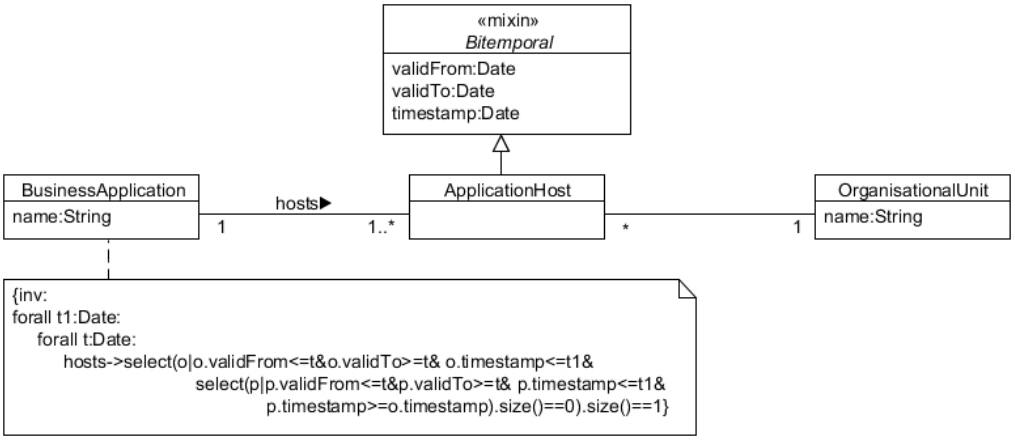


Figure 3.7: Bitemporal modeling

Albeit bitemporal modeling enables the traceability of changes, the modeling of different versions is not enabled yet, since irrespectively of the traceability there cannot be multiple plans for a certain point in future. As projects are the major means of EA evolution, the existence of distinct project portfolios for a certain point in time may realize variant modeling, which is addressed by the cross-cutting aspect project in Section 3.2.2.

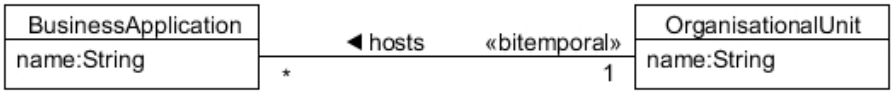


Figure 3.8: Bitemporal modeling by UML stereotype

Moreover, the life-cycle of a business application has to be taken into account, enforcing the consideration of retired business applications. Currently, the retirement of a business application cannot be expressed by the devised model without an object-oriented workaround, e.g. a subtype of ORGANISATIONALUNIT used only to state that a business application is not hosted. Alternatively, the multiplicity for organizational units might be softened for deriving the retirement by the non-existence of a valid host-relationship after a certain point in time. Both mentioned ways are

suboptimal solutions neglecting the true nature of the relationship or the participating types, causing further problems. Hence, the concise modeling of the true nature of lifecycles considering changing relationships of a type is inevitable, as addressed in-depth in Section 3.2.1.

Requirements of temporality in information modeling may be addressed by the following questions:

- How can modeling elements be defined as time-dependent, i.e. having a period of validity, or as time-independent?
- What modeling elements can be defined as time-dependent, i.e. properties or relationships?
- Which means of introducing bitemporality are provided?
- How can modeling elements be denoted as an intermediate or final state thereof?
- How can different variants of an element be specified?

3.1.3 Non-rigid typing and concept of identity

The EA is subject to a process of continuous change and enhancement (cf. [Bu07, Ma09]). Broken down to single elements of an EA, this implies the change of relationships between elements, of properties of elements or even the entire type of architecture elements, in order to consistently model the change process. Thereby, the distinction between ontological types supplying a principle of identity and not is a prerequisite of coherently modeling non-rigid types, since a type change has not to imply changing the identity of concerned elements. Non-rigid means that types defined in the information model can be made changeable or at least extensible, that is to say additionally to assign new properties and references to a type, even the nature of an entity's instance, embodying its type is supposed to be changeable.

Throughout the conceptual modeling of EAs, instances of different classifiers having the same identity in an ontological sense may be dealt with. For instance, this issue occurs while modeling the different steps that have to be passed through when transforming a demand into a project proposal, in turn into an executable project and finally in a completed project, retained for documentation purposes. During the entire transition process the demand, the project proposal, the executable project and the completed project can be traced back to an embracing project without changing the identity. This may be proven by a simple example, in which an already finished project is regarded. Whereas during the execution different stages are explicitly

named and sometimes cannot directly be ascribed to its actual identity. Afterwards, the effects of a project are ascribed to a project as a whole, irrespective of the specific phase in which the effect has taken place and whether this phases contains the word "project" in its name, exemplifying that the different phases bear the same principle of identity.

Similar issues may be met with by employee management. Assuming that primarily every member of a company is an employee, it is necessary to define more precisely the actual position within the organization. For example, an employee can be a member of a specific department, but also of one or more projects not linked to this department. The same employee is concerned, whether the department member or project member role is regarded, but in each role with specific relationships to other business entities.

As outlined in Section 2.3, Guizzardi et al. devises in [GWS04] an ontologically well-founded theory for conceptual modeling to address the abovementioned problems of individuation and identity supply. Whereas mixin universals only state whether a general term applies to a particular, sortal universals additionally supply a principle of identity enabling to decide upon the equality of two particulars. Figure 2.3 exemplifies the usage of ontological types supplying a principle of identity and not. In contrast to the introduction in Section 3.9, sortal universals are annotated by `«sortal»` throughout this section of non-rigid types for emphasizing the distinction to other ontological types.

Employee and BusinessApplication in Figure 3.9 are both sortal universals inheriting from the mixin universal Element. In an organizational context an employee and a business application have some common properties, i.e. a name, but different principles of identity. Owing to this fact, Element cannot be the identity supplying classifier and thus, it describes a mixin universal subsuming common properties. Moreover, Employee is the ultimate sortal and thereby the only identity supplying sortal inherited by MaleEmployee and FemaleEmployee, otherwise individuation might not be ascertained.

Continuing the aforementioned example of different stages a project passes through, it may be assumed that demands are primarily realized by a textual description. Subsequently, it might be required to add further or more structured information, as well as to define relationships, such as assignments of responsible or involved people during the further proceeding. Moreover, a demand may evolve to a concrete project proposal, for which already documented information is not supposed to be maintained a second time. After the project proposal is approved, the project is

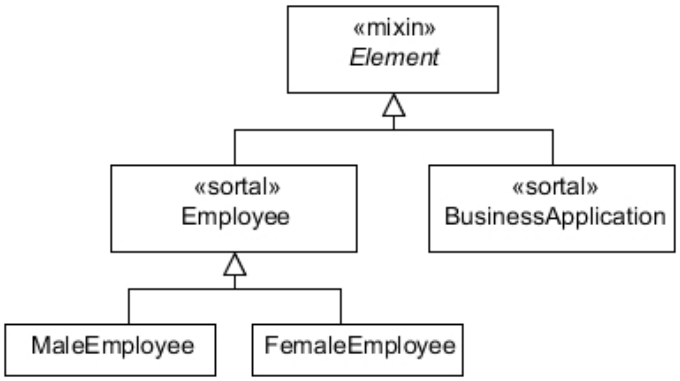


Figure 3.9: Sortals and Mixins

introduced and executed, until its completion. These explained stages of a project might be regarded as its life-cycle, but at least as disjoint states that it consecutively belongs to.

Guizzardi et al. addresses in [GWS04, Gu05] the problem by introducing a phased-sortal that represents a non-rigid type describing a part of a partition of a sortal in which all phased-sortals are mutually exclusive. In this sense, there cannot be a phased-sortal without a supertyping sortal, which vice versa is subtyped into phased-sortals constituting a complete, disjoint specialization set. The different non-rigid subtypes are annotated with the stereotype `«phase»`. Buckl et al. devise in [BMS10a] the modeling building block Lifecycled resorting to the ontological type phased-sortal, which is applied to the abovementioned example of the evolution of a project in Figure 3.10.

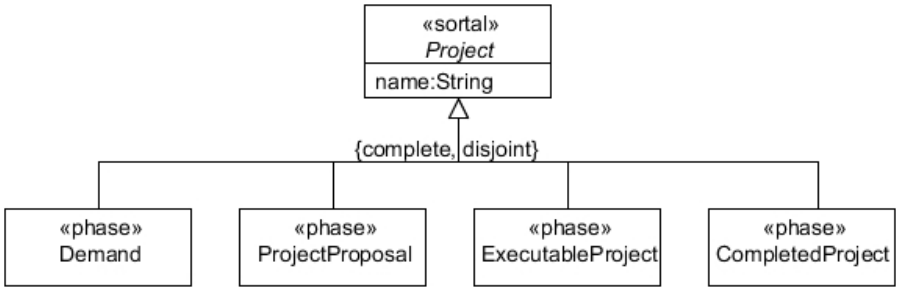


Figure 3.10: Modeling building block Lifecycled applied on Project

Using phased-sortals does not allow to model roles, as described at the example of employees, even since it is closely related to the modeling of phases. Roles also em-

body a non-rigid type, but several roles can be hold simultaneously by an object. Guizzardi et al. proposes in [GWS04, Gu05] another specialization of sortals, namely roles denoted by the stereotype «role». Figure 3.11 extends the employee hierarchy of Figure 3.9 by specializing EMPLOYEE to the role PROJECTMEMBER that relates an EMPLOYEE to a commitment in a PROJECT as reflected in the relationship MEMBEROF. This is also an important distinction between phased-sortals and roles, since a role reflects an external dependency that can be manifested by a relationship, such as the membership in the example in Figure 3.11, but also by certain properties of a type.

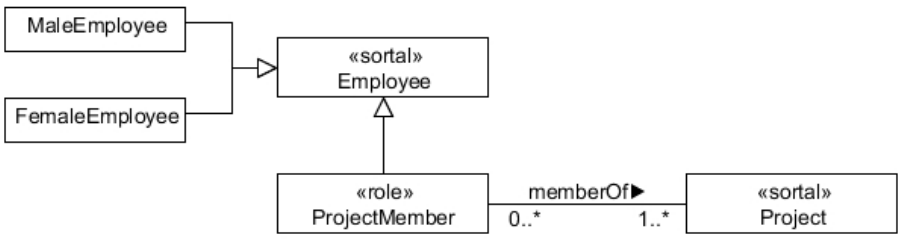


Figure 3.11: PROJECTMEMBERSHIP as a role of an employee

The following questions subsume the requirements that have to be fulfilled for non-rigid typing and the principle of identity:

- How can non-rigid types be specified?
- Which non-rigid types can be specified?
- How can the principle of identity be modeled?
- How can be distinguished between types supplying a principle of identity or not?

3.1.4 Multi-level modeling

EA information models are enterprise specific design artifacts (cf. [BMS10b]) and describe the domain of the specific EA of an enterprise. According to Atkinson and Kühne in [AK07], the inherent classification levels of a specific solution-independent domain are often mismatched by the available levels of the used modeling mechanism, since many of them are based on UML [OM10] that is rooted in a two level paradigm. According to Atkinson and Kühne many approaches trying to address this level mismatch are based on workarounds folding multiple domain classification levels into one modeling layer, i.e. the *item-description pattern* (cf. [AK07]) or also known

as *type-object pattern* (cf. [YJ02]). An approach enabling the modeling in multiple levels could avoid unnecessary complexity caused by such workarounds.

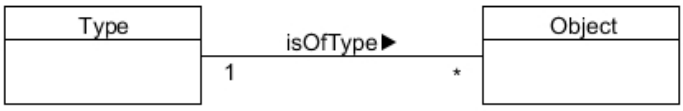


Figure 3.12: Type-object pattern (cf. [YJ02])

According to Yoder and Johnson in [YJ02], object-oriented design normally uses a separate class, in the meaning of the UML [OM10], for each type of object requiring changes of the information model when introducing new types. Therefore, Yoder and Johnson propose in [YJ02] not to model each type as class, rather by descriptions that have to be interpreted at run-time. Staying with this idea, Yoder and Johnson elaborate in [YJ02] the type-object pattern, in order to define subtypes of an entity and corresponding objects at run-time, as depicted in Figure 3.12.

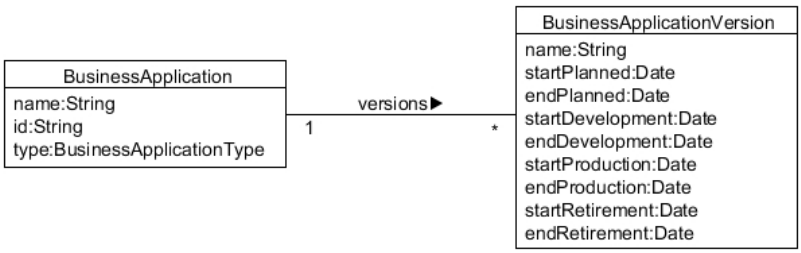


Figure 3.13: I-Pattern I-26 of the EAM Pattern Catalog [se10b]

In accordance with Yoder and Johnson in [YJ02], subtypes are simple instances of Type defining the description for specific entities. Object in turn is instantiated to the actual objects conforming to an instance of Type. Thereby the type-object pattern folds two ontological levels into one modeling level. To retain the specificity of the type-object pattern, but to avoid unnecessary complexity and to convey the true ontological nature of types in the information model, a multi-level modeling approach, as presented in the following may be utilized.

The modeling of versions as shown in I-Pattern I-26 in Figure 3.13 can be regarded as an application of the type-object-pattern, exemplifying this level mismatch by modeling the versions of a business application as associated type. Assuming that a specific version of a business application is always meant when dealing with the IT support for business functions or processes, respectively, in the EA, a version is

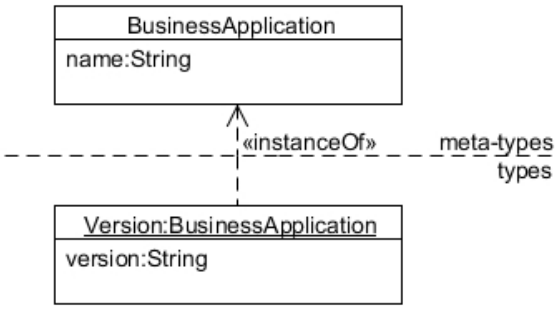


Figure 3.14: Multi-level modeling applied on business applications and their versions

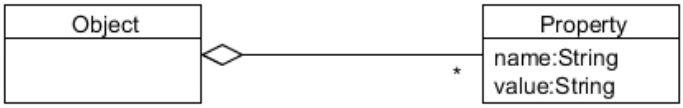


Figure 3.15: Property pattern (cf. [YJ02])

nothing else than an ontological instance of a business application and constitutes a specific release thereof. In this sense, the element BUSINESSAPPLICATION resides on an ontologically higher level than the elements that use the ontological instances or versions thereof, respectively. Applying multi-level modeling on the I-Patterns of the EAM Pattern Catalog [se10b] withdraws the need to deal with the type of business applications and the type of their versions within a single ontological level, which entails a reduction of complexity by a uniform use. Figure 3.14 resolves the mentioned level-mismatch of I-Pattern I-26.

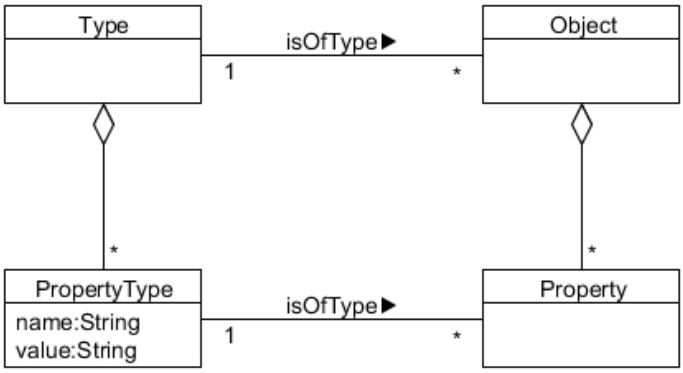


Figure 3.16: Type-Square pattern (cf. [YJ02]

According to Yoder and Johnson in [YJ02], the type-object pattern is often used in combination with the *property pattern*, that is used to enable varying attributes for instances of the same type, as illustrated in Figure 3.15. A combination of these patterns constitutes the *type-square pattern* (cf. [YJ02]) for which the type-object pattern is applied twice on ENTITY and PROPERTY of the property pattern, as shown in Figure 3.16.

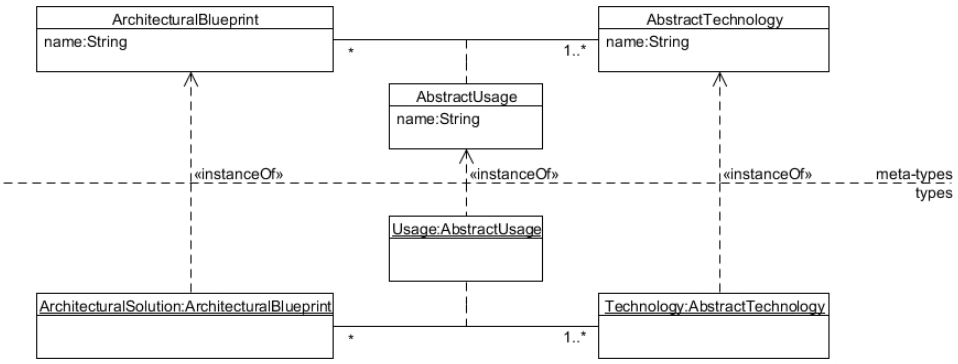


Figure 3.17: Mutli-level modeling applied on I-Pattern I-66 in [se10b]

I-Pattern I-66 in [se10b] is a prominent example for the application of the type-square pattern. The I-Pattern I-66 describes which abstract technologies are used by an architectural blueprint, which is in turn specialized to architectural solutions using concrete technologies. In accordance with the EAM Pattern Catalog [se10b], an architectural blueprint stands for a software architecture, i.e. three-tier- or pipe-and-filter-architecture, and an abstract technology is a class of technologies offering similar, or even standardized functionalities, e.g. web server or database management system (DBMS). Thereby, an architectural solution concretizes an architectural blueprint by selecting concrete technologies for each abstract technology that has to be specified and thus describes a basic architecture for a business application. In this context, a concrete technology represents a technical constituent of a business application or architectural solution, respectively, specifying abstract technologies, e.g. "Oracle 9i" is a specification for "DBMS". In an ontological sense, architectural solutions and their associated technologies are instantiations of architectural blueprints and their used abstract technologies, as depicted in Figure 3.17.

The following question has to be answered to analyze multi-level modeling capabilities:

- How can multiple ontological levels of a domain be modeled?

3.2 Cross-cutting aspects

In accordance with Buckl et al. in [BMS10a], cross-cutting aspects make up a couple of concepts, which are not selectively assigned to a single aspect of EA management (cf. Section 2.2), rather may influence various other concepts of EA information modeling. Due to their influence throughout the EA, they are paid notable attention to. In [BMS10a], Buckl et al. identify five cross-cutting aspects, namely *projects*, *life-cycle*, *standards* and *goals*, as well as *responsibilities*.

3.2.1 Lifecycle

Life-cycles are already alluded to in Section 3.2.2 as a challenge of coherent modeling non-rigid types and the principle of identity, but the life-cycle also constitutes a cross-cutting aspect of EA information modeling. Almost all architecture elements may have a life-cycle that can reach from their introduction or development over an operational period to their retirement. Illustrating that using a business application, a business application may be in the life-cycle phases *in planning*, *in development*, *operational*, and *replaced*, as described by the I-Pattern I-26 of the EAM Pattern Catalog [se10b]. In each phase a particular business application may have distinct relationships and qualities but is still the same business application. To address this problem, the non-rigid type phased-sortal is resorted to (cf. Section 3.1.3) and the modeling building block LIFECYCLED (cf. [BMS10a]) is applied to BUSINESSAPPLICATION, as depicted in Figure 3.18.

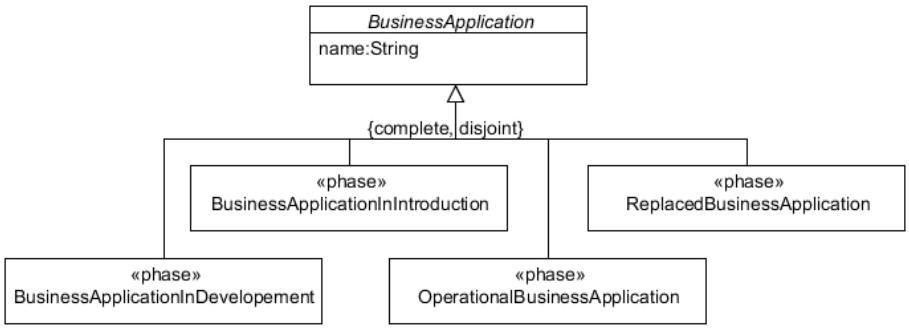


Figure 3.18: Modeling building block LIFECYCLED applied on business application (cf. [BMS10a])

According to Buckl et al. in [BMS10a], each instance of a lifecycled type is supposed to retain information of the time of transition between two phases. As well, an ordering of the life-cycle phases or constraints, which defines possible transitions

between life-cycle phases, is conceivable, but neglected in the aforementioned example in Figure 3.18. The temporal information of transition may be added by modeling the period of validity of a certain phase, but an ontologically meaningful modeling of transition constraints is a more intricate task. As subsequently elaborated in Section 3.2.2, projects are a central means of affecting architecture elements and therefore control the transition between two life-cycle phases. Therefore, Buckl et al. decompose in [BMS10a, Bu09a] projects into tasks or work packages, respectively, that transform architecture elements. Subsequently, Buckl et al. compose in [BMS10a] the modeling building blocks Project-affects-Affectable (cf. Figure 3.21 in Section 3.2.2) and Lifecycled (cf. Figure 3.18) to the modeling building block Project-Lifecycle-Affectable. Figure 3.19 applies the modeling building block Project-Lifecycle-Affectable on BusinessApplication for exemplifying the usage.

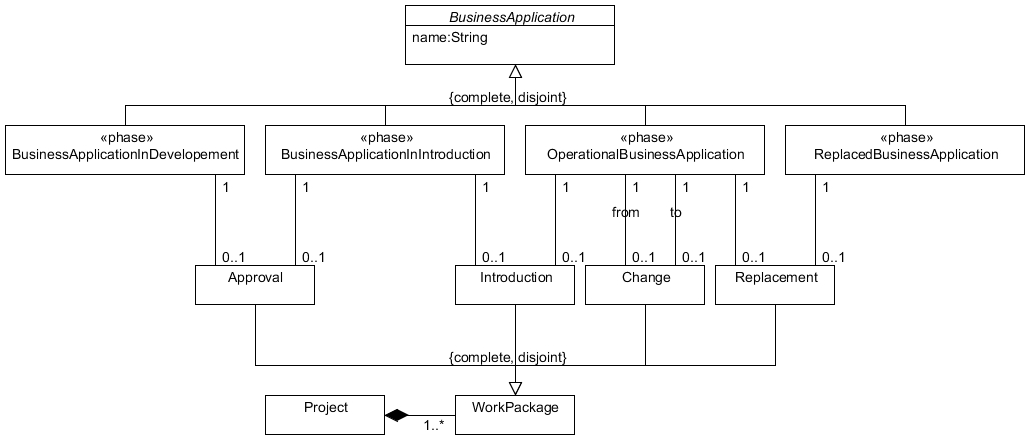


Figure 3.19: Modeling building block Project-Lifecycle-Affectable applied on a Lifecycled BusinessApplication

The transition between states of an EA element is conducted by different types of work packages that make up a kind of reification of the affects-relationship of the modeling building block Project-affects-Affectable (cf. Figure 3.21). The subtypes of work package are elaborated in conformance with the phases of a business application. Thereby it is assumed, that a business application in "planning phase" is transformed into the "in introduction phase" by an approval of the plans. The transitions between the other phases can be regarded analogously, except for the change of an operational business application, that is supposed to be an evolution of the operational business application without replacing it, that otherwise might be achieved by the combination of introduction and retirement. Thus, an ordering and control of transitions between different states of EA elements is implicitly achieved.

The addressed issues of this section can be evaluated by the following questions:

- How can the life-cycle of an element be defined?
- How can the life-cycle stage of an element be changed?
- Can the time of transition be documented?
- Can the life-cycle phases be ordered or their transition restricted?
- How are relationships and properties adapted to the life-cycle stage?

3.2.2 Projects

Projects are important for steering the evolution of the EA. Projects result from received demands or pursued goals and hence, the execution of a project can always be ascribed to a particular rationale in order to achieve a particular goal. Goals and their achievement form another cross-cutting aspect, dealt with in one of the subsequent sections. The cross-cutting aspect project, as depicted in following, primarily describes the effects on other architecture elements. Effects may be the introduction of new architectural elements, as well as changes and replacements thereof or even the retirement of elements. In doing so, distinct project portfolios affecting the EA, play the major role of defining different variants of the future EA. In this context, projects enriched with temporal information are the means to achieve the requirements of temporal and variant modeling of Section 3.1.2 that remained open yet.

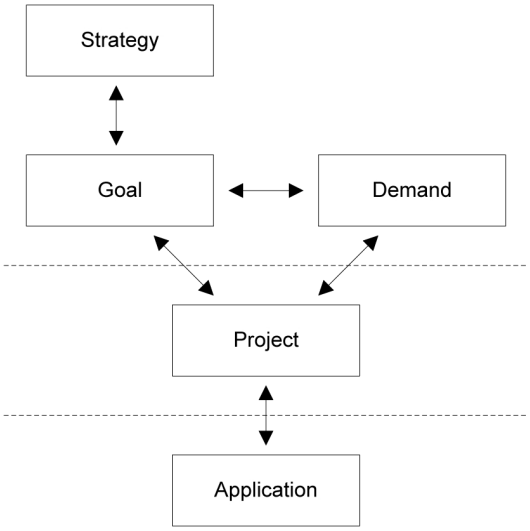


Figure 3.20: Exemplified impact analysis, connecting applications via project with their introducing goals and demands and vice versa

Considering the diverse effects caused by a multitude of demands or pursued goals, it is obvious that taking dependencies among projects into account is not a negligible challenge. Projects pursuing similar goals and the temporal progression of the projects have to be synchronized, as well as mutually exclusive projects have to be avoided. In order to achieve this, projects have to be comparable in their effects, pursued goals and their temporal dimension, irrespective of starting the comparison at the affected elements or vice versa at the pursued goals. The project's central role for the EA evolution is illustrated in Figure 3.20, in which PROJECT is the connector between GOAL and DEMAND, introducing an effect, and APPLICATION, as example of an affected element.

The multitude of occurrences of the type PROJECT in the EAM Pattern Catalog [se10b] confirms the fact of being a cross-cutting aspect. PROJECT is contained in several patterns affecting other architectural elements, such as business applications (cf. I-Patterns I-33, I-35, I-36, I-39, I-57, I-89), technologies (cf. I-Pattern I-38) or services (cf. I-Patterns I-44, I-70), thus validating the commonality of effecting architectural elements.

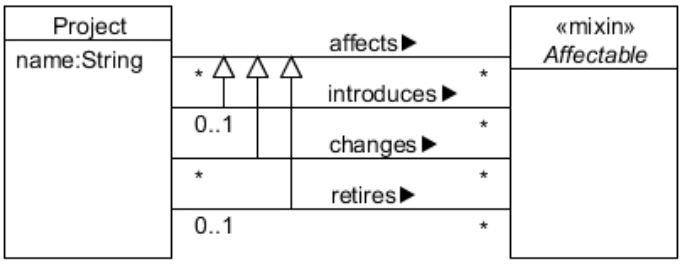


Figure 3.21: Modeling building block PROJECT-INTRODUCES-CHANGES-RETIRES-AFFECTABLE (cf. [BMS10a])

Buckl et al. propose in [BMS10a] to generalize the affected element types to the mixin AFFECTABLE in order to address the specificity of projects, capable of affecting a plurality of EA elements. Furthermore, Buckl et al. refine the modeling building block PROJECT-AFFECTS-AFFECTABLE to express that actually specializations of AFFECTS, namely INTRODUCES, CHANGES, RETIRES, are utilized, having different multiplicities, as well. In order to pay the deserved attention to this relationship Buckl et al. recommend resorting to the distinction of *formal* and *material relationships* elaborated by Guizzardi in [Gu06]. According to Guizzardi in [Gu06], formal relationships are directly hold between entities without any further intervening particular and are reducible to intrinsic qualities of the entities. Comparison relationships are a prominent example for formal relationships. In contrast, material

relationships are induced by a mediating entity called *relator*. Since almost all relationships dealt with throughout the scenarios are material ones, it is abstained from annotating material relationships, rather only formal relationships are explicitly denoted. The affects-relationship and its specializations are all material relationships and can be reified by their corresponding relator types. Buckl et al. propose in [BMS10a] a relator hierarchy consisting of the superclass Effect and the three specializations Introduction, Change and Retirement. The mentioned extensions are shown in Figure 3.21 and Figure 3.22.

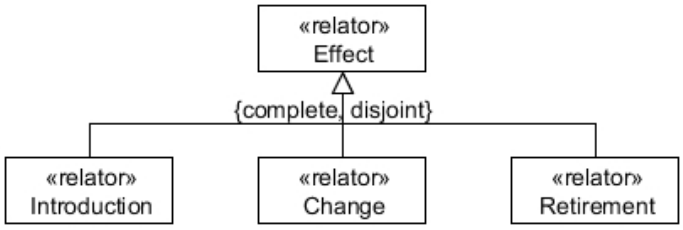


Figure 3.22: Relator hierarchy Effect, Introduction, Change and Retirement (cf. [BMS10a])

The already repeatedly mentioned requirement of variant modeling constitutes a specific challenge in EA information modeling, since a plenty of aspects have to be taken into account in order to achieve an ontologically coherent modeling thereof. This goal can be achieved by composing the elicited aspects of bitemporal modeling, life-cycle modeling and the transition among life-cycle phases, as well as projects as means of EA evolution. The example OrganizationalUnit-hosts-BusinessApplication in Figure 3.3 is supposed to be complemented with aspects elaborated in the corresponding sections.

The following requirements subsume the aspects that have to be fulfilled:

- Incorporation of the business application's life-cycle (cf. Figure 3.18)
- Effects on business applications introduced by projects (cf. Figure 3.19 and Figure 3.21)
- Introducing of temporality to projects
- Composition of the different aspects to achieve variant modeling

Initially, the temporal dimension of projects is introduced by simply adding a period of validity stating the time a project is going to be conducted or has been conducted,

respectively. Even bitemporal aspects would be possible but not mandatory to address the subsequently handled variant modeling. Much more important therefore is to determine project dependencies, particularly those among projects affecting the same elements of the EA simultaneously, as if to deduce different variants of the EA. Figure 3.23 shows the simple temporal extension of Project that is in turn conveyed to the different phases of Project, as described in Figure 3.10.

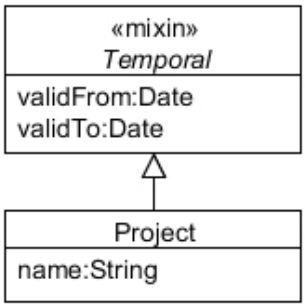


Figure 3.23: Temporal project

To avoid overloading of the information model while resorting to the needed concepts, a concise modeling is supposed to be enabled by introducing UML stereotypes (cf. [OM10]). The stereotype `«projectAffectable»` is introduced to emphasize in this section that a type annotated thereby participates in an AFFECTS-relationship with PROJECT. Additionally, the stereotype `«lifecycled»` expresses that a thus annotated type is subtyped in different life-cycle phases that have to be further specified. Figure 3.24 illustrates the application of these stereotypes, in doing so the life-cycle phases in Figure 3.18 are assumed.

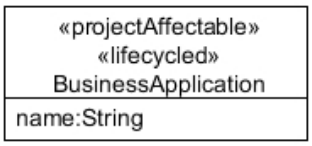


Figure 3.24: projectAffectable» and «lifecycled» applied to BUSINESSAPPLICATION

The extensions made in Figure 3.24 enable the modeling of different versions of a business application depending on the affecting projects. The incorporation of a business applications life-cycle influences the HOSTS-relationship in Figure 3.3, since only a business application in an operational state can be hosted by an organizational unit. Hence, not the supertype BUSINESSAPPLICATION participates in the HOSTS-

relationship, rather the subtype OperationalBusinessApplication, as depicted in Figure 3.25. Thereby, the retirement of a business application no longer asks for a workaround for the hosts-relationship, instead simply a phase transition of the concerned business application provokes that the host-relationship ceases to exist.

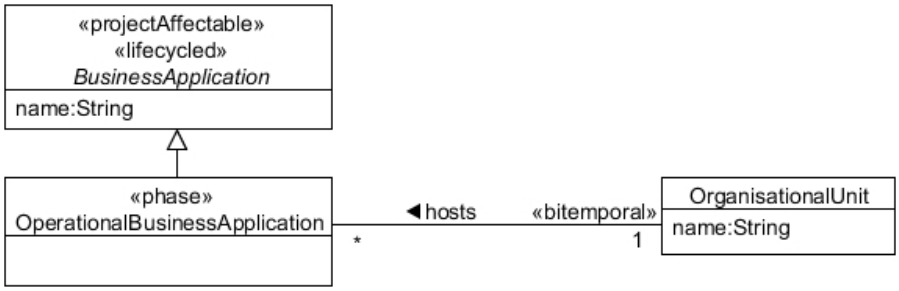


Figure 3.25: Modeling building block OrganistionalUnit-hosts-Operational-BusinessApplication

Albeit variant modeling resorts to multiple other concepts for being adequately realized, it is fairly simply achieved after having prepared the required concepts. By annotating their utilization, a sufficient expressive and concise modeling of variants is achieved. Thus, the clarity of the original model fragment of I-Pattern I-24 is preserved and the true ontological nature of the utilized types is pointed up.

Summarizing this section, the following questions are relevant, when dealing with projects:

- How can elements be defined as "project-affected"?
- Which effects can be distinguished for the effect-relationship?
- How can the start and end time of the different project phases be defined?
- How does the existence of a period of validity affect the EA?

3.2.3 Standardization

Many companies have to cope with an EA that is the result of an unguided evolution over a long period of time (cf. [Bu08a]). That might mean to deal with a heterogeneous EA using a high number of different technologies constituting an EA far away from being suited for the organization-specific context and problems. Consequences might be high maintenance, license or operating costs, lower flexibility or an inadequate business support.

Standardization as cross-cutting aspect of EAM addresses these issues by documenting the existing standards and solutions, as well as the conformance of architecture elements, such as business applications, while incorporating the underlying rationales of their existence. Thereby, the awareness of nonconformance is increased and deviations from standards are supplied with descriptions whether they are necessary, targeting on the consolidation of utilized technologies to get closer to an EA as well adapted to the organization-specific needs as possible. An important case thereby is the distinction between standard software and individual software. Whereas the former has to be only adapted to the company's specific needs, the latter is individually developed for the using company, including all side effects, such as maintenance or further development, which is otherwise provided by the vendor of the standard software. Thus, it has to be well evaluated, for which piece of software it is necessary to be developed individually and which one can also be obtained from a software vendor and subsequently configured for the company's needs.

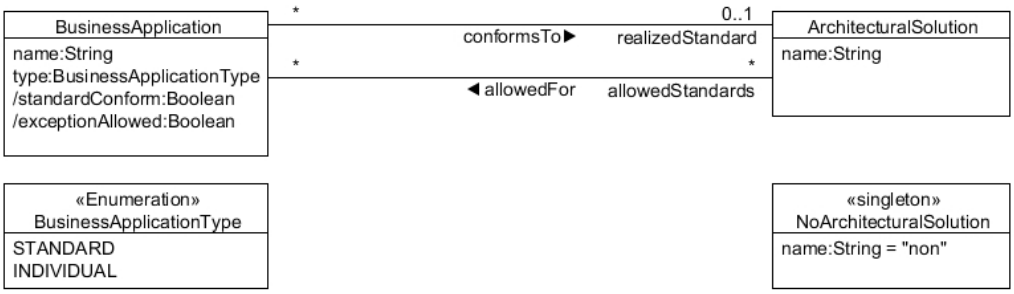


Figure 3.26: I-Pattern I-67 (cf. [se10b]) – Architectural Solution Conformance

The EAM Pattern Catalog [se10b] provides several I-Patterns concerning the cross-cutting aspect standardization. I-Pattern I-6 documents the conformance to architectural solutions by business applications. I-Pattern I-41 introduces a property to each business application for the distinctions whether it is standard or individual software. I-Pattern I-67 (cf. Figure 3.26) introduces two relationships between architectural solution and business application, the one pointing on the architectural solutions allowed for realizing the business application and the other stating the conformed architectural solution, which has to be contained in the set of allowed architectural solutions. For the case that no standard architectural solution is deliberately used, ArchitecturalSolution is subclassed to NonArchitecturalSolution. Furthermore two attributes are derived, namely standardConform and exceptionAllowed. StandardConform indicates whether the business application

conforms to one of the documented architectural solutions and EXCEPTIONALLOWED whether a conformance is mandatory.

For incorporating the true nature of standards, namely the capability of an element to conform to a standard, Buckl et al. propose the mixin STANDARDIZABLE stating this capability to conform to a standard. As well, STANDARD is realized by a mixin, since the true nature of a standard is regarded as its quality to denote existing architecture elements as a standard. Furthermore, this perception goes in line with the notion of STANDARDIZABLE addressing the diversity of elements that are able to conform to a standard. The building block STANDARD-STANDARDIZABLE depicted in Figure 3.27 slightly deviates from the proposal of Buckl et al. in [BMS10a].

Figure 3.27: Building block STANDARD-STANDARDIZABLE

According to Buckl et al. in [BMS10a] both derived attributes of the mixin STANDARDIZABLE can be regarded as a vehicle of backwards-compatibility, since building blocks modeling the conformance by a simple binary property can be consistently exchanged by the building block in Figure 3.27. Rather than simply introducing the STANDARD-STANDARDIZABLE modeling building block into I-Pattern I-67, the thought of achieving an ontologically coherent information model fragment was carried through, causing several other enhancements in both the model fragment and the modeling building block. Thus, the former workarounds as described in the following are resolved starting with those utilizable in general for standardization issues, resulting in a more sophisticated version of the modeling building block STANDARD-STANDARDIZABLE.

The specialization of an architectural solution in Figure 3.26 to a type only stating deliberate non-conformance is understood as a general standardization aspect and hence to incorporate in the modeling building block STANDARD-STANDARDIZABLE. Owed to the fact of introducing technical issues into the information model, an element similar to NOARCHITECTURALSOLUTION is omitted. Regarding the underlying notion, the artificial type NOARCHITECTURALSOLUTION serves the purpose of specifying two different kinds of standardization, namely those deliberately non-conforming and those supposed to conform. The latter type incorporates the case of a lacking documentation for elements that are nevertheless supposed to conform

to a standard. Hence, there are two states that an element, which is able to conform to a standard, may reside in, but between which it can arbitrarily swap in line with the changing requirements of the EA, and is in both of the states perceived as "standardizable". The two specializations of a "standardizable" do not directly state whether such an element conforms to a standard. They only express, whether an element is supposed to be conformant or not and the actual conformance has to be derived from its relationships to other elements.

For realizing that, a dispersive type that is changeable at runtime or a *non-rigid mixin-type*, respectively, is required. Thereby, the notion of a non-rigid mixin-type deviates from Guizzardi's in [Gu05], since Guizzardi conceives a non-rigid mixin type as a dispersive type utilized to subsume common properties of multiple non-rigid sortal universals that change in line with the underlying sortal universal, but cannot change independently thereof. A role-mixin, for example, has a non-rigid character, since the role, by which it applies to a sortal universal, can be changed in accordance with the specific context and therefore a role-change of a sortal universal also causes a change of the assigned role-mixin. Thereby, it may happen that a change between two roles take place, in which both concerned roles are subsumed by the same role-mixin, so that both prior to and after the role-change the general term of the role-mixin applies to the underlying sortal universal. However, a role-change without changing the general term applying by the role-mixin has to be regarded accidental, since this is not a requisite of a role-change, but actually for the mentioned non-rigid mixin type required for standardization issues. For standardization issues, the claim is to primarily express the capability of conforming to a standard, which in turn reside in two specific states, namely to be assumed to conform to a standard and to deliberately non-conform to a standard. Transferring this back to a role-mixin would mean that a sortal universal has compulsorily assigned a specific role-mixin, which in turn causes the acting in exactly one of a finite set of roles, between which the underlying sortal universal can swap arbitrarily. Consequently, the role-mixin would no longer be a non-rigid type, since it is rigidly assigned to the sortal universal, only being represented by one of a finite set of specializations. Returning again to standardization, exactly such a rigid general term is supposed to be assigned to a sortal universal, that can non-rigidly be specialized to the actual representation of its state.

In the following, a *non-rigid mixin*[4] is a dispersive type, mandatorily subtyping a mixin, in doing so a complete, disjoint generalization set has to be defined, in exactly one of these specializations the corresponding mixin has to reside in. Hereby, the meanwhile specialization of the mixin depends on the external context of the mixin. Up to a certain point, non-rigid mixins can be regarded as that for a mixin, which a role is for a sortal universal, since both depend on an external context and can be changed arbitrarily, but in contrast, non-rigid mixins constitute a partition of mutually exclusive types, of which exactly one has to be valid and a common general term permanently has to apply to the underlying type.

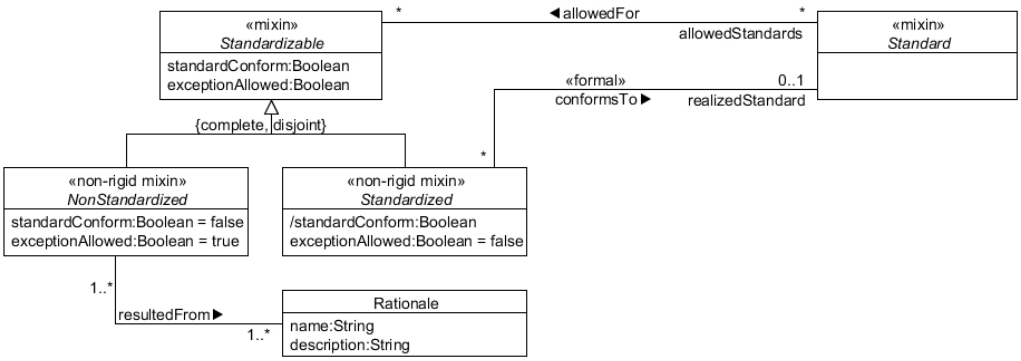


Figure 3.28: Modeling building block STANDARDIZABLE-NONSTANDARDIZED-STANDARDIZED-STANDARD

Transferring this to the context of standardization, the already introduced mixin STANDARDIZABLE is specialized into a complete, disjoint partition of non-rigid mixin-types, subtyping the mixin STANDARDIZABLE. These non-rigid mixin-types are denoted by the stereotype `«non-rigid mixin»`, stating that the mixin STANDARDIZABLE has to be specialized by exactly one of its subtypes at any time and may swap arbitrarily between them according to the evolving organization. Furthermore, the CONFORMSTO-relationship is shifted to the non-rigid mixin STANDARDIZED, whereas the non-rigid mixin NONSTANDARDIZED establishes a relationship to RATIONALE in order to document the justification for deviations. For addressing the deficiency of documentation, the attribute STANDARDCONFORM is derived, being directly interrelated to the CONFORMSTO-relationship. For coherently documenting and planning the situation of standardization of the EA, additional time attributes may be introduced to define the validity of subtypes of STANDARDIZABLE, which is neglected

[4] The short description of non-rigid mixins does not live up a well-founded typology as the one of Guizzardi in [Gu05], necessitating a more in-depth investigation to coherently specify the place of non-rigid mixins within an entire typology of types. Nevertheless, a type that is not directly covered by Guizzardi's UFO is required, embodied by non-rigid mixins.

for reasons of lucidity. The adapted modeling building block StandardizableNonStandardizedStandardizedStandard is depicted in Figure 3.28.

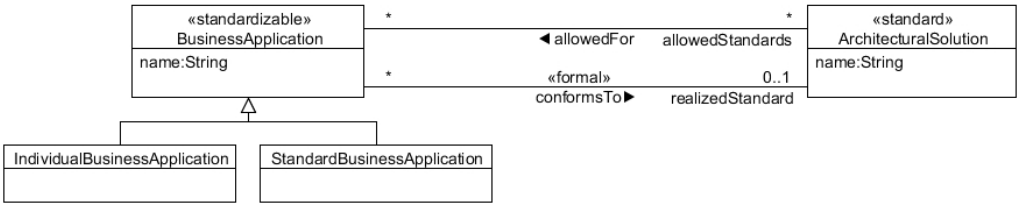


Figure 3.29: Modeling building block StandardizableNonStandardizedStandardizedStandard applied on I-Pattern I-67 [se10b]

Continuing the adaptations on example-specific aspects, leads to a more in-depth investigation of BusinessApplication. The characterization of a business application as standard or individual one in the I-Pattern I-67 by an attribute is a possible technical realization for simple one-level hierarchies, for which reason it is replaced by two subtypes of BusinessApplication. For reasons of lucidity, the resulting inheritance relationship between the mixin Standardizable and BusinessApplication of the I-Pattern I-67 is denoted with the stereotype `«standardizable»`, as shown in Figure 3.29. This stereotype equally expresses that BusinessApplication inherits the mixin Standardizable and contributes to making the information model more concise, since rather modeling a plenty of inheritance relationships, only the stereotype `«standardizable»` has to be assigned to appropriate types. Similarly, the usage of the mixin Standard is facilitated by the stereotype `«standard»`. All enhancements are applied on I-Pattern I-67 in Figure 3.29.

Figure 3.30: Modeling building block BusinessApplication-uses-Technology (cf. [se10a])

Nevertheless, the original notion of Standard-Standardizable, which is to denote architecture elements which are prospectively able to conform to a standard is preserved, but refined into subtypes that the mixin Standardizable resides in, reflecting its true ontological nature. One aspect was silently introduced to the modeling building block StandardizableNonStandardizedStandardizedStandard in Figure 3.28, needing a more detailed consideration, namely the annotation of the conformsTo-relationship by the stereotype `«formal»`. This stereotype resorts to

Guizzardi's distinction between formal and material relationships (cf. [Gu05, Gu06]) as described in Section 3.2.2. Staying with the example of a business application, on the one hand the actual purpose of a business application is presumably not to conform to a standard, but to support a certain business function or process, for which in turn a couple of technologies are required, which is shown in the modeling building block BusinessApplication-uses-Techology in Figure 3.30. On the other hand a set of used technologies constitute an architectural solution as described in I-Pattern I-23 in [se10b] that may act as a standard for business applications. Thus, the nature of a business application may be regarded as its usage of technologies to fulfill its support purpose while conforming to a standard that encompasses the "naturally" used technologies. Consequently, the conformsTo-relationship in the example is implicitly derived from the uses-relationship in Figure 3.30, using the architectural solution composed by these used technologies, validating the intrinsic character of the conformsTo-relationship for business applications. The incorporation of Techology and its corresponding relationships to the model fragment of Figure 3.29 is depicted in Figure 3.31.

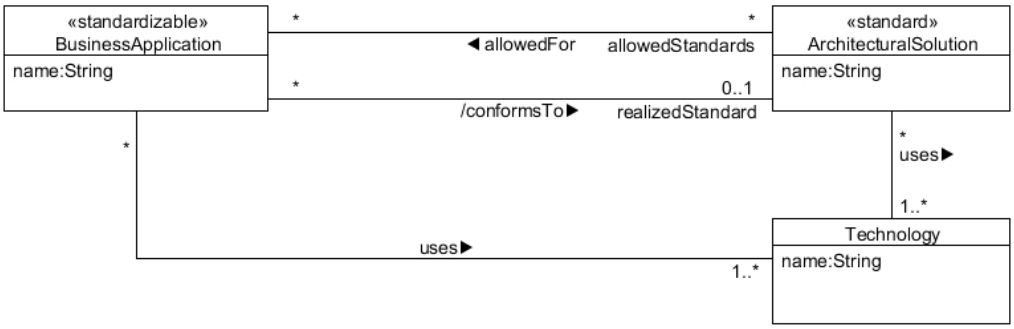


Figure 3.31: Business application standardization extended by uses

Transferring the thoughts of the preceding paragraph to the general notion of standardization reveals that standardization of architecture elements depends on their specific context, they belong to, e.g. their utilization or composition of other elements, which make up potential standards. In this sense, there is always an intrinsic moment of elements that are able to conform to a standard, from which the conformance to a standard can be derived, for which reason the conformsTo-relationship generally makes up a formal relationship, as depicted in Figure 3.28.

Questions arising when setting up a standardization strategy and adjusting them:

- How can architectural solutions be defined as standard?

- How can architecture elements be annotated of being able to conform to a standard?
- How can be distinguished between deliberately non-conforming and accidentally non-conforming elements?
- How can the rationales of non-conformance be established?
- How can a standard be changed that an element conforms to?

3.2.4 Goals

The alignment of the EAM activities to the organization's strategies and goals makes up another cross-cutting aspect. According to Matthes et al. in [Ma09], goals are the finer grained, more detailed decompositions of strategies which are typically the only discipline that is cultivated at enterprise level and thus exerts influence on almost the entire EA. Goals entail demands and projects that in turn affect other elements to achieve the underlying goals, as similarly illustrated above in Figure 3.20. For supporting a consistent alignment of the EAM activities to the goals, a backwards traceability starting at the level of affected elements via the affecting projects up to the goals is necessary, too. This approach seems to be quite similar to the cross-cutting aspect projects, but even though a GOAL-AFFECTS-AFFECTABLE building block allows traversing the dependencies between goals and affected elements, according to Buckl et al. in [BMS10a], it would neglect another important aspect of the nature of goals, namely the measuring of their achievement. Thus, Buckl et al. in [BMS10a] propose an approach in line with the *Goal-Question-Metric-approach* of Basili et al. in [BCR94].

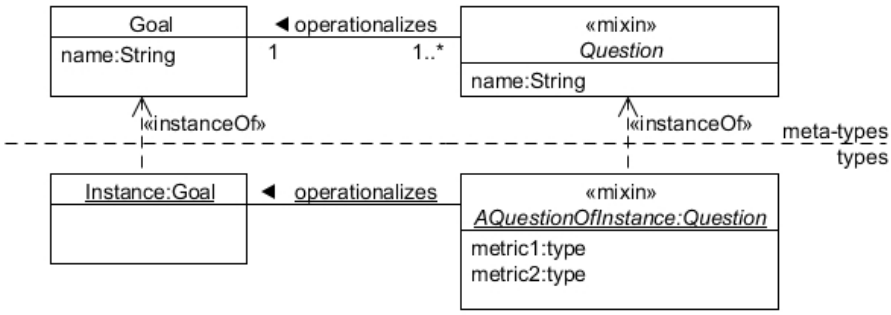


Figure 3.32: Modeling building block GOAL-QUESTION-METRIC (cf. [BMS10a])

Basili et al. in [BCR94] subdivide their approach of measuring the achievement of goals into three levels. The goals whose fulfillment has to be measured, reside on

the highest level. These goals are refined into several questions, which break down the goals into their major components. Each question is in turn associated with a set of metrics in order to achieve a quantitative measuring of goals. Buckl et al. in [BMS10a] resort to the Goal-Question-Metric-approach by introducing a mixin QUESTION aggregating metrics and indicators in order to operationalize goals. But measuring the achievement of a goal is not enabled by simply assigning a mixin QUESTION, since different metrics are needed for measuring a specific question of a specific goal or instance of a goal, respectively. Hence, attributes appropriate to concretize a specific question and to measure the achievement of the associated goal have to be assigned to the question on the same ontological level, on which the specific questions is located. Buckl et al. in [BMS10a] mention in this context the "twofold nature" of goals and questions, leading to a multi-level ontological instantiation approach as depicted by the building block GOAL-QUESTION-METRIC in Figure 3.32.

As shown in Figure 3.32 and outlined in the preceding paragraph, the measurable metrics are located at a lower ontological level as GOAL and QUESTION. In turn, instances of GOAL and QUESTION or specific goals and concrete questions, respectively, reside on the same ontological level as elements reflecting non-cross-cutting architecture elements that may incorporate concrete questions. In the following the GOAL-QUESTION-METRIC building block is applied to the topic of protection requirements introduced in I-Pattern I-86 of the EAM Pattern Catalog [se10b].

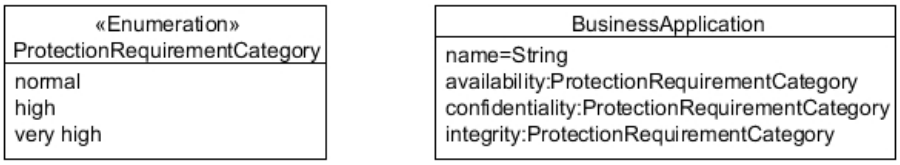


Figure 3.33: I-Pattern I-86 of the EAM Pattern Catalog [se10b]

According to the *Bundesamt für Sicherheit in der Informationstechnik* in [Bu08b], information is highly valuable to companies and government offices, and needs to be appropriately protected. There are a couple of reasons, why companies often spare no effort to protect their information. Maybe the most obvious one is that companies strive to secure their sensitive and confidential data from competitors in order to defend and strengthen their market position. Moreover and not less important is to ensure the availability and integrity of information. These three major qualities that have to be assured, namely confidentiality, integrity and availability, are also described by the Bundesamt für Sicherheit in der Informationstechnik in [Bu08b].

Additionally, the Bundesamt für Sicherheit in der Informationstechnik derives three qualitative statements to categorize these protection requirements, since the quantification is usually not accomplishable. In this vein, it proposes normal, high and very high as requirement categories. Since the assessment of protection requirements is a laborious process, a process model is elaborated in [Bu08b]. Subject to [Bu08b], damage that could occur, if the three qualities are not completely assured, can be subsumed in a couple of general damage scenarios. In order to select the appropriate category for a protection requirement, these scenarios have to be evaluated.

The I-Pattern I-86 of the EAM Pattern Catalog [se10b] (cf. Figure 3.33) deals with protection requirements for business applications. For each of the qualities availability, integrity and confidentiality, a property, stating the according requirement category, is added. The three types of requirement categories, proposed by the Bundesamt für Sicherheit in der Informationstechnik, are realized as an enumeration. According to the Bundesamt für Sicherheit in der Informationstechnik in [Bu08b] the three categories do not measure the qualities, rather state how crucial the impact of any loss or damage due to insufficient fulfillment of one of the qualities is estimated.

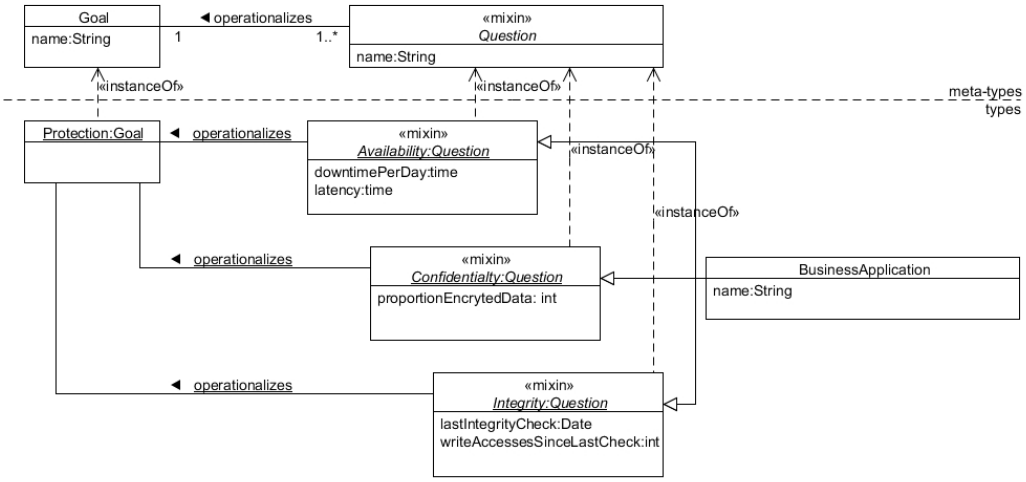


Figure 3.34: Modeling building block GOAL-QUESTION-METRIC utilized to measure protection requirements

In terms of the Goal-Question-Metric-approach, the achievement of an appropriate protection for information represents the goal. This goal is operationalized by the three qualities leading to the corresponding questions: "What is the level of availability?", "What is the level of confidentiality?" and "What is the level of integrity?". For measuring these questions, metrics are introduced to each question instance and

may be derived to the level of a quality. Transferred to the modeling building block Goal-Question-Metric the meta-level types remain the same due to the multi-level modeling approach and on type-level, on which also the other elements reflecting non-cross-cutting aspects reside, the goal instance and the operationalizing questions are introduced, as shown in Figure 3.34. The metrics for measuring the achievement of the questions are realized by placeholders, at least conveying a realistic impression of the applied building block, but have to be investigated for a real application.

When dealing with goals in EA information modeling, it is required to answer the following questions:

- How can goals be assigned to architecture elements?
- How can the achievement of goals be measured

3.2.5 Responsibilities

Within an organizational structure different managerial authorities and authorities to decide may exist. Connected to an authority to decide is to bear the consequences resulting from the decisions made. To put it into other terms, the person who bears the consequences or is responsible for a specific part of an organization, respectively, may be regarded as directly authorized to decide for this specific part of the organization. Such parts of the organization may be diverse elements of the EA, e.g. projects, single work packages of a project, business processes, business applications and so forth. Generally speaking, people can roughly be subdivided into two groups in the context of responsibilities, firstly those only executing tasks without any authority to decide and secondly those responsible for a satisfactory execution of a task or an adequate state of an architecture element. According to Krcmar in [Kr04], a lack of responsibilities in projects rises the time to respond to unexpected problems. Hence, it is obvious to assign responsible people to manageable elements of the EA to anticipate problems subject to power vacuum or simply to clear responsibilities. Since responsibilities issues occur over all layers of the EA, they constitute another cross-cutting aspect.

The modeling building block Responsibilities in Figure 3.35 addresses the mentioned issues of responsibilities by introducing a role-mixin Responsible and a mixin Manageable. Thereby, the role-mixin Responsible subsumes all roles bearing responsibility for an architecture element, in this vein, pursuing a role-based approach. Furthermore, the mixin Manageable constitutes a dispersive type denoting elements

requiring to be managed by responsible people. The responsibility for a certain manageable element is described by the RESPONSIBLEFOR-relationship between the two introduced dispersive types.

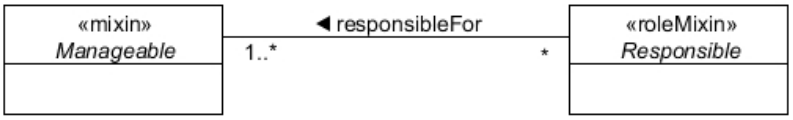


Figure 3.35: Modeling building block RESPONSIBILITIES

The modeling building block RESPONSIBILITIES only alludes to one general kind of responsibilities, but several specializations are conceivable thereof, e.g. responsibilities for the executing people related in however way to the manageable element or for the compliance with the corporate governance of the manageable element. Albeit this section does not handle issues of access control, it is obvious that the RESPONSIBLEFOR-relationship and the participating types are closely related thereto, since a person in charge should possess full access rights, as well. The relation between responsibilities and access rights is treated in-depth in the succeeding section depicting distinctions and commonalities thereof.

Some questions have to be answered for evaluating the support of responsibilities as cross-cutting aspect:

- How can people be declared as responsible for a part of the EA?
- Can architectural elements be defined as manageable or as requiring a person in charge?
- Which kinds of responsibilities can be defined?

3.3 Service aspects

As mentioned at the beginning, besides the scenarios directly reflecting requirements for EA information modeling or the underlying meta-model, respectively, there are further aspects regarding service functionalities, required for a convenient EA information modeling. These aspects are subsumed in the following scenarios.

3.3.1 Role-based access control

Many people are concerned with managing an EA, having different interests therein, needing different information and views on the EA. However, not all information in the repository is supposed to be accessed by everyone, i.e. confidential information is only supposed to be accessible by authorized personnel. Furthermore, it can be distinguished between read and write access on information, depending on the specific role of the user. Generally speaking, the access to each piece of information has to be checked, necessitating the assignment of accessibility information to each element of the EA. The mentioned restrictions and specificities in accessing information can mostly be ascribed to the role in which a certain user acts, asking for role-based access control (cf. [BMS10a]). Since issues of access control recur for elements on different layers of the entire EA (cf. Section 2.2), role-based access control has a cross-cutting nature. Despite having a cross-cutting nature, role-based access control is an aspects that is important for repository services, but does not directly belong to the modeling of the EA. It is a service aspect defining access rights for information of the repository, e.g. coming into play when querying information. The property of an element being accessible can be regarded as a dispersive quality thereof, assigned deliberately to protect information by restricting roles, allowed to access or accidentally denoting that no restrictions are defined but may be as far as needed.

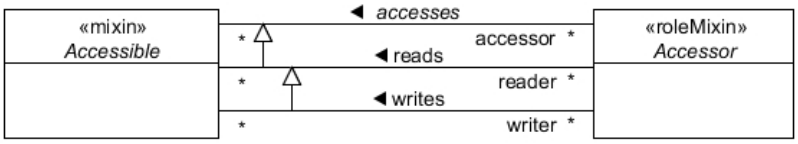


Figure 3.36: Modeling building block Accessibility

The modeling building block Accessibility in Figure 3.36 addresses the just mentioned issues of role-based access control. Therefore, the mixin Accessible is related to the role-mixin Accessor via the acccesses-relationship. The distinction between read and write access is realized by specializing the accesses-relationship into the subtypes reads and in turn into the subtype writes, that constitute material relationships (cf. [Gu06]) and may be reified by the relator hierarchy of Access, Read and Write, as shown in Figure 3.37. Thus, the modeling building block Accessibility enables a simple introduction of role-based access control into the EA information model by using the already existing roles of people involved in EA management.

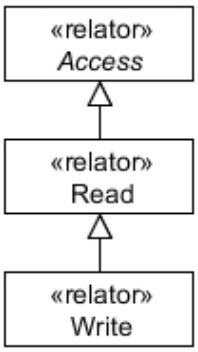


Figure 3.37: *Relator* hierarchy of ACCESS, READ and WRITE

Although a broad field of different information needs and access rights is covered by a role based access control, it might be necessary to grant access to a single person, not acting in the required role. Assuming that each person itself constitutes a unique role resolves the problem and actually addresses the nature of the problem, since adding further types to a role-based access control model fragment do not increase its clarity. Moreover, this approach may be refined to a finer grained distinction of access rights by transforming attributes into value classes being able to inherit the mixin ACCESSIBLE. For retaining the lucidity of the model fragment, attributes deviating from the access-rules on type-level and even type-level elements might be denoted by a stereotype `«accessible»`, expressing that access rights are explicitly defined on attribute-level. In Figure 3.38 this stereotype is applied to BUSINESSAPPLICATION in order to protect the more confidential attribute LICENSECOSTS for unauthorized access, as well as for a concise modeling on type-level.

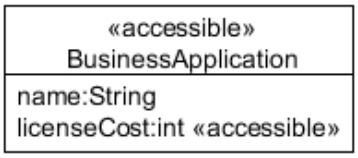


Figure 3.38: Stereotype «accessible» applied to BUSINESSAPPLICATION

As alluded to in the preceding Section 3.2.5, responsibilities are closely connected to access control issues, predefining the access rights for roles in charge. In this vein, responsibility can be regarded as a specialization of access, encompassing full access rights, that is to say read and write access, as well as further organizational commitments as described in the preceding section. Since access rights are qualities

added to architecture elements and the different roles, they have to be added to the involved types of responsibilites, as well, constituting a special case thereof in the context of accessibility. For expressing the relation to access control, the mixins MANAGEABLE and RESPONSIBLE correspondingly inherit the mixins ACCESSIBLE and ACCESSOR, as well as the RESPONSIBLEFOR-relationship subtypes the WRITES-relationship. Thereby, the relator hierarchy may be extended by RESPONSIBILITY, as depicted in Figure 3.40. Figure 3.39 illustrates these connections between access control and responsibility issues, in doing so the qualities of the modeling building block ACCESSIBILITY are utilized by the modeling building block RESPONSIBIITIES. But the specificities of responsibilities are crucial enough to retain the mixins MANAGEABLE and RESPONSIBLE, since they are reflecting a quality of elective elements fitting in certain schemas of manageability asking for responsible people, whereas access rights may be assumed to be pervasively required.

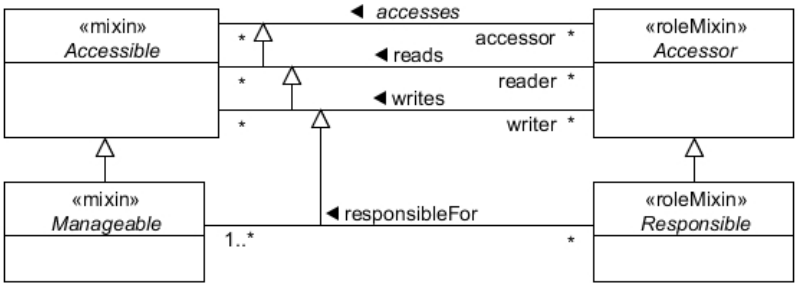


Figure 3.39: Modeling building block ACCESSIBILITY extended by responsibilities

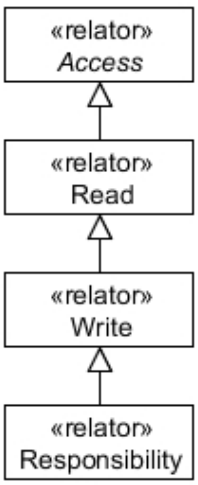


Figure 3.40: *Relator* hierarchy ACCESS, READ, WRITE, and RESPONSIBILITY

Subsuming issues of role-based access control, the following questions have to be answered:

- How is the access to elements controlled?
- Are there different levels for access control, e.g. type- and attribute-level?
- How can users be subsumed into groups with similar access rights?
- How can elements be defined as accessible by a role?
- Which types of access rights can be defined?

3.3.2 Queries

Defining an appropriate information model is a prerequisite to store specific data in the repository. Subsequently, both the concrete instances and the information model make up the information basis for the EAM function. All required information has to be provided by the repository service to be processed and visualized in various views. Queries on the repository are an important functionality for accessing the required information. The needed information is normally not of a single type, rather has to be accessed using complex connections between information of different types. A prominent example for such a query is an impact analysis, as mentioned in the preceding sections. Figure 3.20 exemplifies a conceivable impact analysis starting off with strategies introducing goals, which provoke as well as demands the execution of projects that in turn affect other business applications or the entire impact chain vice versa. According to Kurpjuweit and Aier in [KA09], impact analyses on an ex ante unknown EA are a particular challenge, since information models are organization-specific and users want to perform individual inquiries on the structural relations throughout the EA information model. Hence, the individual relations between the analyzed types have to be determined while conducting an impact analysis, which might be accomplished by traversing the information model along its relationships.

Kurpjuweit and Aier further emphasize in [KA09] the specific characteristic of self-relationships, as used by hierarchies. Transitive queries are needed for querying such self-relationships. Moreover, Kurpjuweit and Aier distinguish in [KA09] between four types of validity for relationships introduced by elements on other hierarchy-level as the currently regarded one. Starting off with a specific element, it has to be distinguished whether the relationships of superordinate, subordinate, both of them or none of them are valid for the specific element, as schematically illustrated in Figure 3.41. In order to perform a sensible impact analysis, the distinctions between

these semantics of hierarchic relationships have to be taken into account asking further for determining the transitive closure of the field of interest. For addressing the intricacy caused by hierarchic refinable structures, Kurpjuweit and Aier allude to the necessity of hiding lower hierarchy-levels, which could even be accomplished by adequate query functionalities.

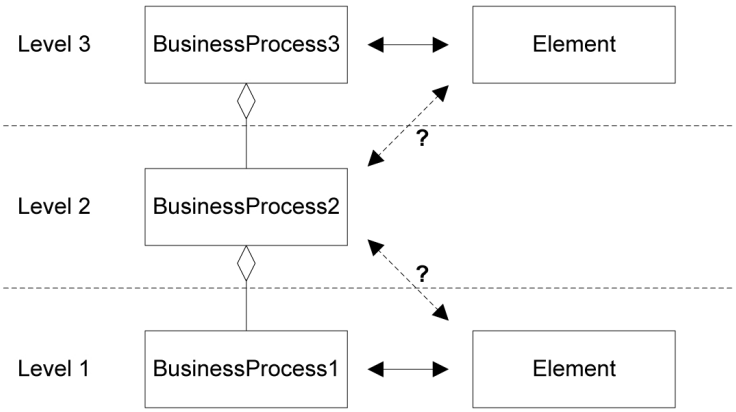


Figure 3.41: Schematic illustration of validity for relationships

Independently of whatever complex queries may be, the query results are supposed to be dependent on the specific access rights of the user issuing the query. In particular, queries are only supposed to access elements and their properties that are accessible by the user, resulting in a limited result set. This information can be provided by a role-based access control approach as elaborated in the two preceding sections. Furthermore, the integrity of information has to be guaranteed considering effects of concurrent multi-user access or uncompleted write operations, as fulfilled by almost any DBMS.

Many of the aforementioned concepts take temporal aspects, such as the temporal validity of information, into account. Querying time-dependent information necessitates specifying the period of validity for the data that is supposed to be inquired. As well, it might be required to adjust the period of validity for an already executed query in order to illustrate the changes of inquired information, e.g. the EA could easily be made comparable without separately executing a couple of queries.

The mentioned issues of querying information can be assessed by the following questions:

- How can an impact analysis be performed on an ex ante unknown information model?
- How can hierarchic structures and self-relationships be queried?

- How can different types of relationships be considered?
- How can the granularity of information be controlled, i.e. can lower hierarchy-levels be hidden?
- Are access rights taken into account while executing a query?
- How is the integrity of information ensured?
- How can temporal aspects be introduced into the query?
- Can the time-dependent validity of information be adjusted after conducting the query (without need to conduct it again)?

3.3.3 Information model changes

According to Buckl et al. in [BMS10b, Bu08a, BMS10a, Ma09], no standard EA information model so far exists, in spite of a plethora of research endeavors therefor, for which reason it is regarded as an organization-specific design artifact, underlying the dynamic of a changing environment. Hence, information models are supposed to refine along with environmental changes to hold up an appropriate alignment of business and IT support. Pursuing this native goal of EAM necessitates adapting the information model in the repository. In this context adapt means that it is indispensable to be able to change, delete or at least to hide predefined types, properties, relationships and all other elements, as well as to introduce new of them. Along with the definition of various information model elements, also their properties have to be specified, e.g. whether features have a default value or have mandatorily to be set.

As information model changes are assumed as indispensible, an appropriate maintenance of data contained in the repository in the case of an information model change is the directly following requirement. The maintenance of data is thereby achieved by establishing a valid state of the repository or to force the enterprise architect or other responsible people to produce a valid state, without any data loss. The establishment of a valid state might be automated by the repository service or partially to entirely be based on manually defined workflows or procedures to achieve such a state.

In order to evaluate functionalities addressing information model changes, the following questions have to be answered:

- Is it possible to introduce new classes/attributes/relationships to the information model?

- Can defined classes/attributes/relationships be adapted, hidden or deleted?
- Can properties or relationships be declared mandatory and can a default value be specified?
- How does the tool react to changes of the information model, especially to changes on classes/attributes/relationships for which data is contained in the repository?
- Does the tool provide standard actions on the repository data in case of information model changes? Can these actions be defined manually?
- Does the service retain a valid state of the stored data?
- Do information model changes lead to data loss?

3.4 Summary

A comprehensive investigation of a wide range of requirements for EA information modeling and the needed functionalities of a repository service therefor is conducted. As conclusion of each scenario, a set of questions, subsuming the most important requirements thereof, is devised. But there are also requirements applying to each of the scenarios, irrespectively whether general architecture aspects, cross-cutting aspects or service aspects are concerned. Requirements embracing all of the scenarios are introduced by overall prerequisites, made at the beginning of the chapter "Scenarios for EA information modeling", asking for a domain appropriate and comprehensible information modeling. As devised throughout this chapter, these requirements are concerned with the ontological correct information modeling, that reflects the nature of modeled domain elements. The following questions roughly subsume the most important aspect applying to every scenario:

- Do the provided functionalities make up workarounds twisting the available concepts without taking the increasing complexity into consideration?
- Do the models reflect the ontological meaning of the used concepts, in particular the special nature of cross-cutting aspects?
- Which perspective have to be taken for the EA information modeling, a domain or a technology centered one?

4 Evaluation of repository services

The scenarios reflecting the requirements for the evaluation of repository services are elaborated in the chapter *Scenarios for EA information modeling*. Thereby, a set of questions for each scenario, as well as overall questions are derived from the detailed descriptions and illustrations, the answering of which serves the purpose of determining the level of achievement of the scenarios. In this chapter selected repositories are evaluated against these scenarios. Initially the procedure of simulating the scenarios is described in detail, in order to ensure consistency of evaluation for all assessed repository services. Subsequently, the repositories that are supposed to be evaluated are selected due to a couple of reasons that are exposed thereby. Afterwards, the selected repository services are consistently evaluated on basis of these prerequisites and the devised scenarios of EA information modeling.

4.1 Scenario simulation and evaluation criteria

In order to achieve a consistent evaluation of the repository services, three general criteria are distinguished during the assessment of the scenarios, namely the overall *fulfillment of a scenario*, the *ontological correctness of the models* and the *tool handling*.

- *Fulfillment of scenario*: It is assessed to what extent the requirements, stated in a scenario can be fulfilled by the repository. For this criterion mainly the capabilities to structure the required information in the information model is evaluated.
- *Ontological correctness of models*: This criterion evaluates how far the ontological meaning is reflected by the produced information model.
- *Tool handling*: The effort of producing the deliverables is determined, taking all pitfalls and shortcomings thereby into account. Particularly, an intuitive handling is expected, asking for, among others, a graphical modeling environment.

The results of the evaluation are illustrated by *Harvey Balls*, which are supposed to provide an overview to what extent a criterion is fulfilled and hence are more appropriate than a fine grained metric scale, requiring quantitative calculations of fulfillment. The fulfillment of each criterion reaches from an almost complete fulfillment (●) via the partially fulfillment (◑) to a complete lack of support (○). Thereby, an almost complete fulfillment states that a criterion was satisfied to the requested amount, the partially fulfillment expresses that a criterion is achievable to a certain extent, but either incompletely or unsatisfactorily, and the lack of support means that a criterion of a scenario was totally neglected by a tool or e.g. in case of tool handling totally unintuitive. For each of these criteria, no comparison beyond this ordinal fulfillment scores is possible or wanted between the evaluated tools, e.g. which best fulfilled a certain scenario cannot be deduced. For the case a criterion cannot be evaluated for whatever reason, that is stated by *n.a.* meaning *not available*, as e.g. applied to the tool handling for scenarios overall rated as lack of support and the ontological correctness of models for some of the service aspects.

4.2 Repository services selection process

The following tool evaluation does not serve the purpose of covering a representative set of all tools available on the market that provide generic repository services, rather constitutes a preselection of a few tools stemming from the fields of meta-modeling, EAM and knowledge management. Nevertheless, conducting the evaluation on the selected tools provides a comprehensive overview of how to utilize the findings of the preceding elicitation of requirements for EA information modeling and asks for an application on a broader field of tools, providing repository services. The preselection is based on experiences gathered in the *Enterprise Architecture Management Tool Survey 2008* (cf. [Ma09]), overall experiences of the research project *System Cartography* and other research projects at the chair for *Software Engineering for Business Information Systems* (cf. [se10c]) hold by Prof. Matthes at the *Technische Universität München*. Finally, the open source web collaboration and knowledge management software *Tricia* developed at the chair of Prof. Matthes, the meta-modeling platform *ADOxx* of the *BOC Information Systems GmbH* and the *Eclipse Modeling Framework* (EMF) as a modeling framework and code generation facility were chosen for the evaluation.

4.3 ADOxx of BOC Information Systems GmbH

The evaluation of *ADOxx* took place in the course of a week-long training at *BOC Information Systems GmbH* in Vienna from 14th June to 18th June 2010. The daily training was subdivided into two parts, that were a guided introduction of the functionalities in the morning, followed by an autonomous investigation thereof in the afternoon, supported by experienced employees of *BOC*. For the training week a copy of *ADOxx* of a new pre-final version offering the latest functionality extensions, was provided for putting under investigation. Since the copy of *ADOxx* was only allowed to be installed and used during the training week due to security reasons, the following evaluation of *ADOxx* is based on the experiences made and the information gathered during this week.

4.3.1 ADOxx – Tool structure

BOC provides the *ADOxx* platform as a meta-modeling based development and configuration environment to create domain-specific modeling tools. Thereby, *ADOxx* generally comprises three workspaces, the *Product Workspace*, the *Administration Workspace* and the *Modeling Workspace*. Before starting the evaluation of the scenarios of Chapter 3, the most important tool components and general modeling functionalities are briefly described in the following.

4.3.1.1 ADOxx – Tool components

The Product Workspace constitutes the *Product Development Environment*, in which new modeling products can be created, predefined components can be configured and new functionality can be defined by using the extension mechanism. In the *Meta Model Management* facility of the Product Workspaces comprehensive modifications of the information model can be conducted, new types and relationships among them can be introduced, as shown in Figure 4.1. The *information model* is called *meta-model* by *BOC*, but to avoid confusing in comparison with other chapters, the naming information model is continued in the following.

The Meta Model Management facility of the Product Workspace is the primary workspace for evaluating scenarios of general architecture aspects and cross-cutting aspects, that is to say the aspects directly concerned with information modeling. The workspace is divided in two major sections. The navigation bar on the left-hand side comprises different tabs, of which the *Library View* hierarchically structures

classes and *relationships* in the corresponding *model types* and those in turn in *libraries*, as visible in Figure 4.1. The area on the right-hand side, displays the properties of the currently selected element of the navigation bar, as shown for the class APPLICATION.

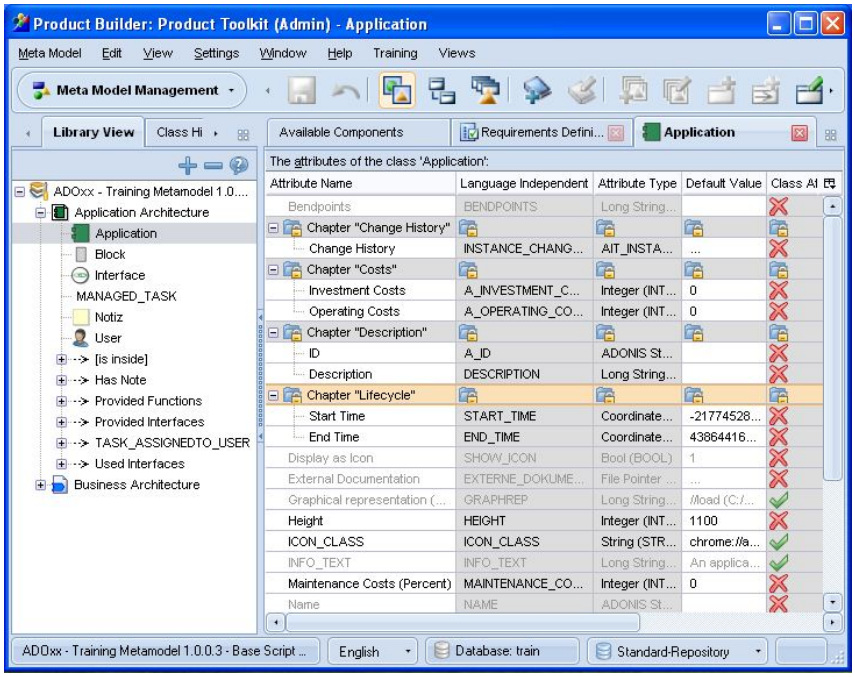


Figure 4.1: *ADOxx* – Meta Model Management facility of the Product Workspace

The Administration Workspace makes up the Configuration and *Administration Environment* of *ADOxx*, providing amongst others functionalities to manage rights on different parts of the platform, to import/export libraries, repositories, models and so forth or to perform other configurations of the platform.

The third part of *ADOxx*, the Modeling Workspace, embodies the *Modeling Environment*, that is either available as *rich client* or *web client*. In the Modeling Workspace the repository may be filled with instances of the before defined information model elements. For that, a model corresponding to a model type of the information model has to be created, in which instances of the defined classes and relationships can be established, as illustrated in Figure 4.2. Graphical representations of a relationship in an object model are only possible between objects residing in the same object model, which encompasses in turn only instances belonging to the same model type. Relationships between objects instantiating types of different model types can also be created but only using the notebook of the concerned types, as far as configured in the Product Workspace. These object models are easily created by using drag

and drop facilities on the before defined symbols to create new objects which are automatically persisted in the repository.

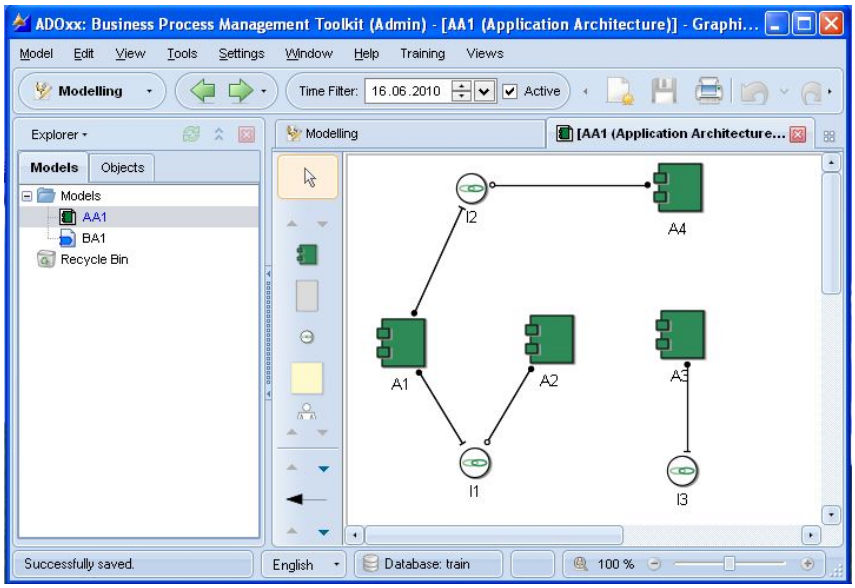


Figure 4.2: *ADOxx* – Modeling Workspace

Attribute values can be edited using the so called *notebook*, which appears after double-clicking on an architecture element, provided that the notebook is activated therefor. This activation has to be done in the Product Workspace, in which the notebook may also be modified. In doing so the information or visualized properties, respectively, may be structured by different chapters of the notebook, to which the editing fields for attributes may be assigned. An exemplary notebook of an application is shown in Figure 4.3.

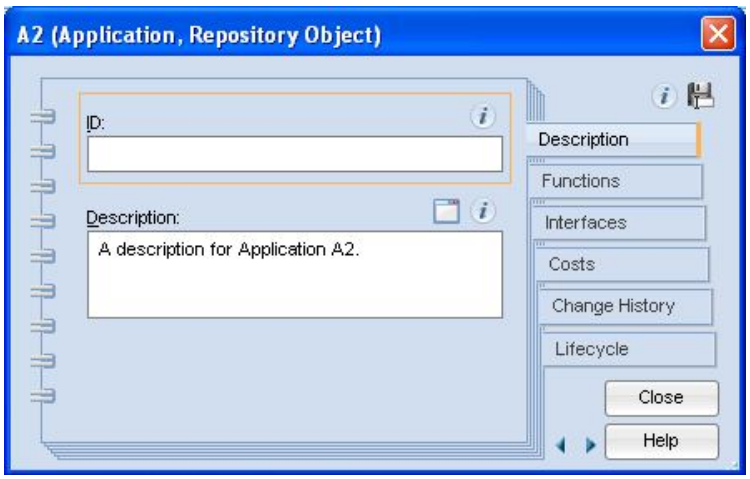


Figure 4.3: *ADOxx* – Notebook

Product Workspace and Modeling Workspace are the major tool kits used to conduct the evaluation of the scenarios. Thereby, the Product Workspace plays the major role in scenarios directly dealing with the information model. Subsequent to creating the information model fragments, the Modeling Workspace is used to assess the functionalities, whereat the notebook is required to edit the specific attribute values. After having introduced some more overall aspects of information modeling with *ADOxx* in the succeeding section, the scenarios are evaluated in the following sections.

4.3.1.2 ADOxx – General functionalities

ADOxx provides some basic functionalities useful throughout the different scenarios, such as UML-based [OM10] information model creation. An information model of *ADOxx* can freely be changed and extended, using the Meta Model Management facility in the Product Workspace. Therein, an information model is created in libraries and subdivided by model types that constitute a logical abstraction layer subsuming classes and relation classes composing a certain part of the information model, as shown in Figure 4.1.

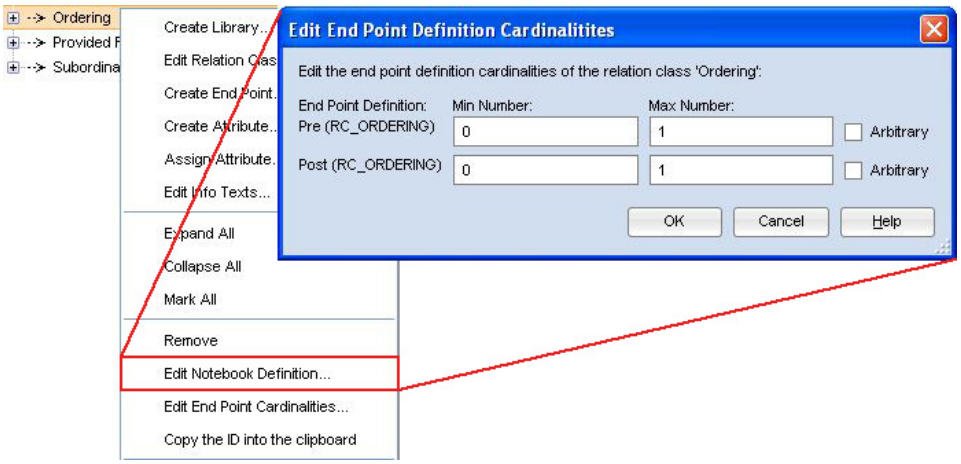


Figure 4.4: *ADOxx* – Edit end point definition dialog of relation class ORDERING

For defining a relationships, two end points, namely a FROM- and a TO-end point, have to be assigned to a relation class, defining the types which the relation class is connected to. Relation classes are restricted to two end points but an end point may define multiple types as target for the connected relation classes. For the end points, cardinalities may be specified, defining the range of elements a relation class is allowed to be connected to by a specific end point. An end point does not exclusively belong to a certain relation class rather can be utilized by different relation

classes. Furthermore, cardinalities of the end points of a specific relation class can be defined in another dialog of the relation class, as depicted in Figure 4.4. The former variant of introducing cardinalities directly to end points does not make an impact on object modeling except unpredictable behavior of the involved relationships, i.e. relationships may be graphically modeled but actually do not exist in the repository and disappear after restarting the Modeling Workspace. Looking up this problem in the documentation of *ADOxx*, reveals that end point cardinality configurations are presently not evaluated in the Modeling Workspace and the feature is not supposed to be used, as emphasized in the end point customization dialog. Maybe, this lack of support is owed to the fact that the evaluated version of *ADOxx* is a pre-final one that is supposed to appear later on this year. In contrast, the latter variant of introducing cardinalities by a dialog of the relation class actually takes an effect, which is to change the attribute dialog in the notebook for the concerned relationship end point, e.g. an end point restricted to at most one object results in a single row for specifying connections, as shown for the linear order relationship of the I-Pattern I-12 of the EAM Pattern Catalog [se10b] in Figure 4.5.

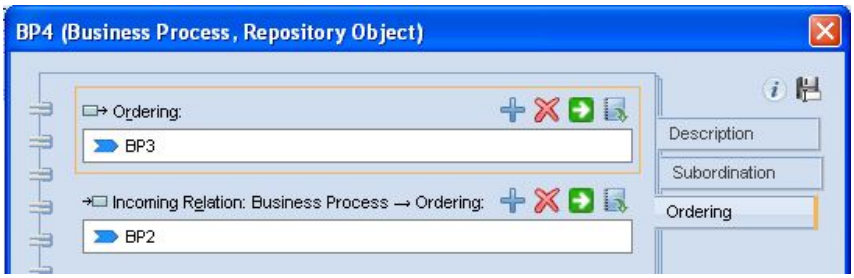


Figure 4.5: *ADOxx* – Notebook restrictions due to end point cardinalities

If one of the just described ways of restricting cardinalities is used, the consistency of cardinalities is not automatically ensured or checked in the Modeling Environment, even though functionality for checking the consistency of cardinalities on demand explicitly exists in the Modeling Workspace (cf. Figure 4.6). Hence, in the Modeling Workspace object models may be created and persisted, modeled relationships between objects of which deviate from the declared cardinalities of the information model.

Even though the definition of cardinalities for end points of a relation class seems to be the intuitive way of specifying such constraints, *ADOxx* offers further ways to specify cardinalities. So, the overall number of object instances of a class within an instance of a model type can be restricted by assigning the attribute OBJECT_CARDINALITIES to a model type, offering an xml-based definition of object cardinalities. Similarly, an attribute RELATION_CARDINALITIES can be assigned to classes, en-

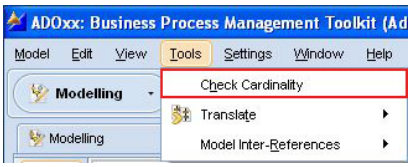


Figure 4.6: *ADOxx* – Cardinality check

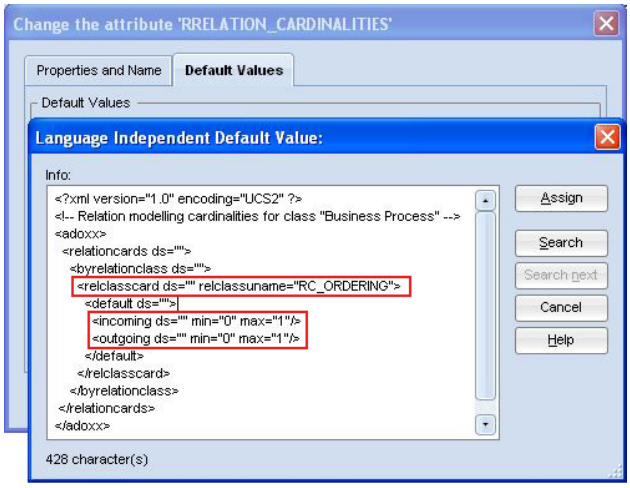


Figure 4.7: *ADOxx* – RELATION_CARDINALITIES definition for ORDERING

abling the confinement of incoming and outgoing relationships in general and for specific relation classes. If the cardinalities of the ORDERING-relationship are specified using the attri RELATION_CARDINALITIES that has to be assigned to the class BUSINESSPROCESS, even the cardinality checker of the Modeling Workspace can be utilized. In Figure 4.8, a business process is connected to two succeeding business processes, which is excluded by the definition of RELATION_CARDINALITIES, as shown in Figure 4.7. After executing the cardinality check, a notification box appears displaying the reason of failure, as shown in Figure 4.8. Thus, the modeling and checking of the ordering relationship is enabled by restricting instances to at most one predecessor and successor using corresponding end point cardinalities.

The above presented functionalities and modeling concepts enable the modeling of information models conforming to the general concepts of UML [OM10]. Generally, the information model is only visualized by tree or list views, distinguishing between class and relationship visualizations by icons in front of their names. Furthermore, there is only one kind of types, namely class, and relationships, namely relation classes, respectively, that can be configured to a certain extend. A finer grained definition of relationships is enabled by their end point definitions, since a single target definitions is not restricted to the explicitly defined target type rather also subtypes thereof are incorporate and furthermore multiple target definitions are possible. Thus, a kind of structure of relationships is enabled. For relationships, it is not directly derivable from the information model visualization which classes are connected thereby, asking for analyzing several attribute dialogs to derive such information. The deficiency of a model view of the information model is a drawback, since the actual nature of a model fragment cannot be displayed. In combination

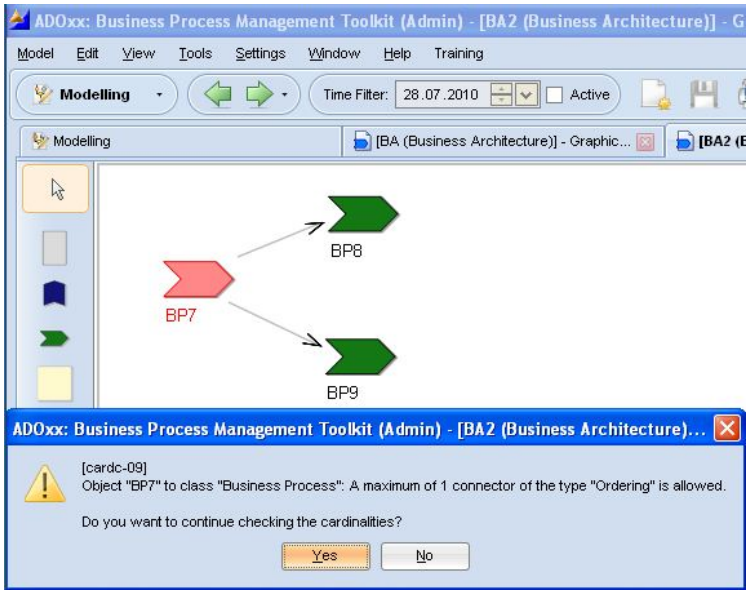


Figure 4.8: *ADOxx* – Cardinality mismatch detected by the cardinality checker

with the fact, that generally only one ontological type of universals and relationships is available, the modeling of the true ontological nature of the scenarios may somehow be restricted, but has to be specifically evaluated for each of the scenarios.

4.3.2 ADOxx – Hierarchy modeling

Since an information model of *ADOxx* can freely be changed and extended, the information model fragment of Figure 3.1 can be created under consideration of the cardinality specificities, as mentioned in Section 4.3.2. As depicted in Figure 4.9, the class BusinessProcess and for the relationships of the model fragment the two relation classes Subordination and Ordering are introduced.

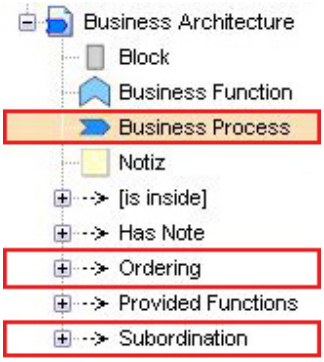


Figure 4.9: *ADOxx* – Hierarchy modeling Product Workspace

An instance of the model fragment in Figure 4.9 is depicted in Figure 4.10, defining a two-level hierarchy of four business processes. Three of them are supposed to make up a linear order on the lower level, detailing the fourth business process.

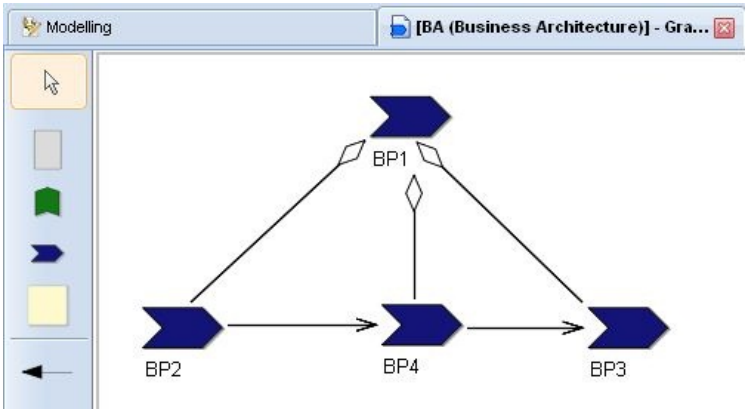


Figure 4.10: *ADOxx* – Hierarchy modeling Modeling Workspace

This standard UML-conform (cf.[OM10]) information model can be established in *ADOxx*, but the actual challenge is to introduce the extensions made in Figure 3.2. *ADOxx* offers no possibility to introduce the demanded constraints via corresponding ontological concepts, since concepts such as hierarchies or linear orders are not regarded as first class concepts by *ADOxx*. As well, standard constraint for realizing hierarchies or linear orders cannot directly be defined for the information model, since no constraint language or similar functionalities are supported. Constraints on information model-level, e.g. validating whether a self-relationship is acyclic, can be realized by the scripting functionalities of *ADOxx* using the provided *JavaScript* libraries. Every action that can be manually performed in the modeling workspace can also be automatized by the scripting functionality, offering much more functionalities beyond the standard actions. In this way, events provoked while creating an object model can be triggered and further be validated whether the triggered event is caused by an illegal modeling action, on which appropriate reactions can be initiated. Thus as complex constraints as possible with a Turing complete language, such as JavaScript, can be established, enabling the modeling of the hierarchy scenario. Using the scripting functionalities, even different kinds of part whole relationships can be specified, checking over which other hierarchy levels a certain relationship is valid. The scripting functionality offers extensive possibilities in controlling the object modeling, but only for professional users, who know how to utilize the expressiveness of Turing complete languages and accept the high effort needed for their definition. Furthermore, software artifacts are produced with very restricted reusability and are laboriously traceable for users not involved in their development.

Summarizing the evaluation of the hierarchy modeling scenario shows that the information modeling itself cannot fulfill the requirements in *ADOxx*. In combination with the powerful scripting functionality of *ADOxx*, all required information and the constraints can be established. The ontological nature of types and relationships cannot really be expressed in the information model, in particular not in case scripting is needed for realizing the scenarios, as required for hierarchy modeling. Due to the mentioned facts the evaluation in Table 4.1 is derived.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	○	◑

Table 4.1: *ADOxx* – Evaluation of hierarchy modeling

4.3.3 ADOxx – Temporal and variant modeling

ADOxx has a powerful functionality for defining time-dependencies, which is called *time filter*. Before being able to denote single types as time-dependent or in the terminology of *ADOxx*, as time filter relevant, the repository has to be defined being with time filter, as illustrated in Figure 4.11. Subsequently, classes and relationship end points can be declared time filter relevant; Figure 4.12 shows this for the end point of the host-relationship of the I-Pattern I-24 of the EAM Pattern Catalog [se10b], positioned at the side of the business application, as depicted in Figure 3.3.

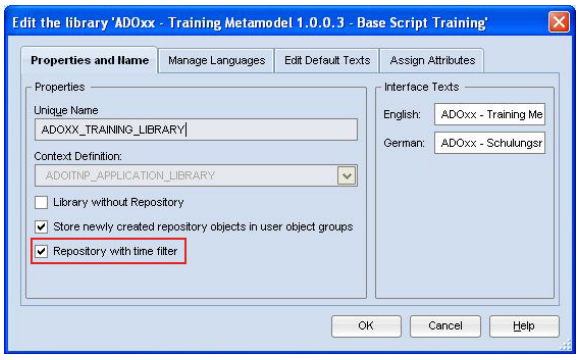


Figure 4.11: *ADOxx* – Repository with time filter

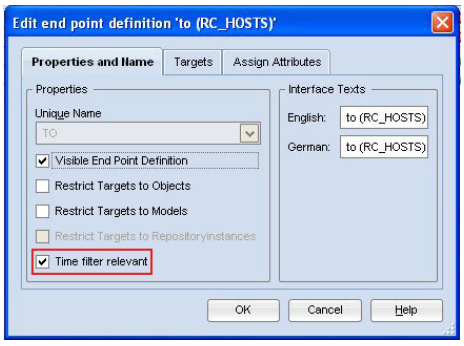


Figure 4.12: *ADOxx* – Time filter relevant end point

In the following, the example in Figure 3.3 is supposed to be realized in order to assess the time filter functionality of *ADOxx*. For that, the class ORGANIZATIONAL

UNIT and the HOST-relationship are introduced in the model type APPLICATION ARCHITECTURE, as shown in Figure 4.13. Furthermore, the TO-end point of the HOSTS-relationship is marked time-dependent, as already shown in Figure 4.12 and therefore the required attributes for stating the start and end points of validity are introduced, as depicted in Figure 4.13. Before the new assigned attributes make an effect on the time filter functionality, the default values of the library attributes LOADED_REPOINST_ATTRIBUTES and LOADED_ENDPOINT_ATTRIBUTES have to be extended by the attributes used for defining the period of validity, provoking that these attributes a permanently loaded in the Modeling Workspace. Moreover, the notebook of the end point is extended by the new chapter Life-cycle including the two new attributes.

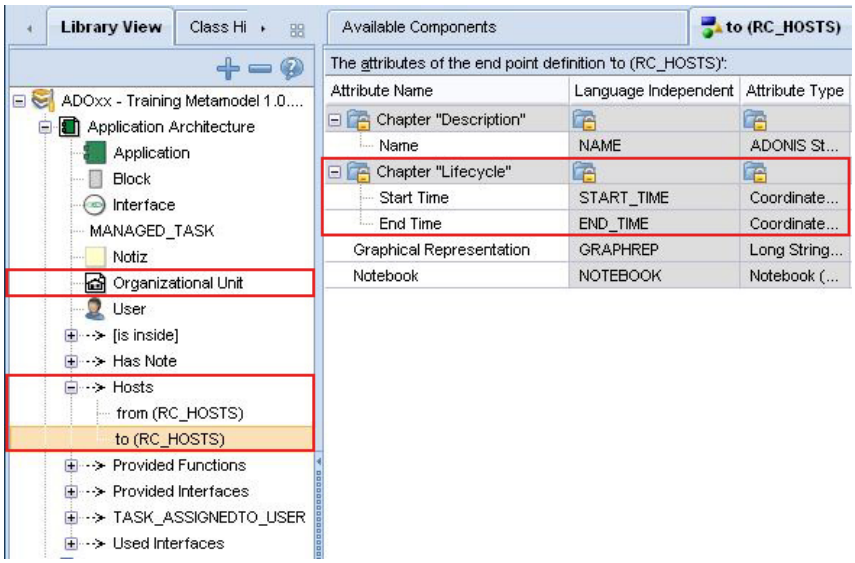


Figure 4.13: *ADOxx* – Introduction of ORGANIZATIONAL UNIT in the information model

After having conducted these steps, the Modeling Workspace is automatically extended with a new section in the menu bar for activating the time filter relevance of the currently loaded object model, as depicted in Figure 4.14. In the same figure, an instance of the I-Patten I-24 (cf. [se10b]) is displayed, showing a business application that is hosted by two organizational units. This is possible and valid, since the time filter is presently not activated, and so several hosts-relationships of a single business application, valid for a different period of time as defined in figures 4.15 and 4.16, are displayed concurrently. This notion is not incorporated by the cardinality check of *ADOxx*, which detects a cardinality violation, when performed using the example without activated time filter. Activating the time filter resolves

this problem, since obviously only displayed objects are taken into to account for checking cardinalities.

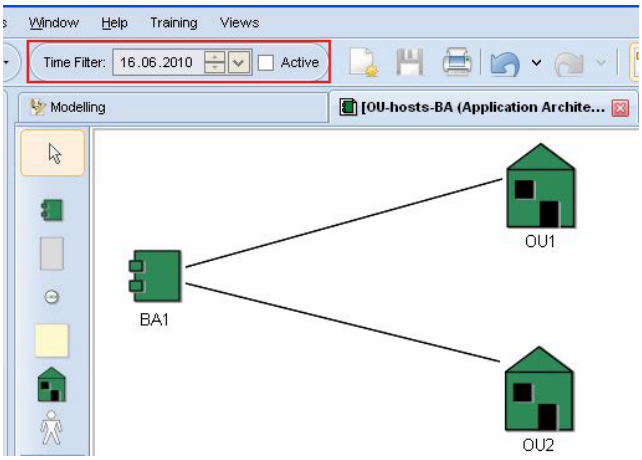


Figure 4.14: ADOxx – Modeling Workspace with time filter and instance of I-Pattern I-24

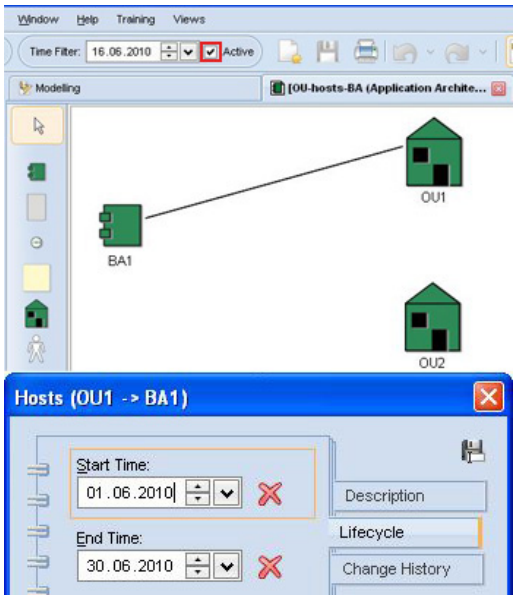


Figure 4.15: *ADOxx* – Valid object model of June 2010

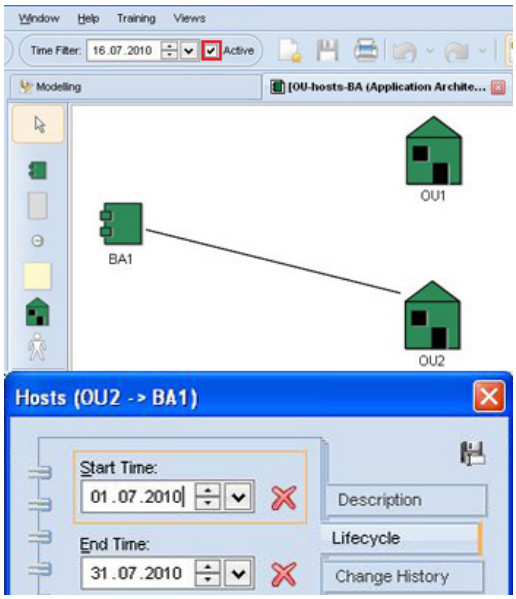


Figure 4.16: *ADOxx* – Valid object model of July 2010

In Figure 4.15 and Figure 4.16 the time filter is activated and hence only valid objects are displayed for the configured time illustrating the HOST-relationship swaps between June and July 2010 in accordance with the defined validity of the HOST-relationship instances in the depicted example. Thus, temporality is introduced for the HOSTS-relationship of I-Pattern I-24 using the time filter functionality of *ADOxx*. Thereby, the activation of time-dependency by check box can be compared

to assigning the mixin TEMPORAL of the model fragment in Figure 4.17. Using the time filter of *ADOxx*, temporality is meaningful introduced reflecting the ontological nature of this aspect.

Figure 4.17: *ADOxx* – CHANGE HISTORY of an end point

Temporality is fairly easy achieved by exploiting the time filter functionality of *ADOxx*, but even functionalities for achieving bitemporality are provided by default. Therefore, the predefined record type CHANGE HISTORY can be assigned, documenting at which point in time, by which user, from which old value to which new value, and even in which language of the value, an attribute change has taken place. Changes are not automatically documented after having assigned the record type CHANGE HISTORY and has to be manually administered for each change in the repository, by default. An automation of administering the CHANGE HISTORY is achievable by utilizing the scripting functionality of *ADOxx* by triggering change events and creating each time a corresponding change history entry.

Since variant modeling is closely connected to the scenario projects, variant modeling using projects is evaluated therein. Besides the realization by projects, *ADOxx* provides a modeling type MODES that can be assigned to model types and for which the utilizable types and relationships in the Modeling Workspace may be restricted. These modes can be selected for each object model in the Modeling Workspace, enabling e.g. the modeling of object models in a DRAFT or PLANNED mode. Thus several drafts or plans of an object model can be created and exist concurrently, providing a possibility to model multiple variants for the same point in time, which have to be decided upon later by changing the mode to e.g. CURRENT.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	●	●

Table 4.2: *ADOxx* – Evaluation of temporal and variant modeling

Temporal and variant modeling could be fairly comfortable realized, even addressing their ontological nature. Only the necessity of having to use the scripting functionality of *ADOxx* and to register the attributes stating the start and end point of validity may be regarded as shortcoming. In conclusion, most of the requirements of the scenario temporal and variant modeling are satisfactorily fulfilled, as depicted in Table 4.2.

4.3.4 ADOxx – Non-rigid typing and principle of identity

In accordance with the *ADOxx* meta-meta-model, an information model primarily consists of libraries, model types, classes, subtyped into modeling classes as well as relation classes, end points and attributes. All instances thereof define and structure the meta-model and are contained in the *Global Container*. In terms of information modeling, libraries and model types serve as logical structure of types and relationships. In doing so, libraries package model types as well as classes, and model types in turn can structure classes. Attributes can be assigned to all of the named concepts. Once defined elements can be reused in different contexts, since all element definitions are contained in the Global Container, i.e. a class defined at first for a specific model type, may be assigned to all libraries or model types of the repository. *ADOxx* provides complex attribute types, the so called *record attribute types* that structure information in a table-like form. These record attribute types can be manually created, enabling the structuring of related information in an attribute type.

ADOxx does not explicitly know something like the principle of identity, devised by Guizzardi in [Gu05]. In *ADOxx* each element has an unique identifier that cannot be shared between different elements or ontological types, particularly only one ontological type is distinguished, that is the class or rigid sortal universal, respectively. Since classes generally make up the only available ontological type of substantial sortals, non-rigid typing is not supported by *ADOxx*. As outlined in Section 4.3.6, life-cycle as a specific case using non-rigid typing in an ontological correct modeling, is partially achieved utilizing the temporal modeling capabilities of *ADOxx* for bypassing type-changes. For realizing a dispersive type that can apply to several other concept, an abstract type encompassing all required properties might be utilized. Though, *ADOxx* does not allow multiple inheritance, which is necessary in all cases the abstract class is supposed to be applied to a type, already participating in a type-hierarchy. Although multiple inheritance and a principle of identity are not supported, record type attributes can be utilized to add structured information to a type, which is comparable to some impacts coming along with a mixin-type.

The essential distinction is, that a record type attribute reduces a mixin from a general term that applies to a particular implying some properties, to the originally implied properties, while omitting the reason or general term, respectively, causing this properties. Hence, record type attribute are a useful means to create reusable structured information, but cannot be regarded as an alternative for a dispersive type as assessed in this section.

Even though a few aspects of the requirements may be achievable, the two general architecture concepts assessed by this scenario are regarded as not fulfilled, since workarounds to partially achieve the requirements of a scenario are evaluated throughout the concrete applications of general architecture concepts to scenarios of cross-cutting aspects. In contrast to the concrete application, the use of this section is in explicitly evaluating the nature of these two concepts, that are thus not supported by *ADOxx*, resulting in the scores in Table 4.3.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
○	○	*n. a.*

Table 4.3: *ADOxx* – Evaluation of non-rigid typing and principle of identity

4.3.5 ADOxx – Multi-level modeling

The meta-meta-model of *ADOxx* is not changeable, but can be extended by new attribute types using the record type attribute facility, which is actually not regarded as a meta-meta-model change. The modeling takes place on meta-level, where the information model is defined in the Product Workspace and on object-level, where the creation of object model and repository objects is performed in the Modeling Workspace. Generally, *ADOxx* provides these two modeling levels for modeling the EA, but within the meta-level model types can be regarded as a means for creating additional abstraction levels. Model types enable a logical structuring of elements based on their ontological abstraction level or meta-level, respectively, as illustrated for the ontological structure of I-Pattern I-26 in Figure 4.18.

Nevertheless, a true multi-level modeling cannot be achieved by model types, since besides a kind of structuring in abstraction levels, typical concerns of multi-level modeling are neglected. Model types do not natively provide functionalities to be instantiated to types of other model types and hence do not supply properties to

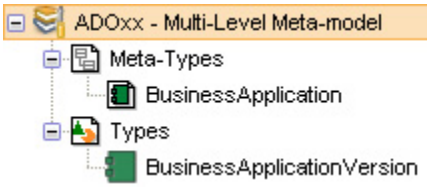


Figure 4.18: ADOxx – Multiple abstraction-levels by model types

lower ontological levels. This functionalities may be achieved using the scripting functionality of *ADOxx* by developing routines that check the consistency between different model types corresponding to their ontological nature and automatically create instances or take over properties of higher ontological levels. In summary, the partially achievements are shown in Table 4.4 are derived.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	◑	◑

Table 4.4: *ADOxx* – Evaluation of multi-level modeling

4.3.6 ADOxx – Life-cycle

As mentioned in Section 3.1.3, *ADOxx* does not provide non-rigid types or a principle of identity, for which reason it is not possible to realize the life-cycle of a business application (cf. Section 3.19) using phased-sortals (cf. [Gu05]). Nevertheless, the temporal modeling capabilities of *ADOxx* can be used to model the life-cycle phases of a business application by subtyping into temporal-dependent life-cycle phases having a kind of expiration date, as shown in Figure 4.19. Thereby, the attributes for the period of validity are inherited from the superclass BusinessApplication and only the time filter relevance of the subtypes has to be activated manually. The visibility of the superclass BusinessApplication may be deactivated to prevent an instantiation in the Modeling Workspace. Furthermore, the business application, which a specific partition of life-cycle phases belongs to, may be realized by either an identifier stating the common hypothetical business application or an actually associated instance of the superclass. The former version necessitates constraints for guaranteeing consistency and the latter one produces an additional persisted object that is why neither of these solutions is regarded completely appropriate. The definition of subtypes composing the different life-cycle phases and a period

of validity themselves do not realize a consistent life-cycle modeling, since further constraints are not accounted for yet. The scripting functionality of *ADOxx* enables the definition of these further required constraints for ensuring that only one business application subtype or life-cycle phase, respectively, is valid at a certain point in time, only valid state transitions can be performed and further restrictions are complied with.

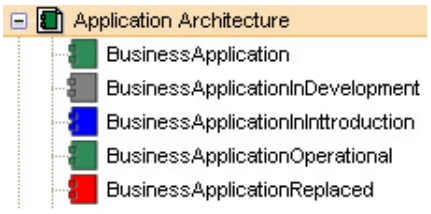


Figure 4.19: ADOxx – Business application life-cycle subtypes

Using the time filter and the scripting functionalities of *ADOxx* the modeling building block LIFECYCLED devised by Buckl et al. in [BMS10a] can be applied to elements of the EA. The superordinate business application asks for an additional attribute as identifier or an association to the superordinate business application, which can ensure in combination with corresponding constraints the bearing of the same identity throughout a specific partition of life-cycle phases. In any case *ADOxx* supplies an additional unique identifier to each of the life-cycle type instances. Moreover, an element can participate in different relationships and can possess different properties dependent on the life-cycle subject to their realization by discrete types. Technically, the requirements of life-cycle modeling can be fulfilled, but only aspects realized by the time filter functionality reflect the ontological nature life-cycles. The constraints, which have to be realized by the scripting functionality, partially require deep interferences in the tool functionalities in order to ensure the consistency of a life-cycle, for which reason the effort of their realization result in an empty Harvey Ball for the tool handling, as shown in Table 4.5.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	◑	○

Table 4.5: *ADOxx* – Evaluation of life-cycles

4.3.7 ADOxx – Projects

Projects are the major means for changing the EA and its elements. To model these concepts actually asks for a mixin AFFECTABLE that is supposed to be inherited by all architecture elements that are potentially able to be affected by projects. *ADOxx* does not directly support such a mixin type to identify architecture elements or a similar corresponding dispersive type, but provides relation class end points that can target multiple modeling classes, as depicted in Figure 4.20. Thus, an AFFECTS-relationship can be defined relating projects to a range of other types and even attributes can be assigned to an end point, that may bundle the properties, which would otherwise reside in a mixin-type. This alternative is almost as expressive as the mixin AFFECTABLE, since objects, the type definitions of which participate in an AFFECTS-relationship, can thus be identified as affectable by projects, which is closely related to the assignment of the mixin AFFECTABLE. The difference lies in shifting properties from the modeling class to the relation class end point, for which reason properties of the end points are as recently available for an object as an instance of the relationship is assigned to a specific object. Due to this reason, amongst others, a relation class end point on its own is not sufficient for substituting a mixin in general, but can realize the specific case of the mixin AFFECTABLE, which is primarily used for denoting the AFFECTS-relationship, as long as no further properties are required.

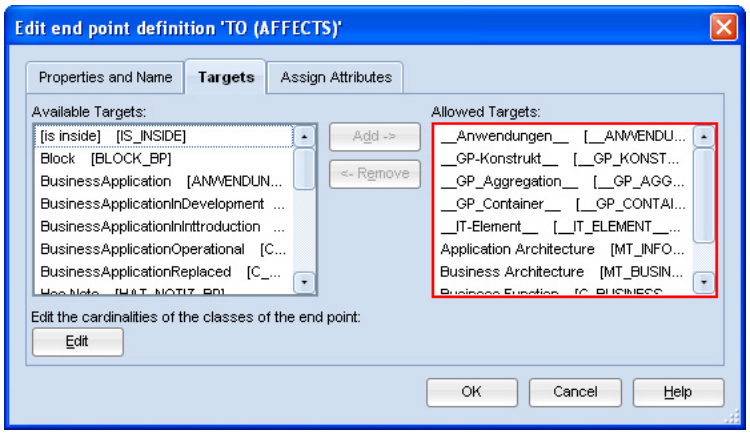


Figure 4.20: ADOxx – Dialog of an end point for assigning target classes

ADOxx realizes relationships via relation classes and corresponding end points. However, relation class is a subtype of class and only the other subtype, namely modeling class can establish inheritance hierarchies and therefore a reification of relationships is supported, but no inheritance relationships thereunder. Hence, the relator hierarchy in Figure 3.22 has to be modeled using unrelated relation classes, the usage of

which has to be synchronized by constraints asking for the scripting functionality of *ADOxx*.

The nature of projects, having different phases with a period of validity of each, can be achieved similar to life-cycles, as described in Section 4.3.6. In combination with the findings of Section 4.3.3 the modeling of variants can be achieved by a sufficiently expressive AFFECTS-relationship and temporal-dependent projects. Thus, the modeling building block ORGANIZATIONALUNIT-HOSTS-OPERATIONALBUSINESSAPPLICATION (cf. Figure 3.25) can be realized, but only using the aforementioned workarounds for the depicted concepts, which only partially reflect the ontological nature of the modeling building block, resulting in the scores in Table 4.6.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	◑

Table 4.6: *ADOxx* – Evaluation of projects

4.3.8 ADOxx – Standardization

The capability of conforming to a standard or to compose a standard, cannot be realized by a mixin or an abstract type, since mixins are not supported and for using abstract types, multi-inheritance would be required in multiple cases, that is not supported by *ADOxx*, too. Standards can be modeled as intermediate types related to elements, capable to conform to a standard and elements appropriate to compose a standard. Thereby, the flexibility of end points can be used to realize relationships to several standardizable types or several elements defining standards, respectively. Thus, the CONFORMSTO-relationship and the ALLOWEDFOR-relationship of the modeling building block STANDATD-STANDARDIZABLE can be introduced to the information model, but the attributes contained by the mixin STANDARDIZABLE have to be manually assigned to each participating type.

Even though the CONFORMSTO-relationship can be established by an auxiliary class STANDARD, its true nature of being derived of the actual context of the standardizable types cannot be realized, since only one type of relation classes is supported. In terms of Guizzardi in [Gu05, Gu06] the provided relation classes can be regarded

as a material relationship and so formal relationships, such as the conformsTo-relationship are natively not supported by *ADOxx*. Derived attributes are also not provided by default, but such a functionality may specifically be developed for the standardConform attribute of the modeling building block Standard-Standardizable using the scripting functionality of *ADOxx*. So as to distinguish between deliberately non-conforming types and a lack of documentation, a concept, such as NoArchitecturalSolution (cf. I-Pattern I-67 [se10b]) or NoStandard can be utilized, as well as an appropriate extension of the derivation functionality for the standardConform attribute may be developed, but the ontological property-type change, according to the two states of a standardizable architecture element cannot be achieved.

After having introduced a model fragment to document standardization into the information model, the standard conformance can be documented and changed by instantiating the corresponding relationships and types. An automatic adaption of the conformed standard in accordance with the changing contexts of an architecture element is not possible by default, since formal relationships are not a concept of *ADOxx*, but could be acquired by scripting. The required functionalities are regarded partially fulfilled, since most of the required information can be structured in the information model, but some of the required functionalities, such as for formal relationships, may only by realized by deep interventions in the tool functionalities. Furthermore, the ontological nature of standardization is not reflected in the solutions for structuring the information, as summarized in Table 4.7.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
◑	○	◑

Table 4.7: *ADOxx* – Evaluation of standardization

4.3.9 ADOxx – Goals

If goals and the measurement of their achievement are supposed to be realized by the *Goal-Question-Metric*-approach, a multi-level modeling approach and mixin types will be required for an ontological correct modeling of the scenario, as outlined in Section 3.2.4. A kind of multi-level structuring can be achieved by using the model types of *ADOxx*, as described in Section 4.3.5. The mixin types have to be substituted by either the corresponding attributes at the place of architecture elements, the goal

fulfillment of which is supposed to be measured or introducing discrete types for each instance of QUESTION that are related to the corresponding architecture element type.

Comparable to the succeeding section, the required information can be structured in the information model, at least by utilizing the scripting functionalities, but the ontological nature of the introduced types is only partially expressed by the utilized functionalities and concepts. In summary the fulfillments of criteria in Table 4.8 are deduced.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	◑	◑

Table 4.8: *ADOxx* – Evaluation of goals

4.3.10 ADOxx – Role-based access control

ADOxx provides the *User Management* component of the Product Workspace to manage users, user groups and their access rights. Users and different user groups can be created and edited in the *User Catalog*, as depicted in Figure 4.21. Access rights can be defined for both user groups and users, which can both use inherited access rights of user groups and overwrite the user group definitions with own ones. In the context of role-based access control, user groups can be regarded as roles played by users, granting special rights to them, since a user can participate in multiple groups at the same time, as shown in Figure 4.21.

The User Management component offers various options for defining access rights of users and user groups. Thereby, *ADOxx* primarily distinguishes between models and objects in the Modeling Workspace, files, users and user groups, languages and the different components of *ADOxx*, as depicted in Figure 4.22. Furthermore, the different options can primarily be specified for read and write access, as well as several finer grained subtypes thereof, as shown for object models of the Modeling Workspace in Figure 4.23. As diverse the configuration options of access right are, besides the components of *ADOxx* access rights can only be defined for instances of the aforementioned elements, e.g. access to information of a particular type of the information model cannot be restricted to a particular user. As well, access rights to attributes can only be defined for the attributes of an object as a whole.

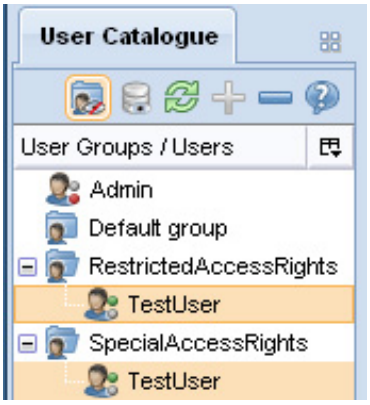


Figure 4.21: *ADOxx* – User Catalog

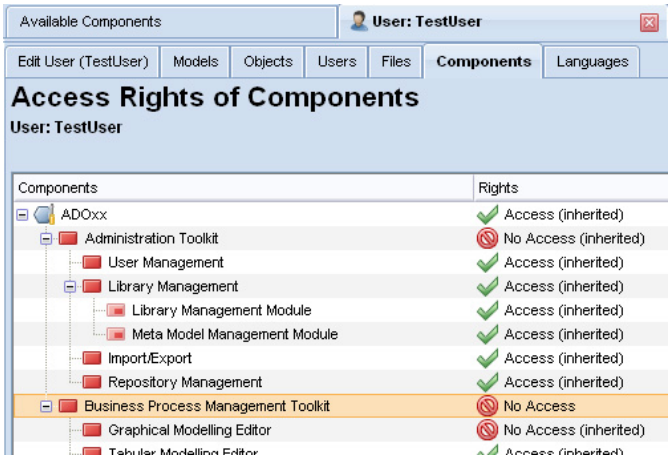


Figure 4.22: *ADOxx* – Access rights editing dialog

In conclusion, *ADOxx* provides an access control mechanism that is able to configure access rights for the different functionalities of the tool, as well as for the repository objects of the modeling concepts. It is not directly possible to restrict access to a certain type of the information model, which however might be achieved up to a certain point by restricting access to components concerned with modeling to prevent instantiating a type and to all instances thereof. Although in combination with the user groups a kind of role-based access control approach, as described in Section 3.3.1, is achievable, the requirements are only partially fulfilled due to the missing support of access rights on attribute-level and for types as a whole. Nevertheless, the provided functionality is intuitively realized, expressing the ontological nature of the scenario. The concluded evaluation is illustrated in Table 4.9.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	●	●

Table 4.9: *ADOxx* – Evaluation of role-based access control

4.3.11 ADOxx – Responsibilities

In *ADOxx* relation classes can be specialized by the Is Ownership option, which can be used by relation classes referencing shared users in the Modeling Workspace. Objects or model types connected to a user using such a relation class automatically get write access right to the connected element.

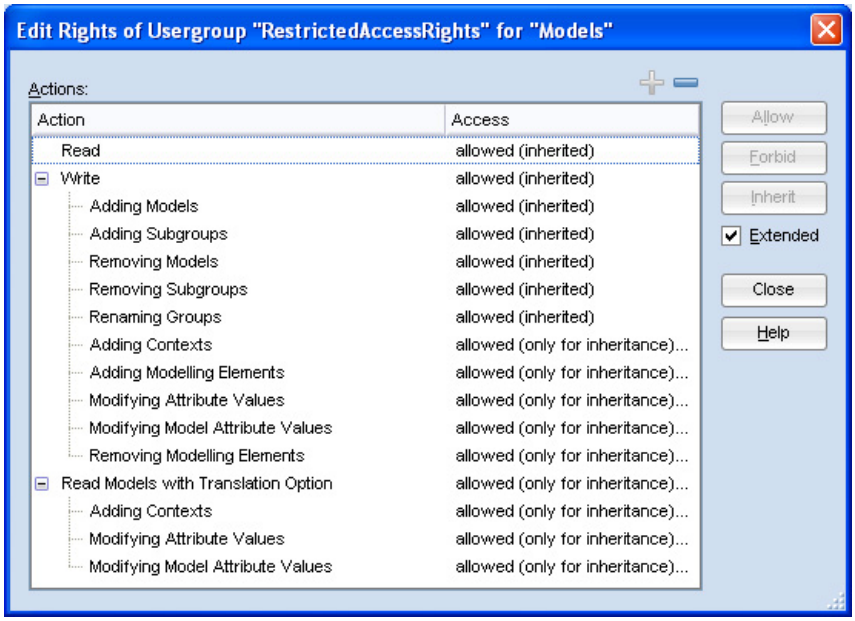


Figure 4.23: *ADOxx* – Access right types for object models

In order to introduce the RESPONSIBLEFOR-realtionship in the modeling building block RESPONSIBILITIES (cf. Figure 3.35), a relation class with activated IS OWNERSHIP option can be utilized. Such a relation class automatically grants write access rights to the connected user for the connected element, while stating the responsibility of the user. The flexibility of end points to define multiple connectable types enables a similar expressiveness as the mixin MANAGEABLE.

Relation classes with activated IS OWNERSHIP option enable an ontological meaningful modeling of the responsibilities scenario and implicitly inherit the access control functionalities, too. Hence, a complete fulfillment of the criteria is concluded in Table 4.10.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	●	●

Table 4.10: *ADOxx* – Evaluation of responsibilities

4.3.12 ADOxx – Queries

ADOxx does not provide a declarative query language, but it provides powerful libraries in the scripting API that encompass such functionalities. Queries are accomplished by traversing object models along their relationships between the contained objects. The *analysis component* of the Modeling Workspace offers a graphical editor for composing queries and provides a set of predefined query building blocks, as shown in Figure 4.24. The different kinds of query building blocks can be combined by different set operations, namely the intersection, the union and the difference of two sets, as well as an operator traversing further relationships of the objects of a result set.

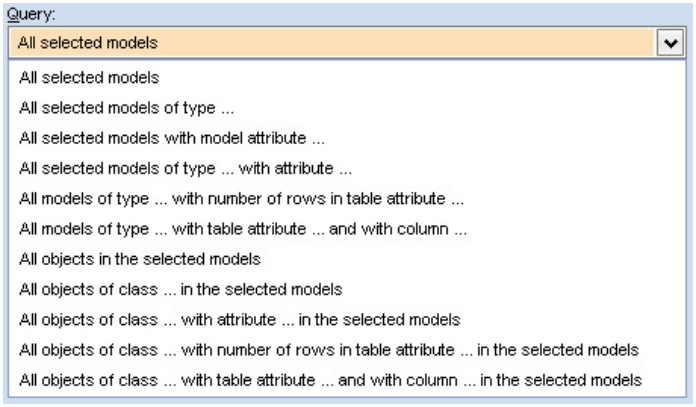


Figure 4.24: *ADOxx* – Available kinds of queries

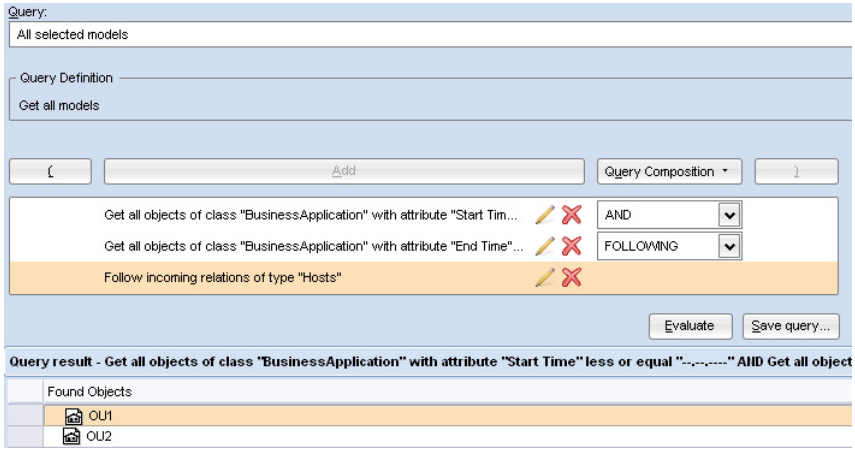


Figure 4.25: *ADOxx* – Exemplary query in the analysis component

Figure 4.25 shows an exemplary query in the analysis component of the Modeling Workspace. Firstly, a set intersection is applied on business applications having a period of validity, starting before 16th June 2010 and business applications having

a period of validity ending after 16th June 2010, resulting in the set of business applications valid at 16th June 2010. Subsequently the incoming HOSTS-relationship is followed to the hosting organizational units. The result set reflects the object model in Figure 3.6, since for the HOSTS-relationship no temporal aspects are taken into account, which is even not possible by the provided functionalities of the graphic analysis component, but can be achieved by manually creating a query using the scripting libraries.

The analysis component of *ADOxx* conveys the impression of using a declarative scripting language. In fact, the conduction of a query results in creating script code, which is subsequently executed. When saving a composed query, the JavaScript code can be edited to achieve queries that are natively not supported by the graphic analysis component, shown in Figure 4.26.

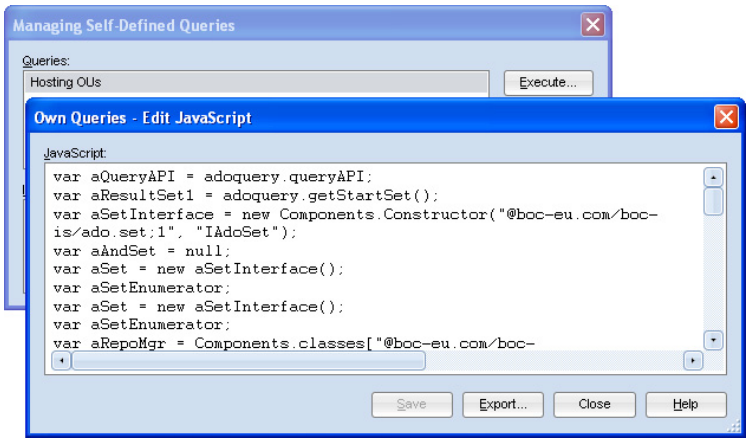


Figure 4.26: *ADOxx* – Editing field of automatically genrated JavaScript code

Besides the operator for following relationships of a result set, the analysis component does not provide functionalities to traverse the relationships of the information model, required to conduct analyses on an ex ante unknown information model. The JavaScript libraries provide such functionalities for traversing relationships of the information model enabling to determine the transitive closure that is regarded indispensable for comprehensive and complex impact analyses. Besides manually developing such functionalities, a product based on the *ADOxx* platform, namely *ADOit NP*, provides views which are able to gather information transitively, which e.g. can inquire the dependencies between two arbitrary types, even via transitive self-relationships. Again not a query language is utilized for the views rather corresponding JavaScript code is produced.

ADOxx provides powerful JavaScript libraries enabling to query repository information, which can only partially accessed via the graphic analysis component. How-

ever, the functionalities assessed in the queries scenario are regarded only partially fulfilled, even since methods to traverse models are not fully comparable to a query language. Nevertheless, the provided graphic query functionality is fairly intuitive utilized, concluding in the evaluation in Table 4.11.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	*n. a.*	●

Table 4.11: *ADOxx* – Evaluation of queries

4.3.13 ADOxx – Information model changes

In *ADOxx* the information model can freely be changed and extended as described in the preceding sections. Thereby, already created types can be removed or only hidden in the Modeling Workspace. Changes are also possible on parts of the information model, for which objects are contained in the repository, but there are no automatic actions to migrate these objects to the changed data schema corresponding to the information model. There are natively no functionalities to manually define such migration reaction to information model changes, except by the scripting functionality. Hence, information model changes may lead to an inconsistent state of the repository including unforeseen side effects of objects and relationships contained in the repository that are no longer part of the information model or even data can get lost due to the removal of types or relationships. In order to return to a consistent state of the repository, the repository data can be exported before changing the information model and re-imported after having completed the changes. Thereby, a consistent state in accordance with the defined structure of the information model is ensured and simple changes may be compensated, but a comprehensive migration to the changed information model is not automatically performed, necessitating a manually adaption thereto. A manual migration of the repository data can be achieved by the scripting functionality.

ADOxx provides the possibility to define default values for each supported language, as well as a language independent default value for attributes, whereas relationships or their end points cannot have default values. Attributes cannot really be declared mandatory, since their values can only be compulsorily set by default values. The definition of an relationship instance can be enforced by specifying their multiplicities greater than zero.

To sum up, the information model can freely be changed in any state of the repository, except for elements presently accessed by other users. But a migration of repository data due to a changed information model is natively not supported, neither automatically nor by re-importing repository data, but may manually be achieved by scripting an import algorithm. The evaluation is summarized in Table 4.12.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	*n. a.*	◑

Table 4.12: *ADOxx* – Evaluation of information model changes

4.3.14 ADOxx – Summary of evaluation

Scenario	Fulfillment of scenario	Ontological correctness of models	Tool handling
Hierarchy modeling	●	○	◑
Temporal and variant modeling	●	●	●
Non-rigid typing and principle of identity	○	○	*n. a.*
Multi-level modeling	◑	◑	◑
Life-cycles	●	◑	○
Projects	●	◑	◑
Standardization	◑	○	◑
Goals	◑	◑	◑
Responsibilities	●	●	●
Role-based access control	◑	●	●
Queries	◑	*n. a.*	●
Information model changes	◑	*n. a.*	◑

Table 4.13: *ADOxx* – Summary of evaluation

ADOxx fulfills most of the requirements and provides a fairly comfortable graphic environment to realize the scenarios in most of the cases. In particular, the utilization of the time filter or access control functionalities is regarded as a realization, reflecting the true ontological nature thereof. In contrast, the need of the scripting functionality resulted in impairing the evaluation of the ontological correctness of models and tool handling, but is needed in most of the scenarios for a complete fulfillment. The evaluation of the different scenarios is summarized in Table 4.13, providing an overview of the general EA information modeling capabilities of *ADOxx*.

4.4 Tricia of InfoAsset AG

Tricia is an open source *Java* platform, continuously refined and extended at the chair of *Software Engineering for Business Information Systems* at the *Technische Universität München*. *Tricia* is used to implement enterprise web information systems and social software solutions (cf. [BMN10]) including integrated web collaboration services for members of an extended enterprise, such as wiki collaboration, personal and team blogging, file and directory sharing, social networking, content publishing and site navigation (cf. [In10]). Thereby, *Tricia* follows a data model driven approach to system implementation, capturing substantial parts by domain-specific models, namely *data model*, *access control model*, and *interaction model* (cf. [BMN10]). This evaluation is primarily concerned with EA information modeling for which reason the focal point lies on the data model and the corresponding *data modeling framework* of *Tricia*.

4.4.1 Tricia – Tool structure

According to Büchner et al. in [BMN10], an application implemented on the *Tricia* platform has a modular layout consisting of the *core* and one or more *plugins*. The core defines abstractions required by virtually all applications built on *Tricia*, getting extended by plugins, which in turn can depend on the core and each other, while spanning a graph of dependencies, that has to be acyclic (cf. Figure 4.27).

Within the composition of core and plugins, each plugin defines a data model, an access control model and an interaction model, in doing so each plugin defines a fragment of the data structure and behavior of the entire application. The core is an inherent part of the *Tricia* platform consisting of three layered Java frameworks, as depicted in Figure 4.27. Büchner et al. describe in [BMN10] that each of these frameworks provides abstractions and extension points, instantiation and customization

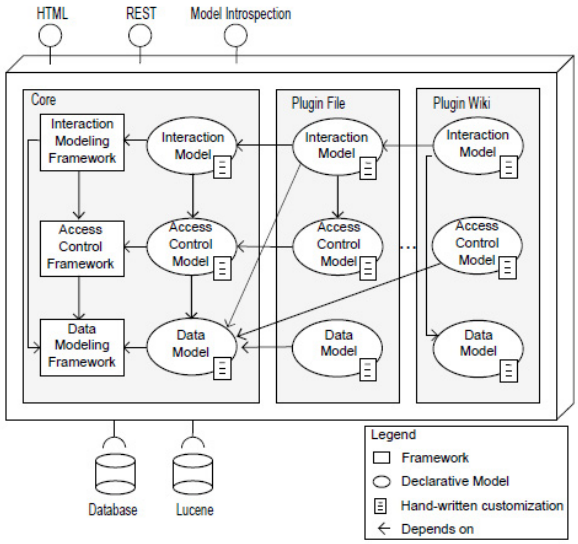


Figure 4.27: *Tricia* – Architectural overview of a typical application in accordance with [BMN10]

of which is used to build applications on *Tricia*. Thereby, declarative, model driven customizations and manually specified customizations are distinguished. Büchner et al. further emphasize in [BMN10] the central role of the data modeling framework as foundation for model driven development.

4.4.1.1 Tricia – Data modeling framework

The data modeling framework provides the modeling concepts in form of Java classes, that are instantiated and customized to structure the data models of the core and the plugins. Since the data modeling framework of the *Tricia* platform constitutes a *introspective whitebox* framework, the information model defined by its instantiation can subsequently be extracted from the code. An overview of all concepts in an UML-based diagram (cf. [OM10]) or the meta-model of the *Tricia* modeling framework, respectively, is depicted in Figure 4.28. So as to convey an impression of the information modeling capabilities of *Tricia*, the most important modeling concepts are briefly touched on in accordance with Büchner et al. in [BMN10], for more details also see [BMN10, In10].

Technically speaking, domain objects are represented by instances of the type ENTITY, that is subtyped by each type of the information model. An entity or a type of the information model, respectively, makes up a rigid sortal universal in line with Guizzardi in [Gu05]. Throughout the evaluation of *Tricia* a type of the information

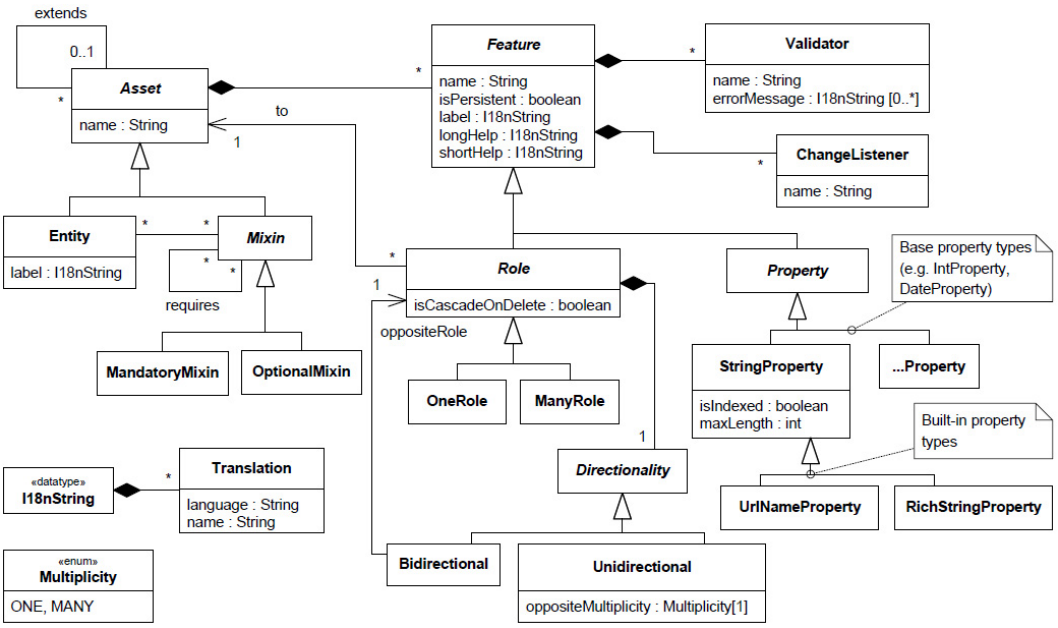


Figure 4.28: *Tricia* – Meta-meta-model of the data modeling framework (cf. [BMN10])

model refers to an instance of the data modeling framework type ENTITY. Since the data modeling framework is developed in Java, the same reuse mechanisms by inheritance as provided by Java can be used, restricting the inheritance of each type to a single other type and in *Tricia* the highest inheritance level always inherits the type ENTITY. To address this problem and to enable finer grained reuse, *Tricia* provides the concepts of MIXINS, that is comparable to mixin universals in terms of Guizzardi in [Gu05]. Mixins constitute a dispersive type, that assigned to a type may introducing additional properties and relationships thereto. In *Tricia*, two kinds of mixins are distinguished, realized by the framework types MANDATORYMIXIN and OPTIONALMIXIN. Mandatory mixins are comparable to rigid mixin universals or categories in line with Guizzardi in [Gu05], which are simply called mixins, as described in Section 2.3. The latter kind, optional mixin, makes up a non-rigid mixin universal, that can be assigned and removed at runtime. The capability of being assigned and removed at runtime deviates from Guizzardi's notion of a non-rigid mixin universal in [Gu05], but fits fairly well to the non-rigid mixin type, elaborated in Section 3.2.3. In contrast to Guizzardi in [Gu05], mixins can only be assigned to entities and are restricted to a single inheritance chain of mixins, but dependencies, stating that the assignment of a specific mixin requires the assignment of one or more further mixins, can be defined. In the following mandatory mixins are only

called mixins, since mandatory mixins are the more common mixin type, as outlined in Section 2.3 ,and hence, it is explicitly stated when alluding to optional mixins.

Properties of modeling types are realized by the data modeling framework type PROPERTY, for which several specializations are natively provided and each of which can further be customized. Relationships among modeling types are realized by modeling the end points thereof, in *Tricia* in *Tricia*. A relationship end point is represented by the framework type ROLE, which can be assigned to the type ASSET, the supertype of ENTITY and MIXIN, for which reasons both of types and mixins can establish relationships. The data modeling framework of *Tricia* specializes a ROLE to be either a one or a many end point, establishing a directed or undirected relationship, for which an opposite end point has to be specified in the latter case. PROPERTY and ROLE are subsumed by the supertype FEATURE, which defines some common properties handed down to both of the specializations. Worth mentioning thereof is the boolean property ISPERSISTENT, which specifies whether a feature is considered while persisting the containing type in the repository. Thus, non-persistent features can be utilized to construct derived relationships and properties.

Trica provides a functionality to check constraints and integrity of feature values, which is based on the concept VALIDATOR. A validator is established by instantiating the type VALIDATOR and providing the actual algorithm as hand-written customization. Validators can be assigned to each feature, accordingly to each role and kind of property. Besides manually creating validators, the core provides a set of predefined validators for properties and roles, such as the NOTNULLONEVALIDATOR for roles. A further concept applying to both kinds of features, is CHANGELISTENER. Change listeners are supposed to propagate data model changes through the system, getting notified when the observed feature is changed.

4.4.1.2 Tricia – Information Modeling

All modeling types, created during the information modeling, have to reside in a plugin that is dependent on the *Tricia* core in order to resort to the therein defined functionality, regardless of directly or indirectly. Therefore, it is assumed that models, described throughout the evaluation, are accordingly realized in such a plugin, for which reason a repetitively stating in each scenario is abstained from. Modeling in *Tricia* means to produce code, or in other terms to code the information model types, their contained features and so forth, in a Java integrated development environment (IDE). Since the *Tricia* core and the already existing plugins are implemented by plugin-projects in the *Eclipse IDE* (cf. [Ec10a]), Eclipse is the recommended IDE

for building applications based on the *Tricia* platform. Besides some Eclipse-plugins facilitating the development using the Eclipse IDE, the *Tricia* platform itself only provide limited support for information modeling. For improving the developer experiences, *Tricia* provides several templates for boilerplate code using the Eclipse built-in *Code Templates mechanism*. Thus, recurring code fragments, such as the definitions for properties or roles, can easily be introduced by corresponding skeletons of the code templates, that subsequently has to be complemented by specific code parts.

Nevertheless, *Tricia* provides no support for graphically modeling new types of an information model. Graphic representations of the information model can only be extracted as recently as the information model is created using Java code. *Tricia* provides a plugin to analyze and visualize classes of the information model based on code introspection. Thereby, tree-based or UML-based visualization can be generated, and the UML-based visualization, which is shown for the core class PERSON in Figure 4.29, even provides functionalities to edit the visualized classes. Thus, existing properties and roles can be edited or removed and new ones added, as well as mixins can be added to and removed from a class, but all of these actions are only possible for existing classes, for which reason this graphical modeling editor cannot be used for introducing new types into the information model. Generally, the editing of the classes takes place by automatically modifying or introducing code in the corresponding Java class during the editing.

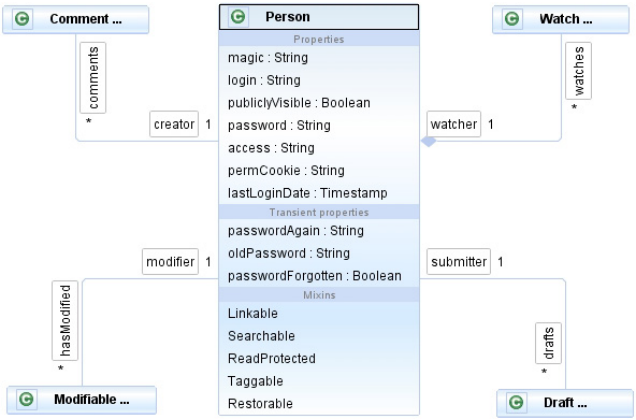


Figure 4.29: *Tricia* – UML-based editor

The *Tricia* Eclipse-plugin used for introspecting implemented classes was extended by the graphical UML-based modeling feature shortly before the submission date of this thesis, for which reason a first version thereof was put under investigation. Since in the used version an introduction of new types, that is to say the prerequisite for

modeling, is not yet supported, the feature does not enable graphical modeling and so the criteria *tool handling* is regarded as unsatisfactory fulfilled, resulting in an empty Harvey Ball in each evaluation thereof, if not otherwise stated in the concrete evaluation.

4.4.2 Tricia – Hierarchy modeling

The model fragment of the I-Pattern I-12 of the EAM Pattern Catalog (cf. [se10b]) depicted in Figure 3.1 can be modeled with the *Tricia* data modeling framework by introducing the type BusinessProcess, as well as the subordinating and the ordering relationship, as described in Section 4.4.1. The restrictions of multiplicities can be realized by corresponding role validators.

The business process hierarchy model fragment extended by constraints for realizing an acyclic subordination, that is to say a hierarchy, as well as a linear order in Figure 3.2 may be realized by creating two validators ensuring these issues. Thus, a HierarchyValidator, representing a building block for hierarchies, may be created, which can be reused for all self-relationships that are supposed to be hierarchic. Similarly, a LinearOrderValidator can be introduced to the information model for creating a building block that ensures a linear ordering. Concepts for distinguishing between different types of part-whole relationships may be developed using validators, too, either by aggregating a couple of simple validators or by creating an ontological meaningful validator representing an entire ontological type of part-whole relationships.

Assigning a validator that ensures a self-relationship to constitute a hierarchy, is regarded as denoting a self-relationship as a hierarchic or acyclic one, respectively. Since comprehensive ontological concepts can be realized by creating a corresponding validator, the ontological correctness is satisfactory fulfilled. The evaluation results are summarized in Table 4.14.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	●	○

Table 4.14: *Tricia* – Evaluation of hierarchy modeling

4.4.3 Tricia – Temporal and variant modeling

In order to achieve temporal-dependencies of architecture elements, the mixin TEMPORAL can be introduced to one of the plugins or the core by subtyping MANDATORYMIXIN. Since mixins can only be assigned to entities, relationships cannot directly be denoted as time-dependent by default. Hence, the model fragment of Figure 3.4 may be realized by introducing the auxiliary type APPLICATIONHOST to the I-Pattern I-24 of the EAM Pattern Catalog (cf. [se10b]), assigning the aformentioned mixin TEMPORAL thereto and adding a validator to the relationship role of BUSINESSAPPLICATION, which ensures the uniqueness of the HOSTS-relationship at a certain point in time. The mixin BITEMPORAL (cf. 3.7) may similarly be introduced for realizing bitemporal modeling.

Since the utilization of an additional constraint in form of a validator and the auxiliary type APPLICAITONHOST impairs the clarity of the model fragment and veil the ontological nature thereof, manually customizations of an extended mixin TEMPORAL can be used to create a more concise solution of the temporal association pattern (cf. [CEF99]). The customizations can be enforced by declaring an extension of the mixin TEMPORAL, exemplary called TEMPORALROLE in the following, abstract, so that each abstract method therein has to be manually complemented when assigning the mixin TEMPORALROLE to a type. Thus, a validator may be assigned to the period of validity of the mixin TEMPORALROLE using an abstract method therein, that specifies the role of the type, the mixin is assigned to, for which a period of validity is supposed to be specified. On basis of the specific role of the base type, the validator of the period of validity can reject combinations of time periods and role instances, if a role instance with the requested period of validity already exists. The abstract method defining the time-dependent role has to be complemented, when the mixin TEMPORALROLE is assigned to a type. Using this extended mixin for temporality, even the intermediate type APPLICATIONHOST is no longer necessary, since the role of the type ORGANIZATIONALUNIT may be directly employed by correspondingly complementing the aforementioned abstract method.

The mixin TEMPORALROLE constitutes a modeling building block for introducing temporal dependency to relationship end points, which can similarly be achieved for bitemporality. Thus, the ontological nature of temporal aspects can be bundled in a single mixin that has to be customized to its actual context of utilization. Despite needing manual customizations, the ontological nature of temporality is partially reflected by the presented solution on basis of mixins The deficiency of directly de-

noting relationships as time-dependent, resulted in an impairment of the ontological clarity of the models. The evaluation is summarized in Table 4.15.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	●	○

Table 4.15: *Tricia* – Evaluation of temporal and variant modeling

4.4.4 Tricia – Non-rigid typing and principle of identity

The data modeling framework of *Tricia* distinguishes between entities and mixins, whereas an instance of a mixin cannot exist on its own without an entity, being assigned to. Similar to the notion of sortal universals and mixin universals (cf. [Gu05]), mixins are dispersive types, which are able to apply to several particulars or entities, respectively. In this vein, the data model framework type ENTITY defines a principle of identity, inevitable being the ultimate sortal universal of each type in the information model.

OPTIONALMIXIN is the only natively supported non-rigid type that can be assigned to and removed from a type at runtime. Hence, there is no natively provided type that can be compared to non-rigid sortal universals. As well, optional mixins do not fit in Guizzardi's typology of substantial universals due to their exceptional position as described in Section 4.4.1. But exactly the capabilities due to this exceptional position enable a way of realizating roles, life-cycle phases and changeable properties. The role PROJECTMEMBER in the model fragment in Figure 3.5 may be realized by creating an optional mixin PROJECTMEMBER that establishes a relationship to the type PROJECT and can be assigned to and removed from instances of the type EMPLOYEE as needed at runtime. Similarly, life-cycles can be realized, which is described in detail in Section 4.4.5.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	○

Table 4.16: *Tricia* – Evaluation of non-rigid typing and principle of identity

In conclusion, the requirements of this scenario can be fulfilled by the modeling capabilities of *Tricia*, but the realization of roles and phases using optional mixins only partially reflects their ontological nature. The evaluation is summarized in Table 4.16.

4.4.5 Tricia – Multi-level modeling

In *Tricia* the information modeling is achieved by instantiating the concepts of the data modeling framework, which is actually implemented by subtyping the framework types. Hence, the instantiation of the data modeling framework even folds two ontological levels into the one modeling level, as normally provided along with an instantiation level by a typical object-oriented programming language, such as Java. A further division into multiple instantiation levels is not achievable in an ontological correct way. The required information may be structured by applying the type-object pattern and related patterns, resulting in a further mismatch of ontological and modeling levels, causing unnecessary complexity of the information model.

Even though the needed information can be structured using e.g. the type-object pattern, the actual purpose of this scenario is to evaluate the capability of modeling in multiple ontological abstraction levels, which is not supported by the *Tricia* platform, resulting in the evaluation shown in Table 4.17. Due to the missing support of the requirements reflected by this scenario, the tool handling criteria is omitted.

Fulfillment of scenario	Ontological correctness of models	Tool handling
○	○	n.a.

Table 4.17: *Tricia* – Evaluation of multi-level modeling

4.4.6 Tricia – Life-cycle

As mentioned in Section 4.4.4, phased sortal universals may be realized using the data modeling framework type OptionalMixin. For stating that a type has a life-cycle assigned, a mixin Lifecycled may be utilized that introduces a dependency to an abstract optional mixin, that in turn has to be specialized into a partition of optional mixins composing the different life-cycle phases. By assigning the mixin Lifecycled, the belonging to a life-cycle phase becomes mandatory for the life-cycled type, since concurrently an optional mixin of the life-cycle phases becomes

required. Using manual customizations the required abstract optional mixin, defining the partition of life-cycle phases, can be individually defined while assigning the mixin LIFECYCLED to a type of the information model. Thus, an individual adaption of the type's life-cycle phases is enabled, while utilizing the same recurring modeling building block.

Since the different life-cycle phases are represented by dispersive types that are able to bear properties and establish relationships on its own, the changing nature of the different life-cycle phases can be expressed thereby. The transition between different life-cycle phases can be controlled by utilizing an approach that controls life-cycle transitions by different kinds of work packages of projects, as depicted in Figure 3.19. The different optional mixins, composing the life-cycle phases in the exemplary outlined life-cycle realization above, may introduce the relationships to the corresponding work packages in order to enable a controlled life-cycle transition.

The scenario can be fulfilled by a partition of dispersive types that may be exclusively defined for a specific life-cycle application or reused for several types as far as possible and the transition between phases of which may be controlled by different kinds of work packages. Although a phased sortal universal is not supported on its own, the outlined solution is regarded satisfactorily reflecting the ontological nature thereof. Hence, the evaluation in Table 4.18 is concluded.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	●	○

Table 4.18: *Tricia* – Evaluation of life-cycle

4.4.7 Tricia – Projects

The mixin AFFECTABLE can directly be introduced to the information model, establishing the AFFECTS-relationship between the type PROJECT and all types the mixin AFFECTABLE is assigned to. In this way, the role of projects being the central means of affecting the EA information model (cf. Figure 3.21) is realized, while reflecting its true ontological nature. Relationships are represented by their end points at the participating elements in *Tricia*, for which reason a reification of the relator hierarchy in Figure 3.22 is not supported by the *Tricia* data modeling framework. The different kinds of the AFFECTS-relationship may be modeled by discrete roles

for each relator-type in the type PROJECT and the mixin AFFECTABLE, the coherent utilization of which has to be ensured by validators, e.g. checking that only one of these roles is used by a specific instance at the same time.

The time at which a project is executed can be introduced by assigning the mixin TEMPORAL to the type PROJECT, which adds a period of validity thereto. Such a temporal dependent project in combination with a life-cycled BUSINESSAPPLICATION, as realized as described in Section 4.4.6, can be utilized to realize the modeling building block ORGANIZATIONALUNIT-HOSTS-OPERATIONALBUSINESSAPPLICATION in Figure 3.25.

Trcia enables the modeling of projects as means of affecting the EA and the modeling of time-dependent project portfolios defining several variants of the future EA. Only the relator hierarchy is not natively supported, but the distinctions of effects can also be achieved by different kinds of project work packages, as describte in Section 4.4.6. As the modeling bulding block ORGANIZATIONALUNIT-HOSTS-OPERATIONALBUSINESSAPPLICATION realizes variant modeling in an ontological meaningful way, also the implementation in *Tricia* satisfactorily fulfills the scenario, leading to an almost complete fulfillment, as summarized in Table 4.19.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	●	○

Table 4.19: *Tricia* – Evaluation of projects

4.4.8 Tricia – Standardization

The modeling building block STANDARD-STANDARDIZABLE depicted in Figure 3.27 can be established using the mixins STANDARDIZABLE and STANDARD, which introduce the ALLOWEDFOR- and the CONFORMSTO-relationships among each other. These concepts are natively supported by the concepts of the *Tricia* data modeling framework. Thus, arbitrary architecture elements can be denoted as able to conform to a standard and in turn elements that potentially can compose a standard can be denoted, as well.

For documenting, whether a standardizable architecture element does not conform to a standard deliberately or accidentally, the modeling building block STANDARDIZABLE-NONSTANDARDIZED-STANDARDIZED-STANDARD introduces two non-rigid

mixins in order to address the true ontological nature of this issue. As outlined in Section 4.4.1, optional mixins are comparable to non-rigid mixins in the notion descibed in Section 3.2.3. Hence, a similar approach as utilized for life-cycle phases can be applied on the complete, disjoint partition of non-rigid mixins specializing the mixin STANDARDIZABLE in its two mutually exclusive states. In this vein, the mixin STANDARDIZABLE specifies a dependency on an abstract optional mixin, which is subtyped into the non-rigid mixins NONSTANDARDIZED and STANDARDIZED. Furthermore, the CONFORMSTO-relationship is shifted to the optional mixin STANDARDIZED, whereas the optional mixin NONSTANDARDIZED introduces a relationship to the type RATIONALE in order to justify an exception.

As described in Section 3.2.3, the CONFORMSTO-relationship is a formal relationship (cf. [Gu05]), which is derived from the actual context of the participating standardizable architecture element. In the data modeling framework of *Tricia*, persistent and non-persistent features are distinguished. Thereby, non-persistent roles and properties can be utilized as roles and properties, which are derived from the actual runtime values of other properties and relationships of the containing type. Hence, the CONFORMSTO-relationship can be realized by such a non-persistent role deriving the conformed standard from the actual usage or composition of architecture elements, i.e. a business application would derive its standard from the actual utilization of technologies. Thereby, the derivation algorithm is defined using manual customizations at the place of non-persistent role. Furthermore, change listener can be assigned to the roles of CONFORMSTO for maintaining the derived relationship.

In conclusion, the utilization of the concepts provided by the data modeling framework of *Tricia* as described enables a complete and ontologically correct modeling of the scenario. Moreover, the introduction of the different mixin-types creates the reusable modeling building block STANDARDIZABLE-NONSTANDARDIZED-STANDARDIZED-STANDARD that only needs a few context-specific adaptions for its assignment, resulting in the evaluation in Table 4.20.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	●	○

Table 4.20: *Tricia* – Evaluation of standardization

4.4.9 Tricia – Goals

Assigning goals to architecture elements and measuring their achievement following the *Goal-Question-Metric*-approach, means to deal with multi-level modeling and mixin-types in order to achieve an ontological well-founded conceptualization (cf. Section 3.2.4. Mixins are natively supported by *Tricia*, but multi-level modeling can only be achieved by workarounds, such as the type-object pattern or the type-square pattern (cf. [YJ02]), as outlined in Section 4.4.5.

The information structured by modeling building block GOALS-QUESTION-METRIC can be modeled using amongst others the type-square pattern to enable a context-specific assignment of appropriate metrics to the questions. The questions can be realized by mixins, thus adding the contained metrics directly the the type, for which the corresponing goal is supposed to be measured. Hence, only the mixins, introduced for questions that operationalize goals, partially reflect the ontological nature of measuring goals, resulting in a partially fulfilled ontological correctness, as shown in Table 4.21.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	○

Table 4.21: *Tricia* – Evaluation of goals

4.4.10 Tricia – Role-based access control

According to Büchner et al. in [BMN10], the *Tricia* access control framework allows to define access rights on type-level for specific types of the information model as well as on instance- or object-level for concrete instances of the types. Thereby, read, write and administration rights are distinguished that can be granted to user groups or individual users. User groups can be compared to role-mixins in Section 3.3.1, since each individual user can be a member of multiple user groups, that can be joined due to the role taken by users in a certain context. The access rights granted to the different roles or user groups, respectively, are automatically enforced by the *Tricia* access control framework and thus *Tricia* natively realizes a kind of role-based access control approach (cf. Section 3.3.1).

The access control functionality of *Tricia*, as already used by existing plugins, such as the *Wiki*-plugin, is currently realized by adding a many-role for writes and readers,

respectively. Shifting these properties into a mixin ACCESSIBLE would emphasize the nature of accessibility and foster the reuse of this feature. Currently, access control policies are restricted on type- and object-level and cannot be defined for single properties of a type. Even the shifting of the roles WRITERS and READERS, defining the access right, into a mixin type does not enable access rights on property level, since mixins can only be assigned to subtypes of ENTITY. The relator hierarchy in Figure 3.37 is natively realized by the readers and writers roles, since readers are regarded as additional users restricted to read access.

The access control functionality of *Tricia* using the mixin ACCESSIBLE reflects the ontological nature of being accessible and also a structuring of users in groups is natively supported, but access rights on attribute level cannot be specified, concluding in the evaluation in Table 4.22.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	●	○

Table 4.22: *Tricia* – Evaluation of projects

4.4.11 Tricia – Responsibilities

For defining a responsibility structure a mixin MANAGEABLE can be introduced to the information model, which establishes a relationship to users or user groups that bear responsibilities for specific, manageable architecture elements, as described in Section 3.2.5.

For extending the modeling building block RESPONSIBILITIES by access control functionalities as depicted in Figure 3.39, the mixin MANAGEABLE may subtype the mixin ACCESSIBLE and extends the properties of which by the role RESPONSIBLES. The automatic inheritance of write access rights for responsible users may be achieved by adding a CHANGELISTENER to the role RESPONSIBLES, that automatically adds new responsible users or user groups to the list of writers. The definition of responsibilities do not have to be restricted to a single type thereof, rather multiple roles can be introduced, defining separate kinds of responsibilities or different constituents of responsibility that have to be considered in conjunction.

The modeling building block RESPONSIBILITIES can be introduced to the information model, while reflecting the ontological nature of the involved concepts and reusing the

existing access control functionality of *Tricia* (cf. Sections 3.2.5 and 3.3.1). Hence, the evaluation of this scenario causes the results in Table 4.23.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	●	○

Table 4.23: *Tricia* – Evaluation of responsibilities

4.4.12 Tricia – Queries

The *Tricia* platform provides set-oriented querying using a declarative query language, as well as element-oriented querying via hyperlinks (cf. [BMN10]). The associations among instances of object models can be traversed by following the roles of an object. The information model can also be extracted and inspected from the code, since the *Tricia* data modeling framework is based on an introspective white-box framework. Furthermore, full-text or structured queries for given or arbitrary types can be performed on the repository.

Impact analyses on an ex ante unknown information model can be performed by introspecting the information model and subsequently applying queries on the extracted meta-information. The declarative query language is realized by the type-hierarchy QUERY, providing functionalities to perform queries ranging from a simple inquiry on a single type up to complex joins[5] on multiple types. Thereby, transitive queries on self-relationships are natively not supported and have to be manually implemented by traversing the relationships of the object models. Moreover, information that is possibly structured in a transitive self-relationship can be inquired by the full-text search of *Tricia*, which inquires all objects containing the entered data or text, irrespective of their information structure.

Temporal aspects, modeled by a mixin type, such as the mixin TEMPORAL, described in Section 4.4.3, are accessible via the properties stating the period of validity, that can be incorporated by a query, as well. While performing a query, the users issuing the query can only access the objects that are accessible according to their access right derived from user groups or individually defined.

The *Tricia* platform provides comprehensive query functionalities, in particular the full-text query functionality provides powerful inquiries of information without hav-

[5] cf. QueryDoc of the Tricia Javadoc

ing to know its internal structure. The deficiency of transitive queries by default, that are regarded important for intricate impact analyses and thereto connected determining of the transitive closure, concludes an only partially fulfillment of this scenario, as depicted in Table 4.24.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	n.a.	○

Table 4.24: *Tricia* – Evaluation of queries

4.4.13 Tricia – Information model changes

The information model can be freely created in *Tricia*. Afterwards types, properties, relationships and all of the other elements of the information model can be changed or removed, even if repository data exists therefor. Necessary data migration in the repository after having performed some information model changes, is automated by the data modeling framework (cf. [BMN10]). The automated migration takes place on application start, when *Tricia* compares the data model defined by the types of the information model and the existing persistent data model in the database. If differences between the new and old data model are detected, an SQL script will be created for migrating the old data schema and the existing data to the now one. This script has to be manually applied to the data base server. The data migration establishes a consistent state of the repository and adapts its information structure to the changed information model. Furthermore, there is the possibility to apply migration handler on start of a *Tricia* application, which can apply more complex migration routines that are covered no more by the automatic schema and data migration.

Properties and roles can be declared mandatory by a validator that prevents a `null` or an empty value for a feature. Furthermore, a default value can be provided by assigning a change listener that automatically sets the default value at creation of a feature. This default value can either be a static value or a compound on, assembled further runtime information.

Thus, the EA information modeling can freely be performed and also changes therein are almost automatically migrated to the repository by the *Tricia* data modeling

framework, resulting in a satisfactory fulfillment of this scenario, as shown in Table 4.25.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	n.a.	○

Table 4.25: *Tricia* – Evaluation of information model changes

4.4.14 Tricia – Summary of evaluation

The *Tricia* data modeling framework provides some powerful concepts. Mixins are propably a kind of unique selling point, which enable an ontological correct modeling of dispersive types and even a non-rigid version thereof is provided, that can be assigned to and removed from information model types at runtime.

Scenario	Fulfillment of scenario	Ontological correctness of models	Tool handling
Hierarchy modeling	●	●	○
Temporal and variant modeling	●	●	○
Non-rigid typing and principle of identity	●	◑	○
Multi-level modeling	○	○	○
Life-cycles	●	●	○
Projects	●	●	○
Standardization	●	●	○
Goals	●	◑	○
Responsibilities	●	●	○
Role-based access control	◑	●	○
Queries	◑	n.a.	○
Information model changes	●	n.a.	○

Table 4.26: *Tricia* – Summary of evaluation

Mixins in combination with validators, which may embody complex algorithms to ensure ontological types of relationships, such as acyclic self-relationships, are primarily responsible for realizing most of the scenarios in a way reflecting the ontological nature thereof. Thereby, the required modeling building blocks can quite closely be introduced into the information model. In contrast, scenarios asking for multi-level modeling are only realized by workarounds twisting the required ontological levels into the two level modeling paradigm provided by the object-oriented modeling languages Java. The deficiency of multiple modeling levels or even a possibility to structure different abstraction levels, as well as a complete graphical editor are shortcomings. The evaluations of the different scenarios are summarized in Table 4.26.

4.5 Eclipse Modeling Framework of the Eclipse Foundation

The *Eclipse Modeling Framework (EMF) Project* belongs to the *Eclipse Projects* as a subproject of the *Eclipse Modeling Project*. The Eclipse Projects endeavor of the *Eclipse Foundation* focuses on building an open development platform comprised of extensible frameworks (cf. [Ec10a]). As part of this endeavor the Eclipse Foundation defines the *EMF* project in [Ec10a] as follows:

> The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

In *EMF* a model can either be defined in Java, XML Schema [Wo04] or UML [OM10], from which the others and the corresponding implementation classes can be generated (cf. [Bu09b]). In this vein, *EMF* relates modeling concepts directly to their implementations, thus unifying the three technologies Java, XML [Wo08] and UML [OM10].

EMF only provides a simple tree-based editor for Ecore, the meta-model of *EMF*, for which reason the graphical UML-based editor of the *Ecore Tools* is used for the evaluation of *EMF*. The Ecore Tools constitute a component of the *Eclipse Modeling Framework Technology (EMFT) Project*, which is similarly to *EMF* a subproject of

the Eclipse Modeling Project. EMFT is meant to be an incubator project for new technologies that extend or complement *EMF* (cf. [Ec10a]).

4.5.1 EMF – Tool structure

The foundation for modeling with *EMF* is Ecore its meta-model and therefore the model of information models realized in *EMF*. Information models are called *core models* in *EMF*. Ecore is a small and simplified subset of full UML [OM10], thus making up an efficient Java implementation of a core subset of the *Meta Object Facility* (MOF) [OM06a]. In accordance with Budinsky et al. in [Bu09b], the *Essential MOF* (EMOF) constitutes a subset of the MOF model that is similar to Ecore. There are small differences between Ecore and EMOF, mostly confined to the naming of concepts. According to Budinsky et al. in [Bu09b], Ecore enables EMF to transparently write and read serializations of EMOF.

As alluded to above, a core model can be created from Java interfaces by introspecting the annotated Java code, XML Schema [Wo04] model definitions or UML [OM10] models. There are three possibilities to start with an UML model. Firstly, the simple tree-based editor of *EMF* or another graphical editor, such as the one of the Ecore Tools, can be used. Secondly, the *EMF Model wizards* provide an extensible framework, into which model importers for different formats can be plugged, which natively only supports model definitions created in *Rational Rose*. Finally, serialized Ecore exports of an UML tool can be imported necessitating a corresponding conversion support by the UML tool. If Java interfaces are used as source of the core model, the interfaces as well as their contained references and attributes or rather their corresponding get-methods have to be denoted by `@model` in the Javadoc annotations to be identified as model classes. The `@model` annotation can be complemented with further information, such as whether a reference is a containment reference or which type is contained in a list of references. Throughout the evaluation, the graphical editor of the Ecore Tools as a part of the Eclipse Modeling Project are utilized for creating core models. This editor overall provides the same configuration possibilities of the tree-based editor, except for i.e. some detailed type-definitions of generic data-types, and hence supports all concepts of the Ecore, required for the evaluation.

Java, UML or XML Schema serve as sources of a core model, but none of them is the utilized form of persisting core models, since all of them carry additional information beyond what is captured in a core model (cf. [Bu09b]). Hence, core models are represented in the *XML Metadata Interchange* (XMI) format, which is

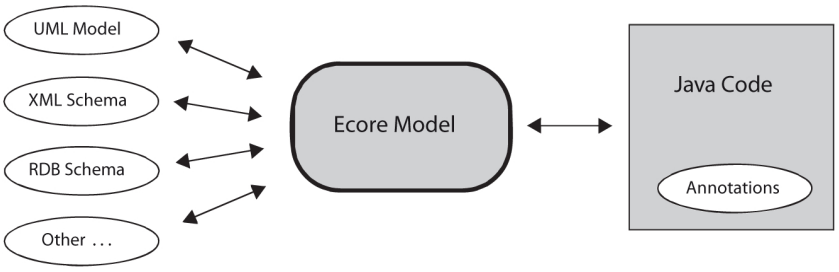


Figure 4.30: *EMF* – An ecore model and its sources (cf. [Bu09b])

the standard for serializing meta-data concisely using XML. According to Budinsky et al. in [Bu09b], the core model is the center of the EMF world between a persistent model form and the Java code. There are also other forms of persistent model representations possible instead of an XML Schema, as depicted in the "big picture" of *EMF* in Figure 4.30.

4.5.1.1 EMF – Code Generation

After having created a core model in whatever way, the core model is used to generate model classes. Thereby, *EMF* generates a Java interface and an implementation class for each core model type. This is a design choice of *EMF* that among others makes up a pattern that is required to support multiple inheritance in Java, as it is in *EMF*. Each generated interface extends directly or indirectly EOBJECT, the base interface of *EMF*, which is the *EMF* equivalent to `java.lang.Object` (cf. [Bu09b]). Along with the extension of EOBJECT, methods for returning the ECLASS of an object, the object's containing object and resource are introduced as well as an API for accessing an object reflectively is provided. These reflective API is required for generically accessing objects, whereas containing objects and resources introduce two integral parts of the persistent API of *EMF*.

Moreover, EOBJECT extends the interface NOTIFIER that introduces model change notifications comparable to the *Observer* design pattern (cf. [Ga95]). Observers can be assigned to the set-methods of an attribute and are notified when the attribute value changes, e.g. *EMF* objects can be observed to update views, dependent objects and so forth. Notification observers are called *adapters* in *EMF*, since they are often additionally used to extend the behavior of the objects, they are attached to (cf. [Bu09b]).

Besides the described interfaces and classes, there are some other important classes that are generated for the model. On the one hand a *factory*, which is supposed to

be used for creating model objects, and a *package*, which provides access to all Ecore meta-data for the model are generated. On the other hand an *adapter factory* is produced to implement type-specific adapters, as well as a *switch class* that provides a callback mechanism based on the object's type. Adapters are used extensively in EMF as observers and to extend behavior, thereby switch classes are resorted to in their implementation. Adapters are the foundation for the *UI* and *command* support provided by the `EMF.Edit` framework.

EMF is targeted on combining generated and hand-written parts of code. For identifying generated parts during a regeneration of code, *EMF* uses `@generated` markers in the Javadoc, so i.e. any method without such an annotation will remain unaffected during regeneration. A hand-written piece of code without an `@generated` tag always takes precedence over an equivalent named generated one.

Most of the information needed by the *EMF* code generator is stored in the core model, further information for generating code and consistently regenerating, e.g. for distinguishing between hand-written and generated parts, are stored in the *generator model*. The generator model provides access to all required information including the corresponding core model, which is wrapped by the generator model. When the *EMF* generator is used, actually the generator model is accessed instead of the core model, which is in turn directly accessed by the generator model. Thus, the core model remains a concise and pure information model, as it is intended to. In order to address synchronization issues among core model and generator model, elements of the generator model are able to automatically reconcile themselves with changes to their corresponding core model elements (cf. [Bu09b]).

EMF facilitates creating objects by automatically generated factories and a mechanism for persisting and referencing other persisted objects is provided, as well. For that, a default XMI serealizer is included, able to persist objects generically from any model, not just Ecore (cf. [Bu09b]). *EMF* supports persisting objects in an XML instance document, provided that the information model is defined using XML Schema. Furthermore, *EMF* allows to persist objects in an arbitrary form using hand-written serialization code and nevertheless supports transparently referencing among objects in different models or documents, irrespective of their form of persistence. At this point the aforementioned methods for returning the object's containing object and resource come into play, since an object is persisted in an Resource and objects contained by another objects are automatically persisted in the same resource. Resources in turn are contained in an ResourceSet, which is also used to create instances of Resource. According to the defined form of persistence,

different types of resources may be contained in a resource set, that manages the cross-document or cross-resource referencing, respectively.

Besides generating the core model and the corresponding implementation from a serialized source, *EMF* provides the creation of a dynamic core model at runtime using the reflective API (cf. [Bu09b]). Thus, a core model can be created without generating any Java classes, using the reflective methods for creating objects, attributes or references as well as manipulating their properties.

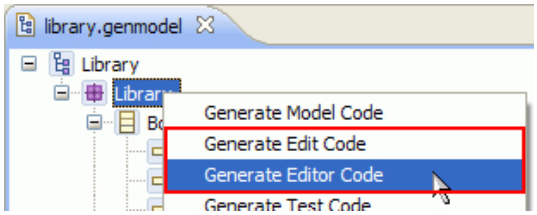


Figure 4.31: *EMF* – `EMF.Edit` code generation

The `EMF.Edit` framework is able to generate functional viewers and editors for editing instances of a core model. Therefore, the `EMF.Edit` framework includes generic reusable classes for building editors for core models, providing different *item provider*, such as content and laber provider, as well as classes to display models using standard desktop viewers, e.g. jFace, and property sheets. Furthermore, `EMF.Edit` provides a *command framework* including several generic command implementation classes. All required item providers and other classes, needed for building a complete editor plugin, can be generated by the `EMF.Edit` code generation support. After having generated the model code, a fully functional editor can be generated by conducting the *Generate Edit Code* and *Generate Editor Code* mechanisms of the generator model (cf. Figure 4.31), showing that the `EMF.Edit` code generator is simply another part of the generator model. The `EMF.Edit` code generation produces a simple tree view based editor to create model instances, that serves as starting point for developing more sophisticated editor, adapted to the specific concerns. The same actions that are provided by the editor can also be performed using the reflective API, even for dynamic core models that do not stem from a generation.

4.5.1.2 EMF – Ecore

As already mentioned, Ecore constitutes the meta-model of *EMF* that defines the structure of core models or information models, respectively. Ecore provides some concepts that are not directly included in Java, such as containment and bidirectional relationships or multi-inheritance, but nonetheless, *EMF* claims to generate correct

and efficient Java code for those concepts (cf. [Bu09b]). ECORE is defined as a core model, the self-defining nature of which enables a treatment much like any other core model. A comprehensive overview of ECORE is depicted in Figure 4.32, the most important aspects of which are described in the following. One important ECORE component is omitted in Figure 4.32, that is EOBJECT, the supertype of EMODELELEMENT and for this reason supertype of all depicted types.

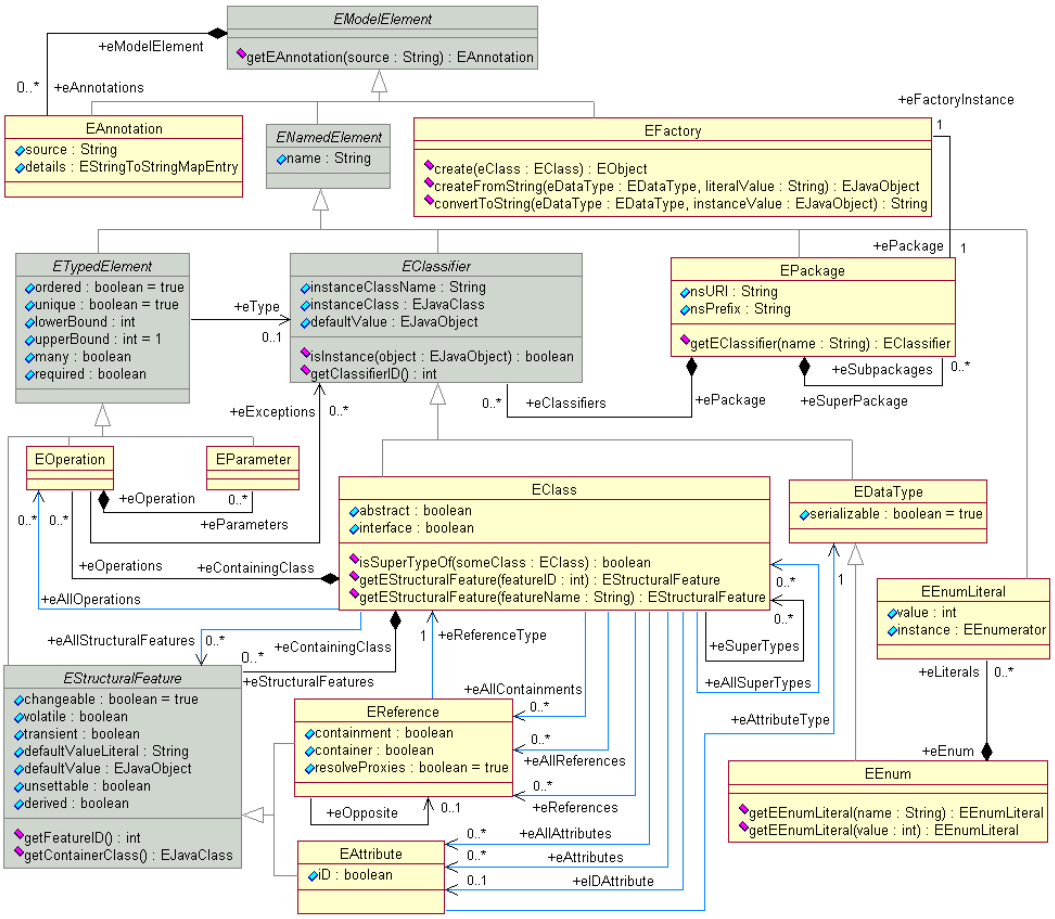


Figure 4.32: *EMF* – ECORE components and their relationships (cf. [Ec10b])

In accordance with Budinsky et al. in [Bu09b], the types ECLASS, EATTRIBUTE, EREFERENCE and EDATATYPE compose the kernel of ECORE. This model fragment encompasses the most common types that can be found in an information model and is the basis for understanding the self-defining nature of ECORE, since i.e. modeled attributes and references of ECLASS correspond to instances of EATTRIBUTE and EREFERENCE.

ECLASS is used to represent modeled classes and interfaces, which can contain multiple attributes and references. The ESUPERTYPES reference shows that a class can

inherit multiple supertypes. However, this capability conflicts with the generation of Java code, because Java only supports inheriting of a single supertype, except for interfaces. According to Budinsky et al. in [Bu09b], this condlict is addressed by using the first class in the list of eSuperTypes as *implementation base class* of the generated code, while the others are treated as *mixin interfaces*, the features of which are re-implemented in the derived implementation class. In doing so, generating code from interfaces composing a multiple inheritance creates an interface of the implementation base class that extends all of the specified suptertypes and an implementation class thereof, which only extends the defined first supertype and correspondingly implements the remainder of the interfaces. To ensure, that a specific class is the first supertype in the list of sypertypes, the reference thereto can be annotated by i.e. the stereotype `«extend»` in UML models. The attributes inherited from the different supertypes are not allowed to have equivalent names, which causes a generation failure.

An EClass inherits all structural and behavioral features of its supertypes and all of the transitively connected supertypes. The direct supertypes are refered to by the reference eSuperTypes, whereas the derived reference eAllSuperTypes provides the complete set of transitive connected supertypes. Analogously, the references eReferences and eAttributes define the directly contained attributes and references, that are extended by all inherited attributes and references in eAllAttributes and eAllRefereces, respectively. The same logic can be transfered to eStructuralFeatures and eAllStructuralFeatures, as well as eOperations and eAllOperations.

EAttribute and EReference, which represent modeled attributes and references, are both subtypes of EStructuralFeature, that in turn is a subtype of ETypedElement, referring to an EClassifier as its eType. An EClassifier is either an EClass or an EDataType, for which reason both attributes and references are theoretically able to refer to classes and data types. Therefore, EAttribute defines the derived reference eAttributeType that casts objects delivered by the reference eType to an EDataType. The derived reference eReferenceType of EReference is analogous to eAttributeType and casts EClassifier to EClass. Generally, an EReference is used to represent an end of an association between two classes, that can optionally be declared bidirectional by defining an opposite EReference or as containing, which defines the containing object for objects of the referenced type. EDataType is used for representing primitive types or Java object types, such as `java.lang.Date`.

ESTRUCTURALFEATURE, the supertype of EREFERENCE and EATTRIBUTE subsumes several commonalities of all features. Among other things, ESTRUCTURALFEATURE defines the boolean properties CHANGAEBLE, TRANSIENT, UNIQUE, UNSETTABLE and VOLATILE. CHANGEABLE defines whether a feature can externally be set. TRANSIENT defines whether a feature is considered for serialization of the containing object. UNIQUE declares whether a single value is allowed to occur more than once in a feature, which is only reasonable for multiplicity-many features. According to UNSETTABLE, it is determined whether the feature has a value assigned different from any other valid values and `null` stating that a feature is not set. Finally, VOLATILE defines whether there is any storage directly associated with a feature. According to Budinsky et al. in [Bu09b], some of these properties are only reasonable in combination, e.g. it seems not to be reasonable to define a feature volatile but non-transient. Budinsky et al. recommend to define derived features as being volatile, transient and non-changeable. This definition also exerts influence on the code generation, resulting in the omitting of set-methods for derived attributes and an annotation reminding that the implementation has to be complemented in the get-method.

4.5.1.3 EMF – Validation framework

EMF natively provides the *validation framework*, which addresses issues of checking the validity of an object's state modeled in *EMF*. The validation framework enables to define constraints and invariants to be verified directly in the model. According to Budinsky et al. in [Bu09b], a constraint constitutes a statement that has to be valid at a certain point in time, whereas an invariant is a much stronger assertion that is supposed to be valid all the time. Consequently, these two kinds of conditions are realized in different ways, since an invariant is supposed to be always evaluable asking for an easy accessible evaluation statement for any code that manipulates the object. Hence, an invariant results in an additional method in the concerned class that has to conform to a specific method signature, consisting of two parameters, of types EDIAGNOSTICCHAIN and EMAP, as well as a return value of type EBOOLEAN, which states the validation of the invariant. In contrast, a constraint is defined as EANNOTAION of a class, the source of which has to be set to the schema for ECORE, and for which a map entry has to be defined, setting its key to `constraints` and the value to the corresponding names of the constraints to be validated. Figure 4.33 exemplifies the annotation with constraints and a methods for invariants.

As mentioned, invariants lead to an additional method implemented in the concerned class rather constraint implementations directly reside in a validator class of

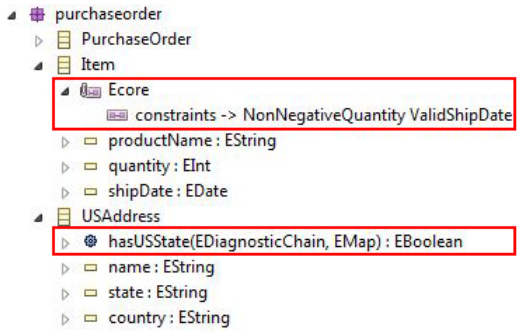


Figure 4.33: *EMF* – Core model with constraints and invariants

the `util`-subpackage of the generated code. Actually, only methods of this class are taken into account for validation using the validation function of the generated editor, for which reason methods for validating invariants additionally exist in the validator class, referencing the actual implementation in the host class. Generally, these generated methods require a manual intervention for specifying the conditions in Java code, because ECORE provides no way to represent them in the model. Besides the generation of modeled invariants and constraints, the code generation facility automatically adds basic constraints, such as for multiplicities, restrictions of the defined data type, the containment of all cross-referenced objects and so forth. The automatic introduction of such basic constraints is even performed for properties, which do not have manually defined constraints or invariants, but overall a preceding manual definition of a constraint or invariant in the core model is a prerequisite to activate their generation. EMF's validation framework includes a helper class called DIAGNOSTICIAN that is the recommended entry point for validation accordint to Budinsky et al. in [Bu09b]. DIAGNOSTICIAN provides methods to verify whether a given value violates any constraint defined for a type, e.g. the DIAGNOSTICIAN class can be used to verify a value before actually modifying an attribute of a class.

In addition to the described kind of *batch validators*, that are used for static validation of a selection of elements, usually in response to a user action, such as the manually invocation of the validation functionality in the model instance editor, the *EMF* validation framework also provides *live validators*. The validation of object models is generally performed by invocation of the *validation service*, that provides the two kinds of validators. A live validator obtains a collection of notifications representing discrete changes to model elements as input, that come from notifiers observed in the model. Along with the notification, needed information, such as the modified structural feature, the old value and the new value, becomes available to the constraint implementation.

For defining which constraints apply to which objects, *EMF* is able to describe a set of objects as a *client context*, which can be bound to constraints that are supposed to be enforced on these objects. Constraints can be bundled in categories and so the binding of constraints can take place by category, individually or in a mixture thereof. Furhtermore, *validation listeners* can be defined for the validation service of the *EMF* validation framework. Validation listeners are able to be associated with multiple client contexts and are notified by the validation service whenever validation in the associated contexts occurs.

The *EMF* validation framework provides support for parsing the content of constraint elements defined in specific languages. The validation framework provides support for two languages: Java and OCL. The framework provides an implementation of an XML constraint parser API that supports definition of OCL constraints in XML form. The only supported constraints for modeling with *EMF* are the aforementioned annotations for constraints and methods for invariants. Though, the actual conditions thereof cannot be specified by modeling. Even though the *EMF* validation framework has much more functionality beyond that, the evaluation of EMF is primarily confined to the constraint functionality that is provided for modeling, at most referring to further solutions.

4.5.2 EMF – Hierarchy modeling

The information model fragment composing a hierarchy of business process in Figure 3.1 can be modeled using the graphical modeling editor of the Ecore Tools as shown in Figure 4.34 or using the tree-based standard editor as shown in Figure 4.35. Thereby, the tree-based editor provides a bit more detailed configuration possiblities, e.g. the type-definition for map entries cannot be modified in the Ecore Tools editor.

The relationship of subordination is modeled by a bidirectional containment relationship, that has the intrinsic nature of creating an acyclic relationship, when establishing a self-relationship. That is owed to the fact that *EMF* interprets containment relationships as nested objects, in doing so the nested object can be compared to a kind of physical part of the containing object, that is thus only possible by one containing object, that in turn cannot be part of its constituents. Hence, the acyclic or hierarchic relationship is realized by the intrinsic nature of containment relationships in *EMF*. Based on this hierarchy, the ordering relationship is annotated by the constraint LinearOrder (cf. Figure 4.35), the resulting method in the generated Java code of which has to be complemented by the corresponding conditions.

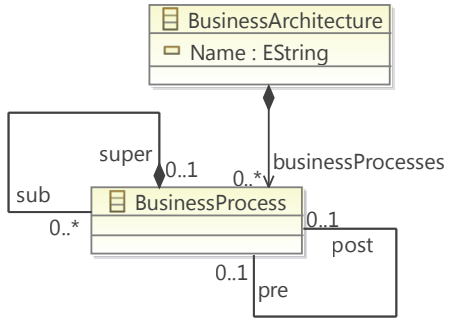


Figure 4.34: *EMF* – BusinessProcess core model

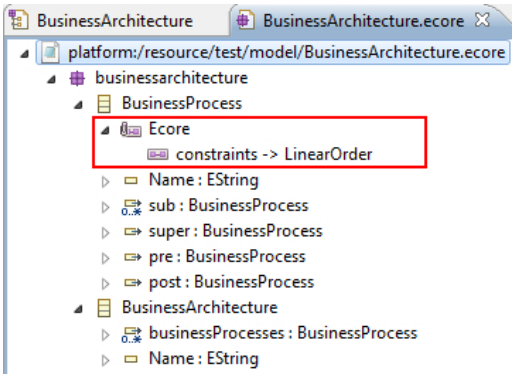


Figure 4.35: *EMF* – BusinessProcess core model in tree view

The need of introducing the additional class BusinessArchitecture becomes obvious, when launching the generated editor for instantiating the core model. The serialization of each instance of the model has to be well formed, that is to say a root element for all created objects is required. This root element is an object of BusinessArchitecure, because otherwise an object of BusinessProcess would have to compose the root object, which would prevent the modeling of more than one business process at the highest hierarchy level. Moreover, the automatically generated editor seems not to be able to handle containment self-relationships over multiple hierarchy levels, but actually this editor is only meant to be an initial point for refinement. Whether the lack of support of displaying transitively associated objects might be subject to the utilized query functionality of *EMF* is evaluated in Section 4.5.12. Thus, a business architecture makes up the container for a core model instance, which alternatively might also be accomplished by an ultimate business process, introduced as hierarchy root node.

EMF natively provides one kind of part-whole relationships by the containment reference, which can be both directional and bidirectional and introduces a kind of part-whole relationships reflecting a nature, which is expected for material compositions. Other directional or bidirectional relationships can be modified by annotating constraints, that can be complemented using the Turing complete programming language Java. Delving deeper into the *EMF* validation framework offers much more possibilities to introduce constraints and to listen on their validation results, but only for professional users of *EMF*.

In conclusion, *EMF* natively provides a kind of relationship for modeling hierarchies, which also reflect the ontological nature thereof. Furthermore, constraints can directly be annotated in the information model, the conditions of which have to be

complemented by Java code. The specified multiplicities are directly taken into account by the generated editor, resulting in a restricted possibility of defining related objects, e.g. the ordering relationship only allows to select a single object for preceding and succeeding business processes and the corresponding counterpart of the relationship end is automatically set or exchanged according to an object selection. In combination with a convenient graphical modeling editor, the need of complementing prepared Java code is not regarded as an impairment of the tool handling and so the evaluation of Table 4.27 is inferred from.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	●	●

Table 4.27: *EMF* – Evaluation of hierarchy modeling

4.5.3 EMF – Temporal and variant modeling

Temporal dependency of objects is supposed to by introduced by a despersive type TEMPORAL, which introduces a period of validity to types intended therefor. Similarly, a temporal dependency for relationships is required in order to achieve the tracking of changes thereof. Since relationships are realized by the corresponding end points in *EMF* and therefore do not make up a first-level concept, a relationship as a whole cannot have any properties or information assigned to. Hence, for evaluating the modeling building block ORGANIZATIONALUNIT-HOSTS-BUSINESSAPPLICATION, the information model fragment in Figure 3.4 is introduced to *EMF*, that constitutes a time-dependent relationship between business applications and organizational units using a mediating type to reify the relationship. Figure 4.36 shows the model fragment realized in the Ecore Tools editor.

EMF supports multiple-inheritance and is able to assign attributes to interfaces, which is also implemented by the generated Java code, despite of deviating from standard Java implementations, as described in Section 4.5.1.2. In this vein, a kind of mixins can be realized by *EMF* using interfaces. Hence, the mixin TEMPORAL of the model fragment in Figure 3.4 is modeled by an interface, which introduces a period of validity to each inheriting type, as e.g. illustrated for APPLICATIONHOST in Figure 4.36. As alluded to above in Section 4.5.1.2, this approach is applicable even to types that are already in an inheritance-relationship, since *EMF* re-implements the properties of the mixin interfaces during the code generation directly inside the base

implemetation class. Consequently, the interface TEMPORAL constitutes a reusable modeling building block that can be assigned to all types or instances of ECLASS in the information model, respectively, and may easily be extended to the bitemporal version thereof. The necessary constraint, which ensures the hosting by exactly one organizational unit at each point in time, is annotated to BUSINESSAPPLICATION and complemented with appropriate conditions in the corresponding validator class after having generated the implementation code.

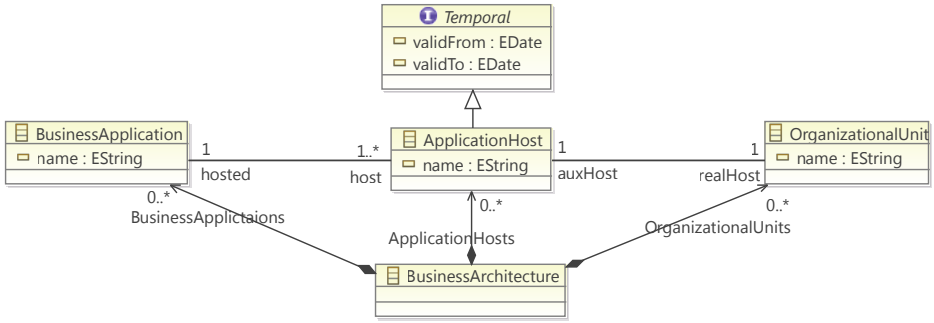


Figure 4.36: *EMF* – Temporal association pattern applied in I-Pattern I-24

Besides the already introduced kinds of models used by *EMF*, namely ECORE, core model and generator model, another model type is provided, the change model. The change model serves the purpose of representing changes or deltas to an objects instance of the core model and can be used to apply predefined changes or to record change as they are made. The record functionality of the change model may be regarded as a kind of change history, that additionally provides roll-back functionality to previous states. In this vein, the change model makes up an alternative for modeling a second temporal dimension besides a period of validity.

Even though the model fragment of Figure 4.36 structures the required information for temporal modeling and ensures necessary constraints in an partially reusable form, a denotation of relationships similar to the stereotype `«temporal»` is not achieved yet and cannot be achieved by modeling either. Professional user might define a client context for the concerned references and assign these to the corresponding constraints or categories of constraints. The listening to validation events might be used to ensure the constraints directly in the modeling editor.

The modeling of different variants of the EA or in the outlined example of the building block ORGANIZATIONALUNIT-HOSTS-BUSINESSAPPLICAITON is another important aspects, that is primarily served in Section 4.5.7, but also multiple instances of the core model can be created to model variants of the EA that may be structured by different folders in the file system, as depicted in Figure 4.37. Nevertheless, this kind

of variant modeling can only act as a makeshift during the introduction of an actual variant modeling, since there are no connections between the object models and also naming problems may occur for instances residing in the same resource, which would notably impairs the usability.

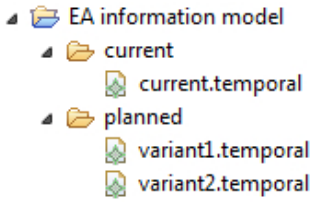


Figure 4.37: EMF – Variant modeling by multiple model instances

The EMFT project includes a further component, namely the *Temporality* feature, that provides an automatic versioning of model instances (cf. [Ec10a]). Since the temporality component is yet in its incubation phase and no downloadable version is already available, only a short summary of available information to complement this scenario by potential capabilities of temporality is given, but without taking any effect on the evaluation. The temporality feature consists of an core model containing a class TEMPORAL that has to be inherited by each type that temporality is supposed to be introduced to. Thereby, two time dimensions are introduced to the inheriting type, that is the *actual time* and the *record time*. Transferred to the aforementioned context, the actual time constitutes the start of the period of validity, whereas the its end has to be derived from the start time of other objects in the instance model. The record time is directly comparable to the second dimension introduced by bitemporality. Which attributes and references are supposed to be kept a history of, can be specified by adding an annotation thereto. In this way, it is kept track of the different values of attributes or the different objects a reference points to over time. The assignment of TEMPORALITY causes that the "current date" of a view in the editor becomes configurable, subject to which currently valid instances are displayed. Thus, the temporality feature of the EMFT project will enable a fairly comfortable definition of temporality for attributes and references of a type.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	◑

Table 4.28: *EMF* – Evaluation of temporal and variant modeling

Finally, the information needed to be structured in this scenario can be modeled by *EMF*, but only partially reflecting the ontological nature of temporal modeling, such as for temporal relationships. Table 4.28 summarizes the evaluation of temporal and variant modeling.

4.5.4 EMF – Non-rigid typing and principle of identity

EMF provides two specialization options for EClass indicated by the boolean properties ABSTRACT and INTERFACE. An abstract class represents an EClass, from which other classes can inherit features, but which cannot itself be instantiated. If interface is additionally `true`, the EClass represents an interface, that declares its properties, operations and relationships but cannot provide an implementation for them. Since an interface has always to be abstract, it cannot be instantiated and serves the purpose of supplying features and relationships to the inheriting classes. The implementation of an interface is modeled by an inheritance relationship to a non-interface EClass. (cf. [Bu09b])

In this vein, interfaces can be used to realize a dispersive type in *EMF*, since their implementation assigns an additional type to the implementing class, which also inherits all features of the interface. Furthermore, an interface does not supply a principle of identity, because interfaces cannot directly be instantiated and the indirect instances by implementation depend on the implementing type. Thus, the implementation of an interface to a non-interface class in *EMF* is regarded equivalent to the assignment of mixin to a type, as described in Section 3.1.3. Abstract classes can additionally provide implementations for features and cannot be instantiated similar to interfaces. Theoretically, a semantically richer version of mixins could be established thereby, but during the code generation only a single inherited abstract class can be considered as an extended class and the others are automatically converted into mixin interfaces (cf. Section 4.5.1.2).

However, *EMF* supports the distinction whether a types supplies a principle of identity, thus enabling the distinct modeling of rigid sortal universals and rigid mixin universals in line with Guizzardi in [Gu05]. Even though such a principle of identity is supported there are no type changes or type extensions possible for instance objects, as required e.g. for changing the presently valid life-cycle phase or to assign a role that is taken due to the changed context of an object. In this vein, *EMF* does not support non-rigid typing at all, asking for alternatives to structure the required information.

How to realize a workaround for life-cycles and different life-cycle phases is described in Section 4.5.6, dealing in-depth with this topic. Roles of a specific object can only be derived of the actual relationships thereof, e.g. a employee can be perceived as a project member once a membership is established for this employee, provided that this relationship has previously been modeled in the information model. Summarizing, *EMF* provides sufficient support for the distinction of identity supplying types and others, but lacks in non-rigid types. The evaluation of non-rigid typing and principle of identity are merged in Table 4.29.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	◑	◑

Table 4.29: *EMF* – Evaluation of non-rigid typing and principle of identity

4.5.5 EMF – Multi-level modeling

Core models or information models are modeled by instantiating the concepts provided by Ecore. Subsequently, models of concrete objects instantiate the information model types. Thus, *EMF* provides the typical two modeling levels of object-oriented modeling approachs. Since Ecore is a self-defining meta-model, it composes its own meta-model, meta-meta-model and so on, but these modeling levels are predefined and unchangeable, for which reason only the two levels of the object-oriented programming paradigm remain for modeling.

As described in Section 3.1.4, the ontologically correct realization of model fragments resorting to design patterns, such as the type-item pattern or the type-square pattern (cf. [YJ02]), asks for modeling in more than the typical two modeling levels in order to prevent a mismatch between ontological levels and modeling levels. Such issues cannot be addressed with the natively provided functionalities of *EMF*, because only the two "classic" object-oriented modeling levels are provided and Ecore, *EMF*'s meta-model cannot be modified, as well.

Although a structuring of the required information can be achieved by workarounds using patterns that twist multiple ontological levels into a single modeling layer, the fulfillment of this scenario is regarded as lacking in support. That is reasoned in the claim of evaluating multi-level modeling and not whether or not well known workarounds can be achieved, as done in scenarios evaluating concrete applications

thereof. Hence, the evaluation results in an overall lack of support of the requirements, as depicted in Table 4.30, which is also the reason the tool handling is not evaluated.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◯	◯	n.a.

Table 4.30: *EMF* – Evaluation of non-rigid typing and principle of identity

4.5.6 EMF – Life-cycle

As mentioned in Section 3.1.3 *EMF* does not support non-rigid types, for which reason an alternative modeling of phased sortals is required. An interface LIFECYCLED could be used for assigning the general term of being life-cycled to modeling types and could enforce the implementation of general behavior in the inheriting type, such as methods for life-cycle phase transitions. The different life-cycle phases could be introduced by modeling each phase by a discrete subtype of a common, abstract, life-cycled supertype, as depicted at the the example of BUSINESSAPPLICATION in Figure 4.38.

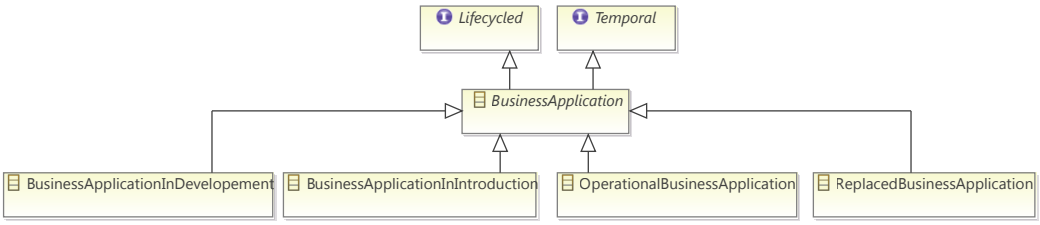


Figure 4.38: EMF – Life-cycle phasaes of a business application

For determining at which time a life-cycle phase is valid for a specific business application, the interface TEMPORAL is assigned to the type BUSINESSAPPLICATION that in turn supplies the properties, stating the period of validity, to its subtypes. Thus, the validity of a life-cycle phase can be determined, but a relation to a common identity is yet needed to identify several instances of BUSINESSAPPLICATION subtypes as partition of life-cycle phases belonging to a specific business application. There is a variety of possible solutions therefor, such as a simple attribute stating the common identity or in case of a non-abstract supertype BUSINESSAPPLICATION, a relationship to one common instance thereof. However the relation to a common

identity is modeled, on basis of the subtypes of BusinessApplication and the common identity, constraints can ensure that a business application always resides in exactly one life-cycle phase and that the life-cycle is permanently documented asking further for a controlled transition between life-cycle phases.

A controlled life-cycle phase transition can be achieved by resorting to projects and their work packages, the different types of which account for the transition between two life-cycle phases, as depicted in Figure 3.19. These extensions can be directly introduced to the information model fragment in Figure 4.38, since only supported modeling concepts are required therefor. In this way, a permanent documentation of the life-cycle and a controlled transition between the different phases thereof is established.

As outlined, the life-cycle modeling of a business application can be realized by the provided modeling concepts of *EMF*. The different life-cycle phases are modeled by discrete types that enable the modeling of various properties and relationships of a business application depending on the actual life-cycle phase. Besides the advantages coming along with life-cycle phases modeled in separate types, a type transition forces to migrate required information from the object representing the ending life-cycle phase to the object representing the beginning life-cycle phase at each life-cycle phase transition. If the common identity is modeled by an object, shared information may be contained by this common object, but the thought of transforming information along with the life-cycle phase without redundantly storing, cannot be carried through.

Nevertheless, the information of the scenario can be modeled by some workarounds, but the nature of switching between life-cycle phases without changing the identity, is not satisfactorily reflected from an ontological point of view. Complex constraints have to be manually developed to be able to assess the validity of object models asking for deep interference in the generated code, the effort of which results in an empty Harvey Ball for tool handling. The concluded evaluation findings are shown in Table 4.31.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	◑	○

Table 4.31: *EMF* – Evaluation of life-cycle

4.5.7 EMF – Projects

In *EMF* elements can be denoted as "project-affectable" or in other terms as potentially able to be affected by a project using the interface AFFECTABLE, which establishes a relationship between the type PROJECT and all inheriting subtypes of AFFECTABLE, as depicted in Figure 4.39. Due to the specific interpretation of interfaces by *EMF* during the code generation, interfaces can be used for realizing the modeling building PROJECTS-AFFECTS-AFFECTABLE (cf. Figure 3.21) in a reusable way and can thus denote each type in the EA information model as "project-affectable".

Figure 4.39: *EMF* – Modeling building block PROJECTS-AFFECTS-AFFECTABLE

In *EMF's* understanding, a relationships is composed by either a single reference that points to another type of the information model or by two references correspondingly pointing to the type of the opponent reference. Thus, directional and bidirectional relationships are realized in *EMF* by modeling their end points. In line with this understanding, relationships as a whole do not make up a fist level modeling concept in *EMF* rather are derived from refernces of information model types. Consequentially, a reification of different types of the AFFECTS-relationship as well as associated relator hierarchies (cf. Figure 3.22) cannot natively be realized in *EMF* and have to be modeled as discrete references without inheriting common properties of a super-relator.

Projects as major means for affecting the EA ask for a time period at which projects are executed and valid, respectively. For that, the interface TEMPORAL can be assigned to the type PROJECT that introduces a period of validity. This extension of PROJECT enables to determine when an effect takes place and to model competing project plans affecting the same architecture elements at the same time in future, that have to be decided upon at an appropriate point in time.

Transferring this to the example of the modeling building block ORGANIZATIONALUNIT-HOSTSBUSINESSAPPLICATION and its refinements elaborated over several scenarios of Chapter 3, the combination of time-dependent projects, life-cycled business applications (cf. Section 4.5.6) as well as a time-dependent hosts-relationship (cf. Sec-

tion 4.5.3) enable variant modeling of this modeling building block in a way comparable to the possible solution in Figure 3.25.

In conclusion, the information modeling of this scenario is sufficiently achieved by resorting to solutions already devised in the preceding sections of *EMF*'s evaluation. The primary qualities of projects, such as affecting other elements, scheduled by the time-dependency of projects, are reflected by the modeling solutions in an ontological correct way. Therefore, the ontological character of projects is regarded satisfactorily fulfilled, even though some of the used solutions of other scenarios do not similarly well reflect ontological qualities. Since potentially shortcomings are already evaluated, they are not incorporated a second time, as depicted in Table 4.32.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	●	◑

Table 4.32: *EMF* – Evaluation of projects

4.5.8 EMF – Standardization

Modeling of standardization demands the denotation of both architecture elements that are able to conform to a standard and elements that are able to constitute a standard, as illustrated by the modeling building block Standard-Standatdizable in Figure 3.27. In *EMF*, the mixins Standard and Standardizable can be realized by the corresponding interfaces that introduce the conformsTo- and the allowedFor-relationship among each other, as depicted in Figure 4.40. These interfaces can arbitrarily be assigned to every information model type as far as required for documenting and planning the situation of the EA's standardization.

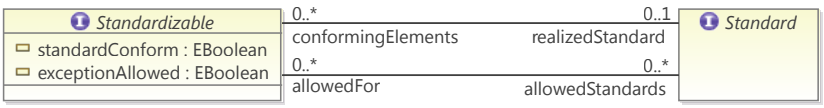


Figure 4.40: EMF – Modeling building block Standard-Standardizable

The derived attributes of the modeling building block Standard-Standardizable can directly be realized by declaring these properties volatile, transient and non-changeable, as described in Section 4.5.1.2. Thus, the simple modeling building

block enables the definition of standards and standardizable elements, as well as the relationships among them. As described in Section 3.2.3, standardization has to consider aspects, such as the distinction between deliberately and accidentally non-conformance to a standard by architecture element that are able to conform to a standard. For the ontologically correct distinction of the two states outcoming from this considerations, a non-rigid mixin type specializing the mixin STANDARDIZABLE is necessary, as devised in Section 3.2.3. Since non-rigid types are not supported by *EMF*, this proposed solution cannot be modeled, but alternatively aspects of unset values that are comparable to this issue and natively supported by *EMF* can be utilized here, as describe later in this section.

The formal relationship (cf. [Gu05]) CONFORMSTO can be realized similar to the derived attributes, by declaring the references thereof volatile, transient and non-changeable. Thus, the get-methods of the corresponding references has to be complemented in the implementation of types inheriting the interfaces STANDARD or STANDARDIZABLE. This context specific implementation of the derived relationship CONFORMSTO makes it possible to calculate their references in dependence of the actual composition or other context-specific relationships of the containing type. In this vein, *EMF* supports a kind of formal relationships that are grounded in the intrinsic qualities or in other terms the already defined attributes and references of their establishing types.

One of the different specializations of features can also be used for determining whether or not there is deliberately no conforming standard, since features can additionally be defined unsettable. Setting this property for the CONFORMSTO-relationship, enables the definition of a state different to `null` that is conceived as unset, in this way introducing a standard value for unset references. Thus, a lack of documentation can be determined by means natively provides by *EMF*, but the state-change of the assigned general term to be standardizable is only partially.

Fulfillment of scenario	**Ontological correctness of models**	**Tool handling**
●	◑	●

Table 4.33: *EMF* – Evaluation of standardization

Altogether, the non-rigid character of the qulatity of being standardizable as expressed by the two non-rigid subtypes of the mixin STANDARDIZABLE cannot satisfactorily be modeled in *EMF*. But besides this missing modeling concept, the scenario

is completely fulfilled by interfaces and the different kinds of feature provided by *EMF* that appropriately reflect the ontological nature of these constituents of standardization. Furthermore, the modeling of the provided concepts is fairly conveniently achieved, resulting in the evaluation in Table 4.33.

4.5.9 EMF – Goals

The *Goal-Question-Metric*-approach proposes to operationalize goals by a set of questions, which are assigned to architecture elements in order to attach the corresponding goal thereto and to measure its achievement by calculating the metrics of each question (cf. Section 3.2.4). An exemplary modeling solution of this approach is realized by introducing an additional ontological level for meta-types, because an operationalization of goals asks for metrics that have to be adjusted to the specific context a question is supposed to be measured. Hence, the actual assigned question on type-level is an instance of a question on meta-type-level complemented with appropriate metrics. Furthermore, questions are modeled by mixins, which add the required metrics directly to the types, the fulfillment of goals of which is supposed to be measured.

As alluded to in Section 4.5.5, *EMF* does not support multi-level modeling, for which reason the two-level nature of the information modeling building block Goal-Question-Metric has to be modeled by design patterns, such as the type-item pattern or the type-square pattern, folding multiple ontological levels into a single modeling level. The modeling of the mixin for questions can be realized by discrete interfaces for each required question by *EMF*.

The required metrics as well as the corresponding questions and goals can be modeled by *EMF*, but the actual character of questions, to be complemented with context-specific metrics by instantiation cannot sufficiently expressed by the modeling concepts provided by *EMF*. Furthermore, the need of introducing complex model structures complicates the realization of the scenario, which is reflected in an impaired tools handling. Table 4.34 summarizes the evaluation of goal modeling.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	◑

Table 4.34: *EMF* – Evaluation of goals

4.5.10 EMF – Role-based access control

EMF natively provides no functionality to control the access to any data of the information model or persisted instances thereof and so there are natively no types to document access rights for different users of the application contained in the information model. Access right information can be structured by introducing an interface ACCESSIBLE that establishes different relationships between instances of the inheriting types and users that are allowed to access these objects, while distinguishing read and write access, as depicted in Figure 3.36. A subsumption of users by roles using role-mixins cannot be modeled in *EMF*, since no modeling concepts similar to the non-rigid types role-mixin and role are provided. The required information can be modeled by introducing types for users and user-groups, that are related to the interface ACCESSIBLE for defining their access rights that can be realized either by directly relating the two types or indirectly via a common supertype. In whichever way the relationship between the interface ACCESSIBLE and the types USER or USERGROUP is established, user-groups bundle a set of access right definitions that are supplied to members of a group, which aggregate the access rights supplied by different user-group memberships. The aggregation of access right definitions supplied by user-groups to their members realizes a kind of role-based access right policy.

The mentioned distinction between read and write access in the preceding paragraph serves the purpose of exemplifying a possible modeling solution, but certainly these access rights can be structured in more detail as needed for a specific application. Even an extension to access right definitions on attribute level is conceivable by manually extending the get-methods and set-methods along with finer grained access right definitions, which enable such detailed distinctions.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	○

Table 4.35: *ADOxx* – Evaluation of role-based access control

In this vein, a role-based access right policy can be modeled in *EMF* reflecting the quality of being accessible and controlling this accessibility by defining success rights. The aggregation of success rights as needed for a role-based access right policy can only cannot be derived of the users' roles, taken due to their external dependencies to other elements and therefore requires an additional type for structuring such information. Thus, the required information is structured, but the actual control of

access is not yet achieved and has to be manually developed based on the outlined access right information modeling building block, for which reason the tool handling evaluation states a lack of support, as shown in Table 4.35.

4.5.11 EMF – Responsibilities

Responsibilities relate manageable architecture elements to the persons that bear responsibilities therefor. In Section 3.2.5 the modeling of this information is proposed by a mixin MANAGEABLE and a role-mixin RESPONSIBLE, as depicted by the modeling building block RESPONSIBILITIES in Figure 3.35. In *EMF* this modeling building block can be realized by an interface MANAGEABLE that establishes a relationship to the user-groups of responsible people or people, which are solely responsible, respectively. User-groups resort to those ones described in the preceding Section 4.5.10. Depending on the concrete definition of responsibilities, different kinds thereof can be distinguished, e.g. multiple relationships between the interface ACCESSIBLE and responsible people or the membership of specific user-groups might be utilized for determining the specific responsibilities of people.

People are supposed to have complete access rights to the architecture elements, they are responsible for. After having introduced a role-based access control approach as outlined in Section 4.5.10, the modeling building block ACCESSABILITY (cf. 3.36) can be extended by the modeling building block RESPONSIBILITIES as illustrated in Figure 3.39. Thus, the aspects of responsibility are complemented with aspect of access control for ensuring complete access right for responsible people.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	◑	◑

Table 4.36: *ADOxx* – Evaluation of responsibilities

The scenario is realized by modeling the reference of a responsible person by a direct reference or a membership of a user-group, since a role (cf. [Gu05]) cannot be utilized in *EMF* for their derivation. Provided that a role-based access control as described in Section 4.5.10 exists, the interface MANAGEABLE can resort to this functionality by subtyping the interface ACCESSIBLE. Summing up, the required information can sufficiently be structured in *EMF*, but a derivation of responsibilities of existing roles

cannot be achieved, for which reason the ontological nature of responsibilities is only partially reflected, as shown in Table 4.36.

4.5.12 EMF – Queries

For specifying and executing queries against a set of information model elements and their contents, *EMF* provides the subproject *model query framework* that encompasses the main classes used when formulating and executing query statements. In accordance with [Ec10c], the *EMF* query framework provides two different classes of query statements, namely SELECT and UPDATE, while the UPDATE class extends the SELECT class. The SELECT class is used for querying without modifying data, whereas the UPDATE class serves for modifying the query result. Both of the statement classes return the query result based on an instance of the *FROM* class, representing the elements to search, and an instance of the *WHERE* class, which applies search conditions to the elements specified by the FROM object. The search scope of elements to be used in a query is defined by the interface IEOBJECTSOURCE.

For specifying conditions applied by the instances of the WHERE class, the *EMF* model query framework provides a variety of conditions that implement predicates on primitive data types, as well as a condition API intended specifically for working with model elements, that is EOBJECTS in *EMF* resources. EOBJECTCONDITION is the root of the condition hierarchy for EOBJECTS. EOBJECTCONDITION can be subclassed for defining customized predicates using regular Java code, but the framework also provides a wide range of condition classes covering the majority of *EMF*'s reflective API (cf. [Ec10c]).

According to [Ec10c], all objects contained in the collection specified by a FROM object are traversed recursively by the corresponding SELECT object until the leaves of their containment subtrees are reached to find the matching objects. Furthermore, customized predicates defined in subclasses of EOBJECTCONDITION can introduce conditions, which assess transitive references to objects in the search scope of elements by traversing their relationships, since Java as a Turing complete programming language is used therefor. Since the information model structure can be retrieved using the EPACKAGE class, the information thereof can be utilized for the construction of conditions. At least by using manually developed subtypes of EOBJECTCONDITION transitive queries or the determination of the transitive closure may be achieved in *EMF*, whereby complex impact analyses on the repository data can be conducted.

The model query framework of *EMF* provides comprehensive possibilities to use the predefined functionalities or to introduce new or extended versions thereof. The short summery in this section only partially touches on the entire functionality, but conveys an impression thereof. Technically, the scenario is thus sufficiently fulfilled, however the whole definition of queries takes place by coding, impairing the tool handling evaluation to an empty Harvey Ball, as depicted in Table 4.37.

Fulfillment of scenario	Ontological correctness of models	Tool handling
●	n.a.	○

Table 4.37: *ADOxx* – Evaluation of queries

4.5.13 EMF – Information model changes

The information model can freely be modeled using the modeling concepts provided by Ecore. Hence, there are no restriction for introducing, changing or removing of types, properties, relationships and so forth in the information model. The information model is not dependent on existing instantiation data thereof rather the instances resort to the information model data, for which reason all described modeling actions can be performed irrespectively of the existence of any object model data. Using the modeling editor of the Ecore Tools, changes on the information model are fairly comfortable performed, e.g. attributes and methods can be shifted between different type by drag and drop.

On the one hand the ability to change the information model independent from the object models enables the evolution of the information model at any point in time, but on the other hand demands migration of existing data to the information model changes. Such a migration functionality is not provided by *EMF*, resulting in corrupted object data after having performed changes on the corresponding information model types. The migration of existing data is indeed not accomplished, but as well no object data gets lost by changing the information model, since the object models are serialized in XML Schema [Wo04], which only no longer fits to the changed information model. Such serializations are the starting-point for a manually data migration. However, the XML Schema serializations might be adapted by hand or by scripts adapting the performed changes of the information model, in order to achieve a serialization that suits to the changed information model. Irrespective of

which way of migration is chosen, there seems to be no way preventing a manually inference in the migration process.

In *EMF*, for each EStructuralFeature a default value as well as an upper or lower bound can be defined. The minimum number and maximum number of of allowed values are specified by the attributes lowerBound and upperBound, respectively, thus enabling the definition of mandatory properties and references by setting lowerBound greater than zero. The attribute defaultValueLiteral can be used for defining a default value, which is always a Sting value, irrespective of the definded data type. Default values can only be defined for EAttributes and so default value for EStructuralFeatures is always `null`.

In conclusion, *EMF* provides fairly flexible and convenient functionalities to create and change information models, but aspects of data migration are almost completely omitted. Although the Ecore Tools encompass a quite comfortable graphical editor, the tool handling cannot be evaluated as satisfactorily fulfilled due to the efforts required for migrating object data to a changed information model. Table 4.38 summarizes the evaluation of information model changes by *EMF*.

Fulfillment of scenario	Ontological correctness of models	Tool handling
◑	n.a.	◑

Table 4.38: *ADOxx* – Evaluation of information model changes

4.5.14 EMF – Summary of evaluation

EMF is an open source framework that is permanently refined, extended and enhanced by its community. *EMF* encompasses multiple subprojects and components that provides extensive functionalities in the fields of information modeling, code generation and related fields. Throughout the evaluation of *EMF*, primarily the most relevant parts of *EMF* are put under investigation for evaluating *EMF* against the scenarios devised in Chapter 3. Nevertheless, the evaluation meets its claim of providing a comprehensive impression of the EA information modeling capabilities of *EMF*, but as well, asks for the application in a "real-world" scenario at a practitioner to assess whether or not *EMF* lives up to its potentially promises.

EMF supports multiple inheritance by its meta-model Ecore and even provides a working implementation thereof based on its powerful code generation facility.

EMF enables to generally put the focal point on the information modeling, since a corresponding implementation as well as a simple editor for objects is generated at push of a button. Nevertheless, the evaluation was intended to evaluate the EA information modeling capabilities of *EMF*, and so additionally functionalities, such as the graphical editor or the code generation facility are primarily considered by the tool handling evaluation. The results of the different scenarios are summarized in Table 4.39 conveying an overview of *EMF*'s EA information modeling capabilities.

Scenario	Fulfillment of scenario	Ontological correctness of models	Tool handling
Hierarchy modeling	●	●	●
Temporal and variant modeling	●	◑	◑
Non-rigid typing and principle of identity	◑	◑	◑
Multi-level modeling	○	○	n.a.
Life-cycles	●	◑	○
Projects	●	●	◑
Standardization	●	◑	●
Goals	●	◑	◑
Responsibilities	●	◑	◑
Role-based access control	●	◑	○
Queries	●	n.a.	○
Information model changes	◑	n.a.	◑

Table 4.39: *EMF* – Summary of evaluation

4.6 Evaluation - Conclusion

The evaluated tools have overall proven to be able to structure the required information of the different scenarios. Though, multi-level modeling, as well as non-rigid typing and the principle of identity seem to be a specific hurdle under the requirements of the scenarios and so at least one of the tools revealed a lack of support for these scenarios. This deficiency of support is then directly reflected in the ontolog-

ical grade of scenarios resorting to the concepts of these scenarios, as just general architecture aspects call for additional concepts than those of the UML [OM10] to achieve their ontological correct modeling.

Generally speaking, the tools did not convey the impression of explicitly providing concepts to model in way reflecting the ontological nature thereof, for which reason the partial or almost complete fulfillments of the ontological correctness is regarded as an accidental bonus, coming along with the modeling concepts that are established for making the "technical" structuring of information as convenient as possible. This perception might be interpreted as another evidence for the importance of ontological correct modeling, since even providing concepts for a convenient information modeling, without being aware of ontological issues, obviously results in concepts reflecting their ontological nature to a certain extent.

Albeit the tools are comprehensively evaluated against the devised scenarios of Chapter 3, it is further to validate whether or not the evaluated expectations can be lived up in a "real" application at an enterprise. Furthermore, the results cannot be representative for all tools available on the market, asking for applying the gathered experiences of this evaluation on an extended set of tool, providing generic repository services for EA information model.

5 Conclusion and outlook

The goal of the thesis was to elicit requirements for EA information modeling and subsequently, to evaluate a set of tools providing generic enterprise model repository services on basis of these requirements. For eliciting requirements the focus was put on the EAM Pattern Catalog as strong knowledge base of EAM. Furthermore, it was to investigate, whether or not pure object-oriented modeling concepts, as provided by the UML [OM10], are sufficient for modeling the EA in a way reflecting the ontological nature of its constituents. After having introduced the lightweight stereotype approach of Guizzardi in [Gu05] that extends the UML by an typology of sortal universals or types, respectively, it became obvious, that at least such an approach is necessary to achieve an ontologically correct information modeling of the EA.

During the conduction of the thesis a set of scenarios has been devised that reflects a wide range of requirements for EA information modeling. However, the elaborated scenarios could be subsumed into three different kinds of scenarios, namely general architecture aspects, cross-cutting aspects and service aspects. Among these aspects the general architecture aspects turned out to the most fundamental one, exerting influence on the evaluation of most scenarios of other aspects. Thereby, the scenarios subsumed by general architecture aspects reflect a variety of important basics for EA information modeling, such as the structuring of the EA using different kinds of relationships and comprehensive constraint building blocks or the incorporation of temporal dependency in multiple dimensions. Furthermore, the need of multiple modeling levels is pointed out, in order to prevent unnecessary complexity caused by a mismatch of ontological levels and modeling levels. As well, the principle of identity and non-rigidity, subject to Guizzardi in [Gu05], are elucidated, resulting in the general distinction between types supplying a principle of identity, referred to as sortal universals, and dispersive types, which are conceived as a general term that can apply to multiple particulars, referred to as mixin universals. Non-rigidity introduces an animate character, since it demands the changeability of types along with the evolution of the EA. All of these concepts on their own approve the need of certain modeling concepts beyond pure UML, but as a whole, the need of a well-founded

meta-language based on an appropriate ontology, incorporating specific issues of EA information modeling becomes apparent.

Mostly, the evaluation of general architecture aspects anticipated an essential proportion of the evaluation of cross-cutting aspects, since the ontological correct modeling of cross-cutting aspects is fairly dependent thereon. Scenarios of cross-cutting aspects encompass some of the most important management subjects, such as the evolution of the EA, the management of standardization, traceability of management decisions, measuring the achievement of goals, and so forth, thus validating the importance of the general management aspects, that provide the foundation for their modeling. In this vein, the need of a well-founded meta-language is not only reasoned in modeling aspects, but can directly be derived from the management subject.

Scenarios belonging to service aspects complement the evaluation by assessing functionalities of the tools that are not concerned with information modeling capabilities, but the procedure of modeling and changing information as well as the management of the access thereto. Moreover, these concepts assess whether the provided services of the tool are overall integrated, e.g. whether responsibilities can be combined with the access control functionality or whether a defined validity can even be taken into account, when performing a query.

Modeling the ontological nature of scenarios was turned notably attention to over the conduction of this thesis, having caused, among others, a discrete evaluation criterion for the ontological correctness of models. But particularly the integration of an ontological foundation in EA information modeling can be regarded in its infancy, since primarily academia is concerned therewith and the tool evaluation conveyed a similar impression. Nevertheless, the necessity of such an integration is emphasized by almost every devised scenario. A sound basis to start the endeavor of an ontological well-founded EA information modeling can be found in the UFO elaborated by Guizzardi in [Gu05]. To start the endeavor is supposed to indicate that even Guizzardi's well-founded typology of universals seems to get exhausted in certain modeling situations of EA information modeling, as discovered in Section 3.2.3. In this section, the need of an additional non-rigid dispersive type, called non-rigid mixin, appears, which seemingly is not covered by Guizzardi's UFO. The need of a non-rigid mixin type was discovered and for the moment assumed as available in the suitable form, but a well-founded specification of this potentially new or extended dispersive type, has not taken place yet, asking for an in-depth investigation thereof.

As a matter of course, there may be aspects that are not yet covered by the scenarios, but the set of scenarios devised throughout the conduction of this thesis has never

claimed to be exhaustive. Notwithstanding, the scenarios as a whole are regarded as a sound basis for further investigations in the field of EA information modeling. The evaluation has demonstrated that aspects of ontologically correct modeling the EA seem not to rank among the highest prioritized issues of the tool vendors and accordingly, assessing the requirements has revealed shortcomings of the tools in supporting such aspects. The fact that for each of the tools one of the general architecture aspects had to be evaluated as an overall lack of support backs this perception. Consequently, the reasonability question of an even more comprehensive and intricate set of requirements inevitably arise, as long as the herein devised requirements are not yet completely fulfilled by the tools. In this vein, the applied intricacy of scenarios is regarded on the one hand as appropriate to reflect the most important requirements of EA information modeling, and on the other hand as appropriate to be evaluable by tools providing generic repository services.

Owed to the possible extent of a bachelor's thesis, only a small set of tools was evaluated, that cannot be representative for all tools providing generic enterprise model repository services available on the market. Nonetheless, the conducted evaluation gives a comprehensive example of how to apply the scenarios on both tools intended for creating an EAM function or comparable applications and tools primarily constituting a meta-modeling environment and are not directly intended for a usage in an EAM context. Even though most of the scenarios are at least proved by experiences of the EAM Pattern Catalog [se10b], the extensions made in this thesis have to be validated in concrete, real situations in enterprises. Thereby, the gathered experiences of actually applying the devised building blocks of the scenarios in practice may help to extend the knowledge base of the just evolving successor approach of the EAM Pattern Catalog, that is BEAMS [se10a].

Bibliography

[Ai08] Aier, S.; Kurpjuweit, S.; Riege, C.; Saat, J.: *Stakeholderorientierte Dokumentation und Analyse der Unternehmensarchitektur.* In (Hegering, H. G.; Lehmann, A.; Ohlbach, H. J.; Scheideler, C., Ed.): *38th GI Jahrestagung - Informatik 2008.* pages 559–565. Munich. 2008.

[AK07] Atkinson, C.; Kühne, T.: *Reducing accidental complexity in domain models.* In *Journal Software and Systems Modeling (SoSyM).* Volume 7, Number 3. pages 345–359. Springer. Berlin / Heidelberg. 2007.

[BCR94] Basili, V.; Caldiera, G.; Rombach, H.: *Goal Question Metric Approach.* In *Encyclopedia of Software Engineering.* pages 528–532. John Wiley & Sons, Inc. New York. 1994.

[BMN10] Büchner, T.; Matthes, F.; Neubert, C.: *Data Model Driven Implementation of Web Cooperation Systems with Tricia.* In *3rd International Conference on Objects and Databases (ICOODB).* Frankfurt am Main. 2010.

[BMS10a] Buckl, S.; Matthes, F.; Schweda, C. M.: *Conceptual Models for Cross-cutting Aspects in Enterprise Architecture Modeling.* In *Joint 5th International Workshop on Vocabularies, Ontologies, and Rules for the Enterprise (VORTE 2010).* Vittoria, Brazil. 2010.

[BMS10b] Buckl, S.; Matthes, F.; Schweda, C. M.: *A Meta-Language for EA Information Modeling - State-of-the-art and Requirements Elicitation.* In *15th International Conference on Exploring Modelling Methods in Systems Analysis and Design (EMMSAD 2010).* Hammamet. 2010.

[Bu07] Buckl, S.; Ernst, A.; Lankes, J.; Schneider, K.; Schweda, C. M.: *A Pattern based Approach for constructing Enterprise Architecture Management Information Models.* In *8. Internationale Tagung Wirtschaftsinformatik.* Karlsruhe. 2007.

[Bu08a] Buckl, S.; Ernst, A.; Lankes, J.; Matthes, F.; Schweda, C. M.: *Enterprise Architecture Management Patterns – Exemplifying the Approach.* In *The*

12th IEEE International EDOC Conference (EDOC 2008). Hammamet. 2008.

[Bu08b] Bundesamt für Sicherheit in der Informationstechnik: *BSI Standard 100-2 IT-Grundschutz Methodology*. https://www.bsi.bund.de/cae/servlet/contentblob/471430/publicationFile/27994/standard_100-2_e_pdf.pdf (cited 2010-06-05). 2008.

[Bu09a] Buckl, S.; Ernst, A.; Matthes, F.; Schweda, C. M.: *An Information Model for Managed Application Landscape Evolution*. In *Journal of Enterprise Architecture (JEA)*. 2009.

[Bu09b] Budinsky, F.; Steinberg, D.; Paternostro, M.; Merks, E.: *EMF - Eclipse Modeling Framework*. Addison-Wesley. Boston, MA. 2nd edition. 2009.

[CEF99] Carlson, A.; Estepp, S.; Fowler, M.: *Temporal Patterns*. In (Harrison, N.; Foote, B.; Rohnert, H., Ed.): *Pattern Languages of Program Design 4*. pages 241–261. Addison-Wesley. Boston, MA. 1999.

[Ec10a] Eclipse Foundation: *Eclipse.org*. http://www.eclipse.org (cited 2010-08-26). 2010.

[Ec10b] Eclipse Foundation: *EMF API Javadoc*. http://download.eclipse.org/modeling/emf/emf/javadoc/2.6.0/index.html?overview-summary.html (cited 2010-08-28). 2010.

[Ec10c] Eclipse Foundation: *EMF Model Query Developer Guide*. http://help.eclipse.org/helios/index.jsp?nav=/19 (cited 2010-09-03). 2010.

[Ga95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley. Reading, MA. 1995.

[Gu05] Guizzardi, G.: *Ontological foundations for structural conceptual models*. PhD thesis. University of Twente. The Netherlands. 2005.

[Gu06] Guizzardi, G.: *The Role of Foundational Ontology for Conceptual Modeling and Domain Ontology Representation*. In *Companion Paper for the Invited Keynote Speech, 7th International Baltic Conference on Databases and Information Systems*. Vilnius, Lithuania. 2006.

[Gu07] Guizzardi, G.: *On Ontology, ontologies, Conceptualizations, Modeling Languages and (Meta)Models*. In *Frontiers in Artificial Intelligence and Applications, Databases and Information Systems IV, Olegas Vasilecas, Johan*

Edler, Albertas Caplinskas (Editors), ISBN 978-1-58603-640-8, IOS Press. Amsterdam. 2007.

[GWS04] Guizzardi, G.; Wagner, G.; van Sinderen, M.: *A Formal Theory of Conceptual Modeling Universals.* In *Deutsches Forchungszentrum fur Kunstliche Intelligenz Report (ISSN 0946-008X), Proceedings of the First International Workshop on Philosophy and Informatics (WSPI).* Cologne, Germany. 2004.

[In10] InfoAsset AG: *InfoAsset Tricia - Open Source Web Collaboration and Knowledge Management Software.* http://www.infoasset.de/, (cited 2010-08-19). 2010.

[KA09] Kurpjuweit, S.; Aier, S.: *Ein allgemeiner Ansatz zur Ableitung von Abhängigkeitsanalysen auf Unternehmensarchitekturmodellen.* In *9. Internationale Tagung Wirtschaftsinformatik.* Vienna. 2009.

[Kr04] Krcmar, H.: *Informationsmanagement.* Springer. Berlin. 4th edition. 2004.

[Ma09] Matthes, F.; Buckl, S.; Leitel, J.; Schweda, C. M.: *Enterprise Architecture Management Tool Survey 2008.* In *ISIS Business Integration Special, Nomina Informations- und Marketing Services.* Munich. 2009.

[OM06a] OMG: *Meta Object Facility (MOF) Core Specification.* version 2.0 (formal/2006-01-01). 2006.

[OM06b] OMG: *Object constraint language (ocl) available specification.* version 2.0 (formal/2006-05-01). 2006.

[OM10] OMG: *OMG Unified Modeling LanguageTM (OMG UML), Infrastructure.* (formal/2010-05-03). 2010.

[se10a] sebis, Chair for Informatics 19, Technische Universität München: *Building Blocks for Enterprise Architecture Management Solutions.* http://wwwmatthes.in.tum.de/wikis/beams/home, (cited 2010-08-11). 2010.

[se10b] sebis, Chair for Informatics 19, Technische Universität München: *EAM pattern catalog wiki.* http://eampc-wiki.systemcartography.info, (cited 2010-08-11). 2010.

[se10c] sebis, Chair for Informatics 19, Technische Universität München: *Research Projects.* http://wwwmatthes.in.tum.de/wikis/sebis/projects, (cited 2010-08-11). 2010.

[Wo04] World Wide Web Consortium (W3C): *XML Schema Part 0: Primer Second Edition*. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ (cited 2010-08-26). 2004.

[Wo08] World Wide Web Consortium (W3C): *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126/ (cited 2010-08-26). 2008.

[YJ02] Yoder, J. W.; Johnson, R.: *The Adaptive Object Model Architectural Style*. In *Proceeding of The Working IEEE/IFIP Conference on Software Architecture 2002 (WICSA3 '02)*. Montreal. 2002.