Multi-Perspective Enterprise Modelling – Conceptual Foundation and Implementation with ADOxx

Alexander Bock and Ulrich Frank

Abstract This chapter describes a method for multi-perspective enterprise modelling (MEMO) and a prototypical implementation of a selected part of the method with ADOxx, called MEMO4ADO. MEMO has been developed during a period of more than twenty years and is still a subject of ongoing research. MEMO includes a set of integrated domain-specific modelling languages to describe organisational action systems as well as information systems. MEMO4ADO implements a subset of MEMO languages specifically tailored for educational purposes. The chapter summarizes the background and evolution of MEMO, illustrates the implementation and functionalities of MEMO4ADO, and outlines future developments.

1 Introduction

1.1 Motivation

Software is a linguistic artefact. On the one hand, this means that it is ultimately made of some form of machine language. On the other hand, it means that we can use it, i.e. *make sense of* it, only if it is supplemented with a linguistic representation that corresponds to a language with which prospective users are familiar. The better this correspondence, the more convenient it will be to use the software. Further, when it comes to the design of enterprise software, it is generally recommended to involve different stakeholders, such as managers, prospective users, and IT experts.

Alexander Bock
Research Group Information Systems and Enterprise Modelling, University of Duisburg-Essen, Essen, Germany. e-mail: alexander.bock@uni-due.de

Ulrich Frank
Research Group Information Systems and Enterprise Modelling, University of Duisburg-Essen, Essen, Germany. e-mail: ulrich.frank@uni-due.de

Analysing and designing enterprise software requires communicating with people who have different professional backgrounds, and who therefore speak different languages. Conceptual modelling has been advanced to help cope with some of these issues for long. A conceptual model in the traditional sense is an abstraction of a software system that represents the intended meaning of the system using concepts that are supposed to be known in the target domain. These domain-specific concepts, in turn, are conventionally built with generic modelling languages that consist of basic linguistic constructs (one could also call them ontological constructs) such as "entity type" or "attribute".

Building on the notion of conceptual modelling, the idea of enterprise modelling goes one step further. It emphasizes the need to model not only software systems but also the context in which these systems are sought to be deployed. This is for two reasons. First, enterprise software does not work autonomously. It has to be aligned with the operations of an enterprise and needs to account for the tasks it should support. Second, taking advantage of the potential benefits offered by IT will often require to reorganise the organisational action system, e.g., to redesign business processes, tasks, and maybe even the entire business model. As a consequence, the analysis and design of software systems and the surrounding organizational action system should preferably be done in conjunction. Enterprise models are intended to support this kind of conjoint analysis and design. A minimal enterprise model integrates at least one model of an enterprise software system (e.g., an object model) with at least one model of the related action system (e.g., a business process model). The integration of models is intended not only to foster a better understanding of dependencies between business and IT, but also to help avoid inconsistencies.

An enterprise model is usually, though not necessarily, presented in the form of graphical diagrams. While diagrams could be created manually, for economic reasons it is advisable to employ software-based modelling tools to develop and utilize enterprise models. Modelling tools promise to help protect the integrity of enterprise models, to enable navigation through multiple integrated models, and to improve the reuse of models. They may further enable various kinds of model analysis and transformations (e.g., model-based code generation). In sum, research on enterprise modelling needs to consider the construction of modelling tools alongside the development of enterprise modelling methods.

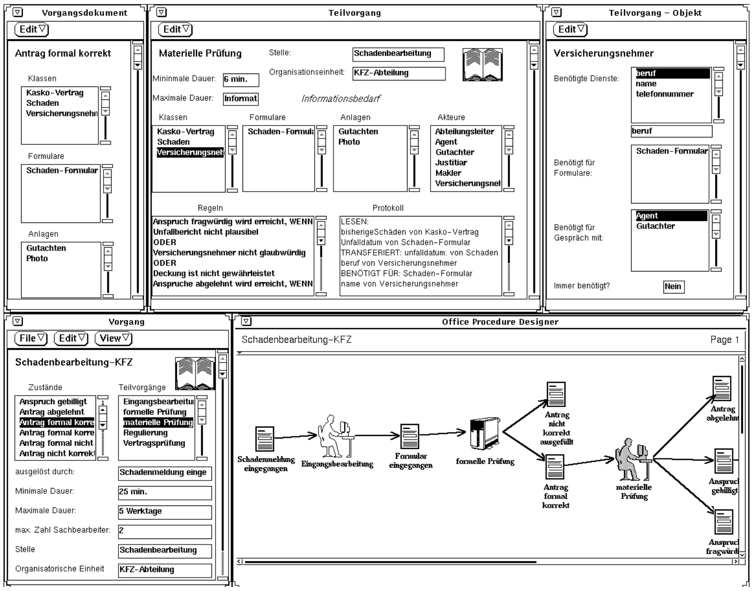
With this in mind, the present chapter has two purposes. First, it presents a method for multi-perspective enterprise modelling (MEMO) [9, 15]. Second, it describes a prototypical implementation of a selected part of the method with ADOxx [8]. The implemented tool is called MEMO4ADO. To begin, the next section summarizes historical developments which led to the method in its current state, considering both conceptual foundations and different tool development platforms. Section 2 provides an overview of the major components of MEMO. Section 3 illustrates particular modelling languages and ways of using MEMO. In section 4, the modelling tool MEMO4ADO is presented. The chapter concludes in section 5.

1.2 Historical Background

The development of the method presented here started 1989 in the German National Research Centre for Computer Science. The centre's chairman felt inspired by the vision of fully automated factories and decided to make it the subject of a project in the business informatics research group. The project's mission was to develop a conceptual foundation for promoting the level of automation in organisations. Very soon the group members agreed that integrative enterprise models would be required as a tool to support the joint reorganisation of an enterprise and the design of corresponding information systems. A conceptual high-level framework for multi-perspective enterprise modelling (MEMO) was created and further developed in a habilitation thesis [9]. The framework included modelling languages for various domains such as business processes, organisational structures, corporate strategies, and object models. Instead of specifying metamodels, the languages were directly implemented in model editors. The first integrated enterprise modelling environment was implemented in 1992 with Smalltalk. The screenshot in figure 1 shows an editor for process models, integrated with parts of role models, object models, and models of documents.

In the following years, research on the method continued at the university of Koblenz. A meta-modelling language (MEMO MML) was created [10] and subsequently used to specify metamodels of various languages. Based on this foundation, an entirely new version of the tool environment, called MEMO Center, was developed in 1997. Despite the undisputed benefits of Smalltalk, it was decided in 2003, then at the university of Duisburg-Essen, to use Eclipse as the development platform and Java as a programming language. This decision was based on two assumptions. First, the Smalltalk community seemed to shrink and Smalltalk environments were not developed much further. Second, we recognized the need for a meta-modelling component. For this purpose, the Eclipse Modeling Framework [34] and the Eclipse Graphical Modeling Framework [21] were chosen as a foundation. The existing metamodels were represented as Ecore instances. Later, the environment, which we then called MEMO NG ("next generation"), was supplemented with a meta-modelling facility that allowed the definition of metamodels with the MEMO MML. The metamodels were transparently transformed into Ecore instances and could be used, after some manual extensions, to generate model editors [22].

In parallel to the tool development, our research on modelling languages identified substantial challenges that concerned the representation of abstraction concepts. It turned out that these problems were related to principal limitations not only of our meta-modelling language and the OMG Meta Object Facility (MOF) [27] language architecture, but also of mainstream object-oriented programming languages. To overcome these obstacles, we decided for a radical change, both with respect to the language architecture and the implementation language. A new recursive language architecture has been designed and a corresponding modelling environment featuring a common representation of models and code is currently under development (see subsection 2.4).



Antrag formal korrekt

Klassen: Kasko-Vertrag
Schaden
Versicherungsnehm

Formulare: Schaden-Formular

Anlagen: Gutachten
Photo

Materielle Prüfung

Stelle: Schadenbearbeitung

Organisationseinheit: KFZ-Abteilung

Mininmale Dauer: 6 min.

Maximale Dauer: Informat

Informationsbedarf

Klassen: Kasko-Vertrag, Schaden, Versicherungsneh

Formulare: Schaden-Formula

Anlagen: Gutachten, Photo

Akteure: Abteilungsleiter, Agent, Gutachter, Justitiar, Makler, Versicherungsneh

Regeln:

Anspruch fragwürdig wird erreicht, WENN
Unfallbericht nicht plausibel
ODER
Versicherungsnehmer nicht glaubwürdig
ODER
Deckung ist nicht gewährleistet
Anspruche abgelehnt wird erreicht, WENN

Protokoll:

LESEN:
bisherigeSchäden von Kasko-Vertrag
Unfalldatum von Schaden-Formular
TRANSFERIERT: unfalldatum: von Schaden
beruf von Versicherungsnehmer
BENÖTIGT FÜR: Schaden-Formular
name von Versicherungsnehmer

Versicherungsnehmer

Benötigte Dienste: beruf, name, telefonnummer

beruf

Benötigt für Formulare: Schaden-Formular

Benötigt für Gespräch mit: Agent, Gutachter

Immer benötigt? Nein

Schadenbearbeitung-KFZ

Zustände: Anspruch gebilligt, Antrag abgelehnt, Antrag formal korre, Antrag formal korre, Antrag formal nicht, Antrag nicht korrekt

Teilvorgänge: Eingangsbearbeitu, formelle Prüfung, materielle Prüfung, Regulierung, Vertragsprüfung

ausgelöst durch: Schadenmeldung einge

Minimale Dauer: 25 min.

Maximale Dauer: 5 Werktage

max. Zahl Sachbearbeiter: 2

Stelle: Schadenbearbeitung

Organisatorische Einheit: KFZ-Abteilung

Schadenbearbeitung-KFZ

Fig. 1: Screenshot of the first MEMO modelling environment from 1992

2 Method Description

In essence, the enterprise modelling method MEMO extends the basic notion of an enterprise model with two additional aspects.

(1) *Emphasis on perspectives*: The notion of perspective is purposefully overloaded here. First, it emphasizes the need to account for users with different cognitive dispositions. This suggests to offer concepts and visualisations that correspond to specific professional interests and related language games. The second conception is related to the notion of a view. It concerns the idea that specific parts of an enterprise model are intended to represent, or to relate to, a certain cognitive perspective. The third conception is an additional "meta" perspective that demands to reflect upon the limits of enterprise models to avoid neglecting important aspects such as informal power relations, symbolic action, and organisational culture.

(2) *Use of domain-specific modelling languages (DSMLs)*: A DSML provides modellers with concepts that are reconstructed from the relevant domain of discourse. It promises to promote modelling productivity because it frees modellers from the need to reconstruct domain-specific concepts from basic linguistic constructs. It also promises to foster model quality because ideally the concepts of a DSML have been carefully developed by domain experts. Finally, domain-specific language specifications can include (domain-specific) constraints that prevent, to a certain extent, the construction of inappropriate models.

The construction of MEMO reflects these general considerations. MEMO comprises four major components. First, the *language architecture* includes the meta-modelling language and defines the relationship between models on different levels of classification. Second, an *extensible set of DSMLs* provides the basic instruments for users of MEMO. Third, a *method for designing DSMLs* guides the creation of new modelling languages and the modification of existing languages. Fourth, users are supported by various *modelling methods* that are essentially composed of DSMLs and corresponding process models. Advanced users who want to create their own methods are guided by a (meta) method for method construction. The following description (section 2.1-2.3) refers to the original language architecture of MEMO, as the new recursive language architecture, considered in section 2.4, cannot be represented within ADOxx.

2.1 Language Architecture and Meta-Modelling Language

The original MEMO language architecture corresponds to the three tier architecture that is also proposed by the OMG Meta Object Facility [27]. Figure 2 illustrates its basic structure. The meta-metamodel (see figure 3), which defines the abstract syntax and semantics of the meta modelling language MEMO MML, forms the linguistic foundation of the method. It is used to specify an extensible set of DSMLs through metamodels. The metamodels, and, as a consequence, models created with the respective DSMLs, are integrated through common meta concepts of the meta-metamodel and concepts they share directly. In other words, the different MEMO languages are integrated because each language includes (meta) relationships to concepts specified in other languages (for examples, see section 3.1), and these integrative relationships are possible only because all languages are defined by a common meta-modelling language. The language architecture thus constitutes the enabling basis for integration among MEMO DSMLs. Further, the meta-metamodel includes constraints that are specified with the OMG Object Constraint Language (OCL) [28]. To support the creation of modelling editors from metamodels, there is a mapping from metamodels to corresponding object models.

The latest version of the MEMO MML (see figure 3) for the three tier architecture was introduced in 2011 [13]. Some concepts of the MEMO MML are similar to concepts of other meta-modelling languages. This mainly concerns basic concepts to specify meta types with attributes and associations. In addition, the MEMO MML also features concepts which address advanced meta-modelling issues. These are explained in the following subsection. Lastly, the MML includes a graphical notation that enables the visual distinction between metamodels at level M_2 and object or data models at level M_1 (see the example in figure 4).

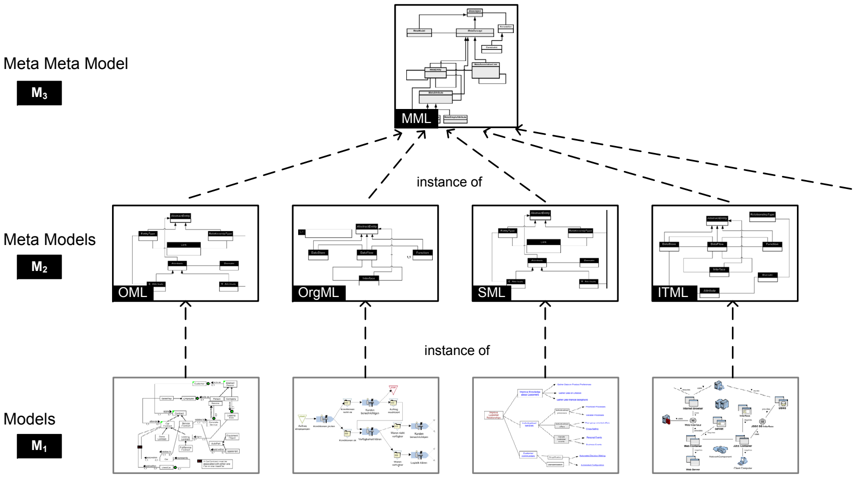


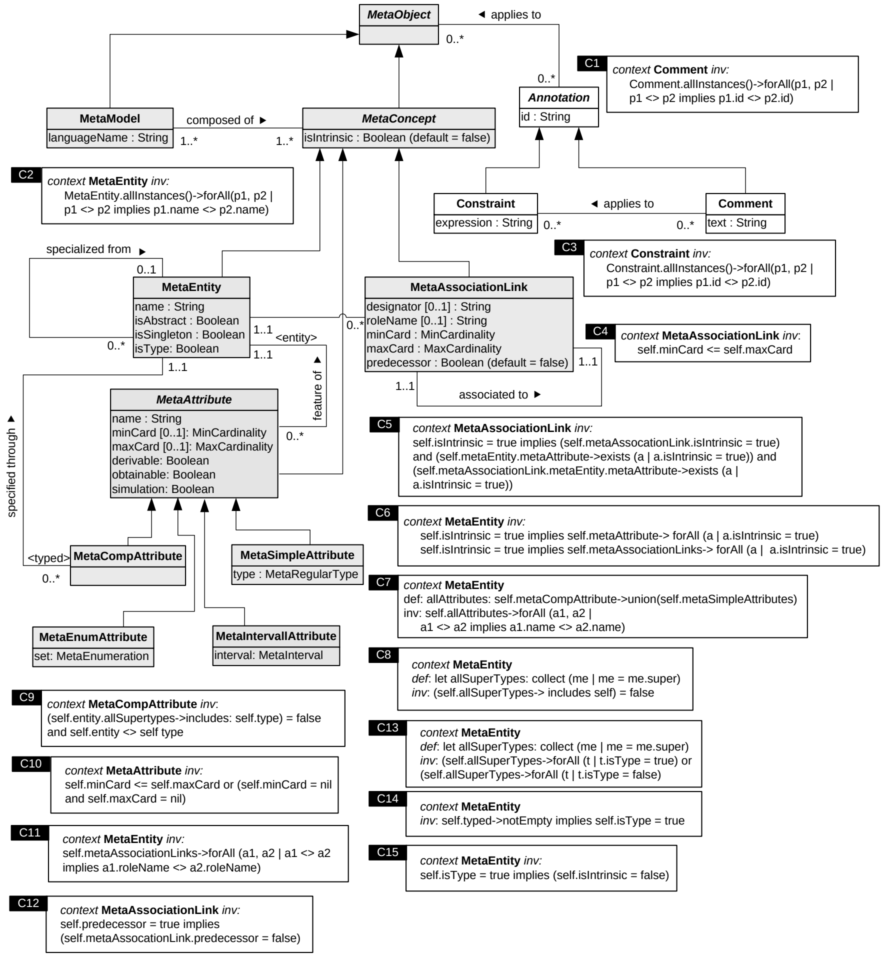
Fig. 2: MEMO language architecture

2.2 Advanced Meta-Modelling Concepts

While the clear separation of classification levels and the restriction to a fixed number of classification levels (up to M_3) together seem to provide a solid foundation for (meta) modelling, there are relevant cases that cannot be handled within such a language architecture. The MEMO MML accounts for these cases with two additional concepts, *intrinsic features* and *language level types*, which are explained below.

Intrinsic features. When we create a conceptual model, we aim at a certain level of abstraction. This means that we intentionally fade out aspects that may become relevant only at lower levels of abstractions. On the other hand, when we design a conceptual model, we would like to create a specification which is as comprehensive as possible at the chosen level of abstraction. This means that we would like to express everything we know at this level, even though it may apply to lower levels only. Failing to do so would prevent the reuse of existing knowledge and jeopardize the integrity of models at lower levels of classification.

The following example illustrates the problem. A language for modelling business processes (M_2) may include a concept “Process”, which could include an attribute such as “maxExecutionTime” that is instantiated at the type level (M_1) and serves to indicate a maximum execution time that applies to all process instances. In addition to that, we know that every process instance has a certain start and termination time. However, if we added an attribute such as “startTime” to the meta type “Process”, we would run into the problem that it would be instantiated at the type level (M_1). This would be wrong because the attribute clearly pertains to the instance level (M_0). The problem is known for some time ([2], [29]), and the only way to deal with it is to introduce additional concepts in the meta-modelling language. Accordingly, the MEMO MML provides “intrinsic features”. These can be



```
C1
context Comment inv:
    Comment.allInstances()->forAll(p1, p2 |
    p1 <> p2 implies p1.id <> p2.id)

C2
context MetaEntity inv:
    MetaEntity.allInstances()->forAll(p1, p2 |
    p1 <> p2 implies p1.name <> p2.name)

C3
context Constraint inv:
    Constraint.allInstances()->forAll(p1, p2 |
    p1 <> p2 implies p1.id <> p2.id)

C4
context MetaAssociationLink inv:
    self.minCard <= self.maxCard

C5
context MetaAssociationLink inv:
self.isIntrinsic = true implies (self.metaAssocationLink.isIntrinsic = true)
and (self.metaEntity.metaAttribute->exists (a | a.isIntrinsic = true)) and
(self.metaAssociationLink.metaEntity.metaAttribute->exists (a |
a.isIntrinsic = true))

C6
context MetaEntity inv:
    self.isIntrinsic = true implies self.metaAttribute-> forAll (a | a.isIntrinsic = true)
    self.isIntrinsic = true implies self.metaAssociationLinks-> forAll (a |  a.isIntrinsic = true)

C7
context MetaEntity
def: allAttributes: self.metaCompAttribute->union(self.metaSimpleAttributes)
inv: self.allAttributes->forAll (a1, a2 |
    a1 <> a2 implies a1.name <> a2.name)

C8
context MetaEntity
def: let allSuperTypes: collect (me | me = me.super)
inv: (self.allSuperTypes-> includes self) = false

C9
context MetaCompAttribute inv:
(self.entity.allSupertypes->includes: self.type) = false
and self.entity <> self type

C10
context MetaAttribute inv:
self.minCard <= self.maxCard or (self.minCard = nil
and self.maxCard = nil)

C11
context MetaEntity inv:
self.metaAssociationLinks->forAll (a1, a2 | a1 <> a2
implies a1.roleName <> a2.roleName)

C12
context MetaAssociationLink inv:
self.predecessor = true implies
(self.metaAssocationLink.predecessor = false)

C13
context MetaEntity
def: let allSuperTypes: collect (me | me = me.super)
inv: (self.allSuperTypes->forAll (t | t.isType = true) or
(self.allSuperTypes->forAll (t | t.isType = false)

C14
context MetaEntity
inv: self.typed->notEmpty implies self.isType = true

C15
context MetaEntity inv:
self.isType = true implies (self.isIntrinsic = false)
```

Fig. 3: MEMO meta-metamodel

used to declare meta types, attributes, and associations as ‘intrinsic’, which means that they are to be instantiated at level M_0 only. In the meta-metamodel, the concept “MetaConcept” includes the attribute “isIntrinsic”, which is inherited to “MetaEntity”, “MetaAssociationLink”, and “MetaAttribute” to allow entity types, attributes and associations to be marked as intrinsic. The example in figure 4 illustrates how intrinsic attributes are represented in metamodels (see the black and white ‘i’ symbol), and it also shows that they are not instantiated in classes at level M_1.

Language level types. A further problem caused by MOF-like architectures concerns the fact that objects of different classification levels may not co-exist at the same level of the architecture. For example, it is not possible that one particular model simultaneously contains objects from level M_1 and M_0. While this constraint

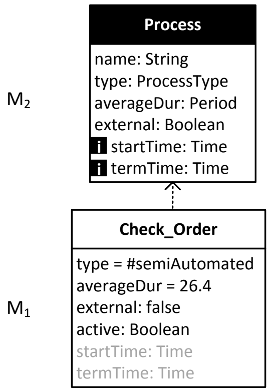


Fig. 4: Illustration of intrinsic features

is for a good reason, it can prevent the construction of perfectly useful models. Take, for example, a model of a logistic chain. A related DSML might include a concept such as “MeansOfTransport”, which could be instantiated into types such as “Truck” or “Boat”. Further, to model a logistic chain, it is essential to include locations. But modelling a location *type* such as “City” will usually not be satisfactory. Instead, a particular location, such as “Vienna”, will be needed. Different from other objects in a logistic model (which would be located at level M_1), however, a location should be represented as an instance at level M_0. But traditional MOF-like architectures do not permit to mix objects from level M_1 (e.g., “Truck”) and M_0 (e.g., “Vienna”) in one model. To overcome this limitation, MEMO allows to mark concepts in a metamodel (M_2) as representing *types* rather than meta types (using the attribute “isType”). As a result, these types will be instantiated into instances at the model level (M_1) already. For a corresponding example model, see [13, pp. 23–24].

2.3 A Method for Designing DSMLs

While MEMO is intended to cover relevant domains of an enterprise, it would be presumptuous to claim that the MEMO languages are sufficient in all cases. On the one hand, it may happen that a particular DSML needs to be modified. On the other hand, it may turn out that additional DSMLs are needed. A number of tools support the specification of DSMLs and the realization of corresponding model editors. But these tools provide little guidance for the conceptual design of a language for a specific purpose. This is even more problematic as many prospective users will not be familiar with the concept of a DSML, which makes requirements analysis for modelling languages especially challenging.

For these reasons, MEMO was supplemented with a method for designing DSMLs [14]. It features a macro-level process model that includes eight phases.

Each phase is structured by a specific micro-level process. To support requirements analysis, the method suggests the use of scenarios. A use scenario is characterized by a problem situation, related questions, and a specific technical language. Use scenarios can be identified based on relevant past decision and modelling scenarios. The method further advises to use preliminary diagrams to help prospective users understand what they can expect from a DSML. To get an idea what information could be represented in certain diagram types, it is suggested to start with a rudimentary graphical representation and then develop a list of questions related to the diagram. Analysing these questions can support the systematic identification of concepts to be included in the target DSML. Further guidelines relate to frequent design decisions to be made during the specification of metamodels (see [14]).

2.4 Next Generation

Even though intrinsic features and type-level concepts enable the construction of more expressive models, their representation in the meta-metamodel suffers from the problem that it is almost impossible to implement them satisfactorily with mainstream object-oriented programming languages. Furthermore, a language architecture that consists of two levels of classification only does not support the refinement, and, hence, the reuse, of DSMLs on higher levels. For example, a DSML concept such as “Printer” at level M_2 could be specified as an instance of “PeripheralDevice” at level M_3, which in turn could be part of a higher-level DSML. Finally, mainstream object-oriented programming languages require overloading the M_0 layer: Types or even meta-types are represented as objects at level M_0. As a result, it is necessary to maintain two distinct representations of models and code, which causes the notorious problem of synchronisation. The only way to overcome these obstacles is to abandon the traditional language architecture, both with respect to modelling languages and programming languages.

A few years ago we decided to pursue such an approach. It led to a radical modification of the MEMO language architecture. The MOF-like architecture was replaced by a recursive “golden braid” architecture that enables an arbitrary number of classification levels [16]. Furthermore, Eclipse and Java were replaced by XMF (eXecutable Metamodelling Facility), which includes a (meta) programming language that is also based on a recursive architecture ([4], [5]). By integrating the MEMO meta-metamodel with the meta-metamodel of XMF, both models and code share a common representation. This architecture enables to build enterprise systems that are integrated with conceptual models of themselves as well as models of the environment in which they operate [19]. Such “self-referential” systems can be represented by interactive models that, when required, may be changed by authorized users—and changing the model, in turn, would mean changing the system [17].

3 Method Conceptualization

The previous section gave an outline of the core modelling facilities provided by MEMO, including its language architecture and meta-modelling language. This section presents selected DSMLs, ways of using and constructing related modelling methods, as well as examples to illustrate the use of MEMO.

3.1 Domain-Specific Modelling Languages

MEMO includes three main languages to model the organisational action system. The Goal Modelling Language (GoalML) [26] [30] supports the design and analysis of corporate goal systems. The Organisation Modelling Language (MEMO OrgML) allows to model organisational structures [11] and business processes [12]. Furthermore, MEMO includes a language to model IT infrastructures (ITML) [24]. The ITML is integrated with detailed concepts to model (IT-related) costs and assign them to cost units [23]. Another language is aimed to support knowledge management [32]. Recently, a variety of further concepts have been developed, which either enrich or build on existing languages. These enable to describe organizational decision processes [3], to model performance indicator systems [35], and to supplement models of IT infrastructures with IT security aspects [20].

Figure 5 shows meta model excerpts for different MEMO DSMLs and demonstrates how they are integrated through common concepts. On the basis of these modelling languages, numerous diagram types can be created. Furthermore, because the MEMO DSMLs are integrated through shared concepts, diagrams may also be created using several DSMLs at once. For example, a diagram may include parts of a business process model referring to goals from a goal model and to IT resources from an IT infrastructure model (for illustrations, see section 3.2 and 4.4.1).

The MEMO DSMLs resulted from research projects that aimed at developing elaborate and comprehensive languages. As a consequence, most of the metamodels are voluminous and include many constraints. Especially for teaching purposes, some of the languages turned out to be too heavy-weight, which suggests to either supplement them with light versions or to provide tool functionality that allows to hide concepts which are not required for certain scenarios (see section 4 for a discussion of the related implementation).

3.2 Examples

The following examples aim to give an overview of MEMO DSMLs and the construction of modelling methods (further examples indicating the wide range of possible use scenarios are found in section 4.4.1). Figure 6 shows the representation of an enterprise model in the form of various diagrams and selected associations

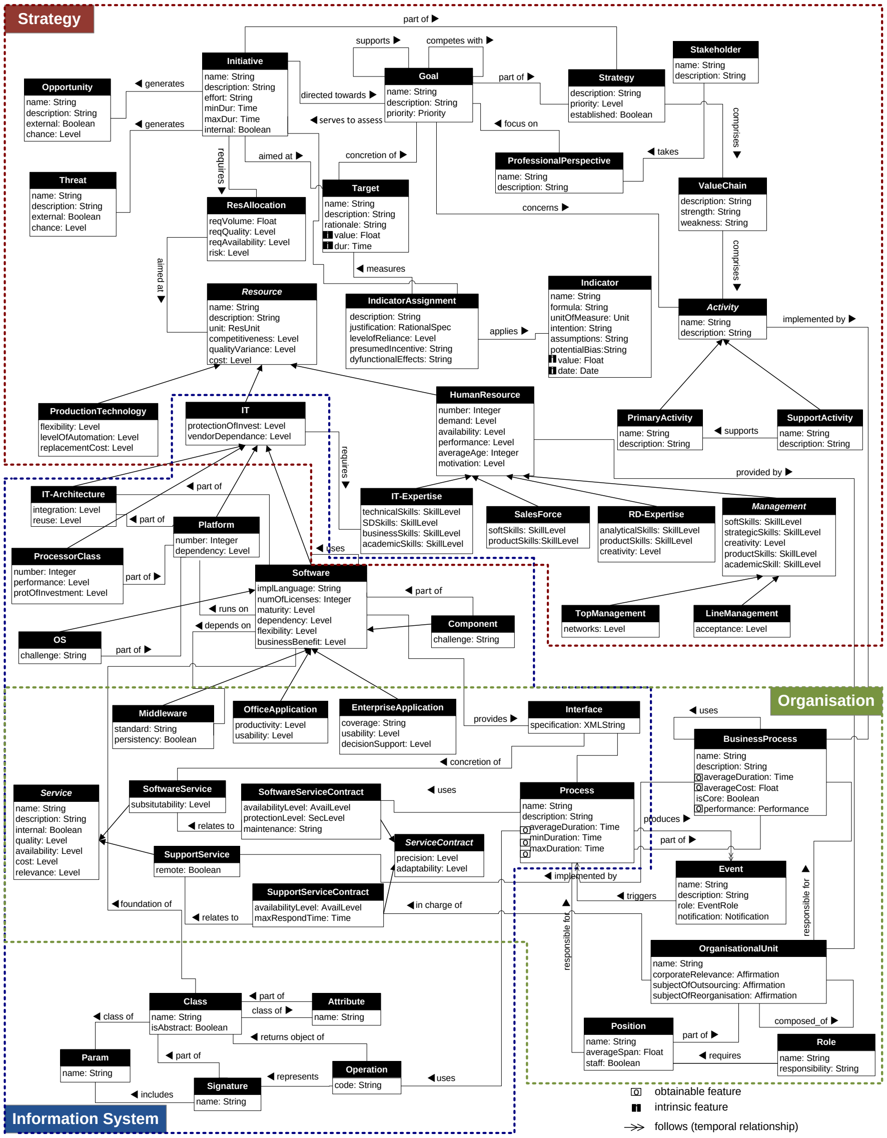


Fig. 5: Excerpt of various integrated metamodels

between them. A goal system diagram represents various types of goals and their interrelationships. As can be seen, the goal “increase number of sales agents” is related to a department in the organisational chart which is responsible for reaching that goal. At the same time, this department is in charge of a particular subprocess within a business process diagram. A business process diagram shows the control flow of a particular process type. An overview of different business process types of

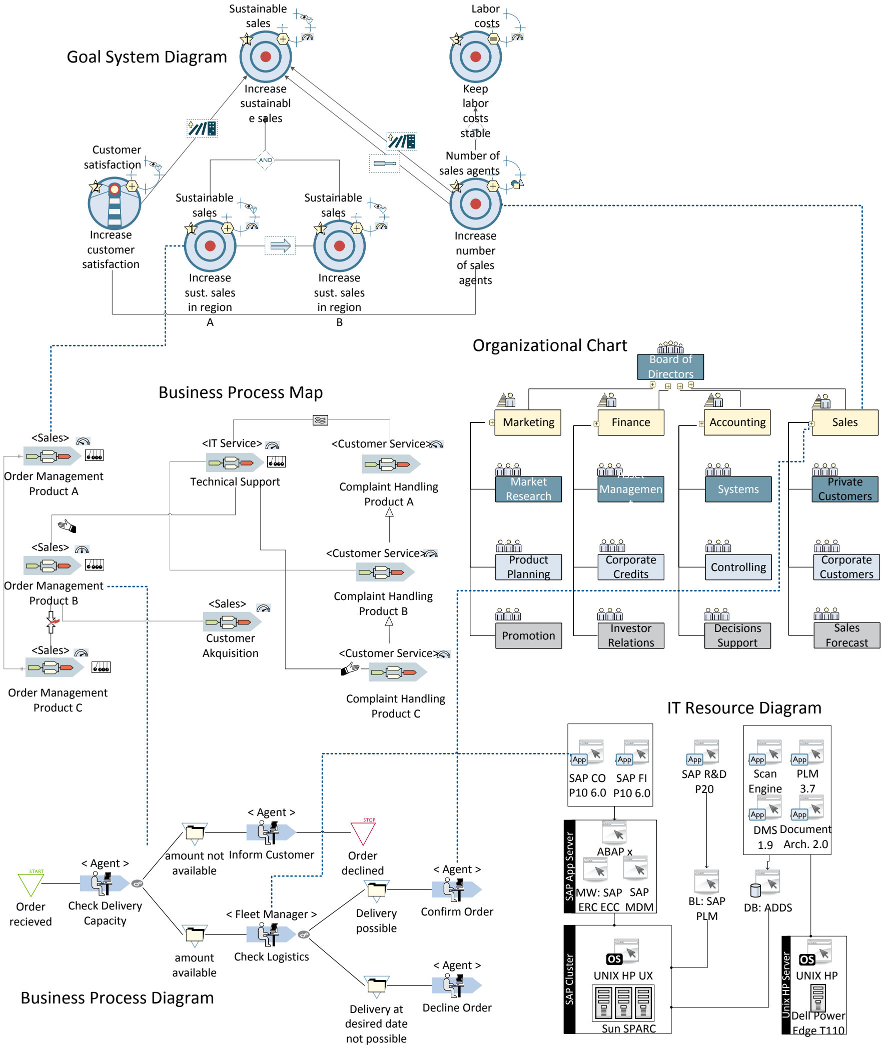


Fig. 6: Interrelated diagrams representing an enterprise model

the enterprise, in turn, is provided by the business process map. The association between the subprocess “Check Logistics” and the application system “SAP CO P10 6.0” is used to indicate that this application system is required to run the subprocess. Finally, to illustrate how the creation and use of such models can be supported, figure 7 shows an excerpt of the representation of an example modelling method that deals with the selection of software systems. MEMO includes a metamodel that supports the construction of further modelling methods (see [15, pp. 950–954]).

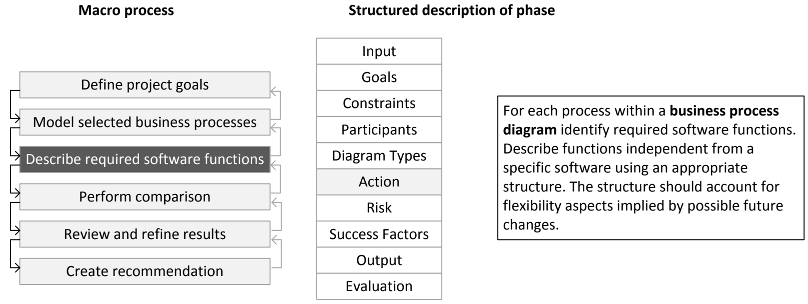

Macro process
Structured description of phase
Define project goals
Model selected business processes
Describe required software functions
Perform comparison
Review and refine results
Create recommendation
Input
Goals
Constraints
Participants
Diagram Types
Action
Risk
Success Factors
Output
Evaluation
For each process within a business process diagram identify required software functions. Describe functions independent from a specific software using an appropriate structure. The structure should account for flexibility aspects implied by possible future changes.

Fig. 7: Example structure of a process within a MEMO modelling method

3.3 Related Work

A variety of enterprise modelling methods are available. An in-depth comparison would go beyond scope of this chapter. Therefore, the following overview aims to point out specific particularities of selected approaches.

Zachman's framework for enterprise architecture [38] suggests to regard enterprise models as integrated conceptual models of data, functions, processes, and other basic aspects. The framework remains on a high level of abstraction and does not include specific modelling languages. The "Architecture of Integrated Information Systems" (ARIS) [33] offers a high-level framework ("House of Business Engineering") together with example diagrams. It provides one original modelling language, the "event-driven process chain", and refers to existing modelling languages such as the ERM or DFDs. The method is supported by a comprehensive commercial toolset. CIM-OSA ("Computer Integrated Manufacturing, Open Systems Architecture") [1] was aimed at modelling manufacturing firms. In addition to a high-level framework that covers three dimensions, CIM-OSA includes various templates that serve to collect data at different levels of abstraction. The completed templates are considered to represent an enterprise model.

SOM ("Semantic Object Model") is primarily aimed at supporting system design and implementation [7]. It combines object-oriented concepts with economic concepts such as "business transaction". SOM includes a few modelling languages, mainly for modelling business processes, transactions, and objects. Different from most other approaches, SOM is based on a cybernetic conceptualisation of the enterprise, which means that it emphasizes the identification and construction of control loops. A modelling environment is available for SOM. DEMO ("Design and Engineering Methodology for Organisations") [6] suggests a unique way of modelling the enterprise. On the one hand, it emphasizes an engineering perspective to support the systematic design of organisations and the analysis of requirements for enterprise software systems. On the other hand, it recommends focussing on collaboration and communication, i.e. on human (inter-)action. DEMO suggests a bottom-up approach to creating enterprise models by starting with basic transactions. While it includes various high-level modelling concepts, which are referred to as "ontology", it does not include the full specification of modelling languages. Tools are available that cover certain aspects of DEMO, but there seems to be no comprehensive tool environment. The "4EM" ("For Enterprise Modelling") method [31] provides basic concepts to model goals, business processes, resources, and related aspects. 4EM particularly promotes enterprise modelling as a participatory process. To describe the languages, the authors mainly refer to example models and place less emphasis on detailed specifications of the abstract syntax and semantics.

TOGAF [36] uses the term "enterprise architecture", which is related to, but not identical with, the notion of an enterprise model. TOGAF is promoted by The Open Group, i.e. it is not an academic project. Its main concern is with the specification of an extensive process model with eight main phases. The documentation includes examples of various diagram types but lacks a specification of modelling languages. ArchiMate [37], also promoted by The Open Group, extends TOGAF with a language to model enterprise architectures. The metamodel is underspecified and leaves room for individual adaptations. ArchiMate modelling tools are freely available.

Taken together, a variety of enterprise modelling methods exist. But different from MEMO, most of these methods do not include a meta-modelling language. Furthermore, most of them lack comparably elaborate specifications of DSMLs.

4 Proof of Concept

In this section, we describe the implementation of selected MEMO languages using ADOxx. The implemented tool is called MEMO4ADO. We decided to use ADOxx especially for two reasons. First, we embrace the idea of creating an "open models" platform. We believe that a movement towards the common development and (re-)use of open models ([18], [25]) is suited to promote the field of enterprise modelling substantially—both in academia and practice. However, we had to learn that a good idea alone is not sufficient to create a movement. It is important to take action and to build incentives. The joint project that is documented in this volume delivers a wide range of reusable modelling tools that are all based on one platform. Therefore, there is a good chance to integrate the tools and, as a consequence, to integrate the models created with these tools as well. This provides a basis to build an attractive collection of reusable models that may also serve as an incentive for the development of further extensions. Second, we realized that our DSMLs are too extensive for teaching purposes. But nonetheless, we wished to involve students in the development, use, and maintenance of our modelling languages and tools. ADOxx seemed to be a good choice for this purpose. Using the ADOxx Development and Modelling Toolkits, we estimated that it would not take students too long to become productive. The implementation in ADOxx also offered an opportunity to devise a MEMO version specifically tailored for teaching purposes. Finally, the "open models" platform provides an attractive vision for teaching: a laboratory of models and modelling languages that cannot only be navigated and examined by students, but which may also be the subject of continuous evolution through student projects.

4.1 Scope and Objectives of the ADOxx Implementation

The basic purpose of the developed MEMO4ADO tool is to offer an accessible facility to help grasp the idea of multi-perspective enterprise modelling, especially for Bachelor's-level students. In addition, we implemented tool-specific auxiliary functions for which the ADOxx environment provided a suitable ground.

The ensuing process by which the modelling languages of MEMO have been implemented was governed by two principal constraints. One concerns the fact that the modelling environment was sought to be usable for teaching purposes. Because the full set of MEMO modelling languages was expected to be too complex for the target group (see section 3), the scope of implementation has been reduced. From the whole set of MEMO modelling languages, a subset of three languages was selected for the present implementation: The MEMO OrgML focusing on *organisational structures* [11], the MEMO OrgML focusing on *business processes* [12], and the MEMO GoalML to describe *organisational goal systems* [26]. In the future, selected concepts of other languages will be added, particularly concepts of the MEMO ITML to describe *IT infrastructures* [23] [24]. Taken together, these languages and concepts provide a coherent method subset to aid basic strategic, organisational, and infrastructural analyses (illustrated in section 4.4). But while limited in number, the selected modelling languages are still complex and comprehensive in scope. They needed to be further adjusted for the target group. The consequential narrowing down of the meta models is described in the following section 4.2.

The second constraint which affected the implementation process is related to the MEMO language architecture. MEMO modelling languages are designed such that a sharp distinction is made between model elements that are instantiated at the type level (M_1) and at the instance level (M_0). To define these relations, the MEMO meta modelling language [13] provides meta modelling concepts such as *intrinsic features* and *language level types* (see section 2.2). Such meta modelling concepts are not a part of the ADOxx meta-metamodel [8, p. 8]. The ADOxx meta-metamodel includes concepts to define meta classes, attributes, and relationships at level M_2, which can be instantiated at type level M_1 in the ADOxx Modelling Toolkit [8, pp. 6–7]. But ADOxx does not offer ways of instantiating and managing instance populations at level M_0 that would represent instantiations of model elements from level M_1. As a result, the selected MEMO meta models had to be redesigned such that the desired domain aspects could all be modeled at exactly one abstraction level. The criteria which have been considered in this process are described below.

4.2 Preparation of Meta Models

The concepts, abstract syntax, and parts of the semantics of the MEMO modelling languages selected for implementation are specified in the form of meta models in different publications (see [11, p. 50] [12, p. 55] [26, p. 201]). Each meta model includes a number of meta types (typically 20-40) plus a range of meta relationships.

The meta models are specified using both common meta modelling concepts such as 'MetaEntity' and 'MetaAttribute' as well as advanced concepts of MEMO MML, including intrinsic features, language level types, and attributes marked as 'derivable' or 'obtainable' (see section 2.1). Additionally, each meta model is augmented with a set of OCL constraints. The existing MEMO meta models served as a starting point for the ADOxx implementation. However, in view of the two constraints described above (section 4.1), a number of modifications and simplifications had to be made to the original meta models so that an implementation would become technically feasible and conceptually adequate for the target group. Because the complete list of modifications and resulting meta models cannot be presented here, the most salient implementation tasks are summarized below and illustrated by means of an example (see figure 8).

Reduction of concepts. In order to advance language accessibility, several concepts were omitted for the tool implementation. This mainly concerns concepts for advanced users to describe domain details or concepts that represent less intuitive abstractions. As an example of the former, advanced OrgML control flow concepts such as 'while' or 'repeat' loops were not considered. As an example of the latter, the GoalML concept 'GoalSystem' was not implemented (see figure 8). While such a concept can serve to specify and analyse joint properties of goal systems (e.g., the transitivity of goal priorities), we opted not to implement it as we did not see a sufficiently clear way of embedding it in the initial set of diagram types.

Modification of concepts. For several MEMO language concepts, we chose to implement a modified and simplified conceptualization. The redesign mainly centered around the aims of easing language use and enhancing tool usability. For instance, the original GoalML meta model demands to specify a goal and its components in terms of several distinct concepts, including 'AbstractGoal', 'GoalMatter', 'SituationalAspect', and various relationships (see the upper part of figure 8). This conceptualization enables to specify a goal in great detail (e.g., it permits to define a goal whose "substance" is composed of a variety of different real-world aspects). It also contributes to reuse (e.g., the same goal matter 'Revenues' could be referenced by several goals). However, in the ADOxx Modelling Toolkit, it might be considered inconvenient to have to create and interlink numerous model elements for the purpose of creating a single goal. Therefore, we subsumed the formerly distinct goal components in one central concept (see the lower part of figure 8). This improves language accessibility—though at the cost of language expressiveness. It remains to be seen which compromise proves useful in the long term.

Modification of attributes. Most type level attributes of the original MEMO meta model concepts remained unchanged. Apart from a few data type mismatches, these attributes were implemented as originally specified. A few type level attributes were dropped to reduce complexity (compare the upper and lower part of figure 8), but this did not affect too many attributes. An important feature of ADOxx is the possibility to specify attributes of type 'Interref'. This enables to have model elements in one diagram hold references to elements in another. We used this feature to achieve integration between diagrams to be created with different MEMO languages (see figure 8 for an example attribute, and see section 4.4 for example diagrams). Be-

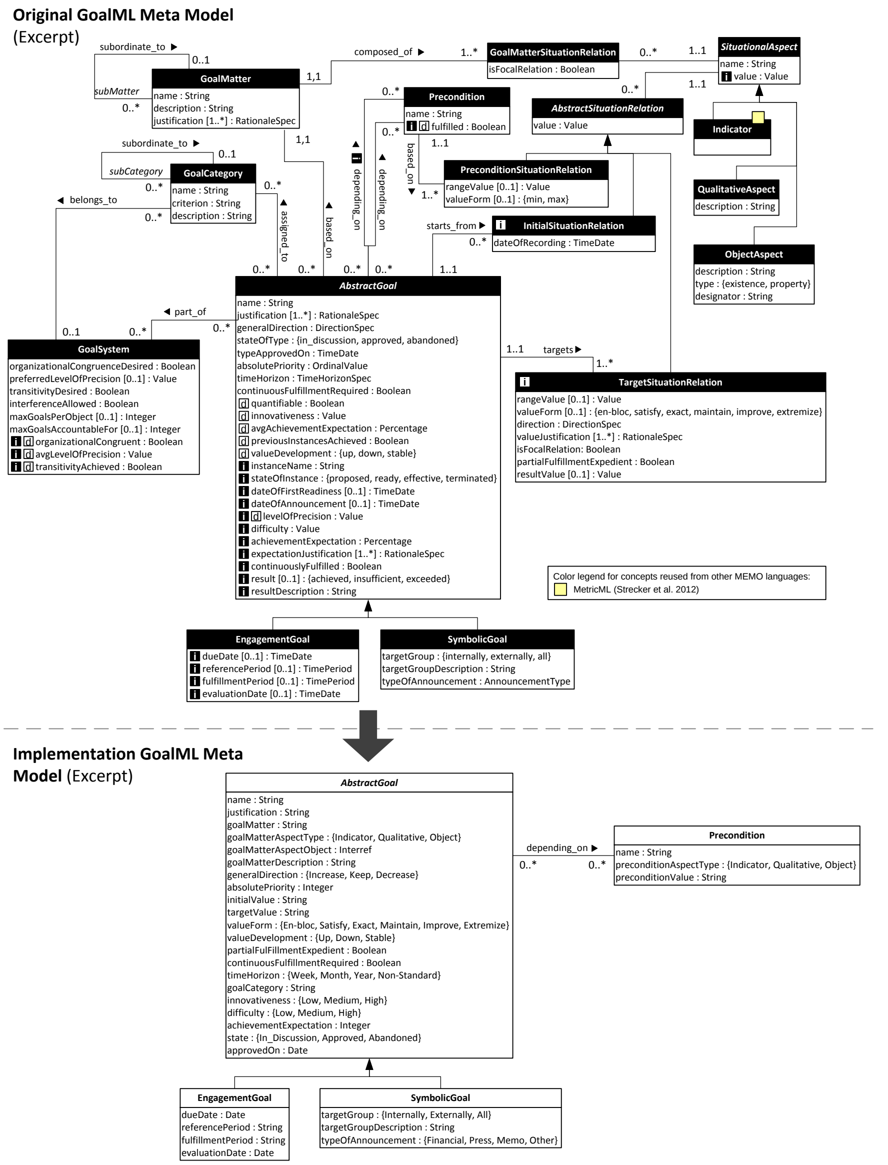

Original GoalML Meta Model
(Excerpt)
subordinate_to ▶
0..1
subMatter
0..*
GoalMatter
name : String
description : String
justification [1..*] : RationaleSpec
1,1
composed_of ▶
1..*
GoalMatterSituationRelation
isFocalRelation : Boolean
0..*
1..1
SituationalAspect
name : String
value : Value
0..*
1..1
AbstractSituationRelation
value : Value
Indicator
QualitativeAspect
description : String
ObjectAspect
description : String
type : {existence, property}
designator : String
0..*
Precondition
name : String
fulfilled : Boolean
0..*
1..1
based_on
1..*
depending_on
depending_on
PreconditionSituationRelation
rangeValue [0..1] : Value
valueForm [0..1] : {min, max}
1,1
subordinate_to ▶
0..1
subCategory
0..*
GoalCategory
name : String
criterion : String
description : String
0..*
assigned_to
based_on
◀ belongs_to
0..*
starts_from ▶
InitialSituationRelation
dateOfRecording : TimeDate
0..*
0..*
0..*
0..*
0..*
1..1
AbstractGoal
name : String
justification [1..*] : RationaleSpec
generalDirection : DirectionSpec
stateOfType : {in_discussion, approved, abandoned}
typeApprovedOn : TimeDate
absolutePriority : OrdinalValue
timeHorizon : TimeHorizonSpec
continuousFulfillmentRequired : Boolean
quantifiable : Boolean
innovativeness : Value
avgAchievementExpectation : Percentage
previousInstancesAchieved : Boolean
valueDevelopment : {up, down, stable}
instanceName : String
stateOfInstance : {proposed, ready, effective, terminated}
dateOfFirstReadiness [0..1] : TimeDate
dateOfAnnouncement [0..1] : TimeDate
levelOfPrecision : Value
difficulty : Value
achievementExpectation : Percentage
expectationJustification [1..*] : RationaleSpec
continuouslyFulfilled : Boolean
result [0..1] : {achieved, insufficient, exceeded}
resultDescription : String
◀ part_of
0..1
0..*
0..*
GoalSystem
organizationalCongruenceDesired : Boolean
preferredLevelOfPrecision [0..1] : Value
transitivityDesired : Boolean
interferenceAllowed : Boolean
maxGoalsPerObject [0..1] : Integer
maxGoalsAccountableFor [0..1] : Integer
organizationalCongruent : Boolean
avgLevelOfPrecision : Value
transitivityAchieved : Boolean
1..1
targets ▶
1..*
TargetSituationRelation
rangeValue [0..1] : Value
valueForm [0..1] : {en-bloc, satisfy, exact, maintain, improve, extremize}
direction : DirectionSpec
valueJustification [1..*] : RationaleSpec
isFocalRelation: Boolean
partialFulfillmentExpedient : Boolean
resultValue [0..1] : Value
Color legend for concepts reused from other MEMO languages:
MetricML (Strecker et al. 2012)
EngagementGoal
dueDate [0..1] : TimeDate
referencePeriod [0..1] : TimePeriod
fulfillmentPeriod [0..1] : TimePeriod
evaluationDate [0..1] : TimeDate
SymbolicGoal
targetGroup : {internally, externally, all}
targetGroupDescription : String
typeOfAnnouncement : AnnouncementType
Implementation GoalML Meta
Model (Excerpt)
AbstractGoal
name : String
justification : String
goalMatter : String
goalMatterAspectType : {Indicator, Qualitative, Object}
goalMatterAspectObject : Interref
goalMatterDescription : String
generalDirection : {Increase, Keep, Decrease}
absolutePriority : Integer
initialValue : String
targetValue : String
valueForm : {En-bloc, Satisfy, Exact, Maintain, Improve, Extremize}
valueDevelopment : {Up, Down, Stable}
partialFulFillmentExpedient : Boolean
continuousFulfillmentRequired : Boolean
timeHorizon : {Week, Month, Year, Non-Standard}
goalCategory : String
innovativeness : {Low, Medium, High}
difficulty : {Low, Medium, High}
achievementExpectation : Integer
state : {In_Discussion, Approved, Abandoned}
approvedOn : Date
depending_on ▶
0..*
0..*
Precondition
name : String
preconditionAspectType : {Indicator, Qualitative, Object}
preconditionValue : String
EngagementGoal
dueDate : Date
referencePeriod : String
fulfillmentPeriod : String
evaluationDate : Date
SymbolicGoal
targetGroup : {Internally, Externally, All}
targetGroupDescription : String
typeOfAnnouncement : {Financial, Press, Memo, Other}

Fig. 8: Meta model preparation example

cause ADOxx keeps track of inter-diagram references and it raises a warning in case referenced elements would be deleted, this also contributes to model consistency. Beyond the implementation of basic attributes, however, a crucial question concerned possible ways of implementing ‘intrinsic’ attributes (see below).

Reconfiguring abstraction levels. MEMO meta models contain elements declared as ‘intrinsic’. These elements are to be instantiated at level M_0 rather than M_1. The ADOxx meta modelling environment does not possess directly comparable (meta) language concepts. More generally, the ADOxx Modelling Toolkit does not maintain an instance level where these elements could in fact be instantiated. But dropping intrinsic elements altogether would not be desirable either, as in some MEMO languages important domain aspects are captured at that level (e.g., time-related goal aspects or organisational goal responsibilities, see figure 8). In the end, we decided to follow a compromise approach. Aspects which clearly relate to singular occurrences and cannot usefully be considered in the present modelling environment were neglected (e.g., attributes such as ‘startTime’ of the concept ‘ConrolFlowSubProcess’ [12, p. 55]). Other intrinsic model elements whose assignment to the instance level is not as unambiguous were moved to the type level to avoid losing important domain-specific language expressiveness. For example, this concerns time-related attributes of the concept ‘EngagementGoal’. It also concerns the relationships ‘AccountabilityRelation’ and ‘InitiationRelation’ (see figure 8). These relationships offer a linking point between goal system diagrams and organisational structure diagrams. Omitting these relationships would have significantly decreased the value of the MEMO4ADO tool.

4.3 Process

The implementation was conducted in the form of a small project at our chair, coupled with a few Bachelor’s-level student projects. We decided to involve students in the process for two reasons. First, implementing modelling languages addresses skills central for our field—the ability to grasp advanced abstractions (as required in meta modelling), scrutinizing and integrating domain-specific concepts from the field of management studies, and technical proficiency. Second, involving students provided an opportunity to gather first-hand feedback and suggestions on language and tool usability. At our chair, a few student assistants supported us in implementing the languages.[1] Bachelor’s student projects, in turn, were completed by student groups of 3-4 persons. From the conducted students’ projects, a project concerned with the OrgML for organisational structures has particularly contributed to the present tool.[2] Figure 9 shows a timeline of implementation milestones.

In our experience, the time needed to complete implementation tasks varied greatly between different kinds of tasks. The preparation of meta models typically took considerable time, as it demanded to reconcile different language architectures (see section 4.2). This task was particularly challenging for Bachelor’s-level student

---

[1] In particular, we would like to thank David Becher for his major contributions to the implementation of the OrgML (business processes), the GoalML, and the integration of the languages.

[2] We would like to thank the project members Jeannot Gerth, Jonas Kaiser, Jesse Okpure, and Marijan Srsa for their important contributions to the implementation of the OrgML (organisational structures).

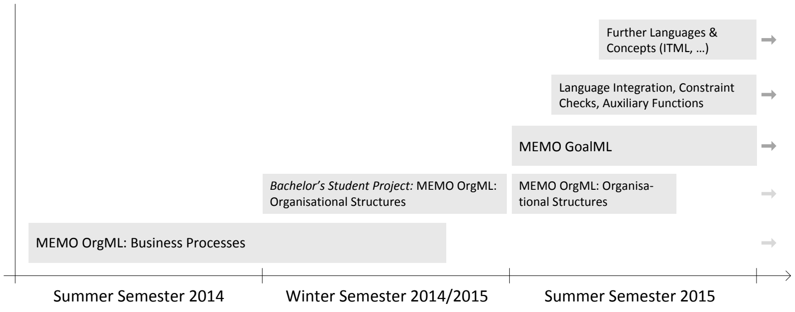


Fig. 9: Implementation milestones

groups because most members were not familiar with advanced meta modelling concepts at the beginning of the process. The following implementation of selected concepts and their attributes could usually be done swiftly, as this is a straightforward exercise in the ADOxx Development Toolkit. The same is true for the concrete syntax. Then again, what tended to require great effort was the implementation of additional constraints which accompany all MEMO meta models. In the MEMO language architecture, these constraints are defined using OCL. In ADOxx, such constraints can be implemented in the form of manually invocable or event-triggered routines written in the ADO script language. However, because OCL is a declarative language and the ADOxx script language is an imperative one, the transformation was not trivial. It was also complicated by the fact that several MEMO language constraints are quite complex in nature (see, e.g., [12, pp. 61–63] [26, pp. 203–207]). In consequence, the present tool is capable of evaluating some, but not all MEMO language constraints. The implementation of further constraint checks is an ongoing activity (see figure 9). Finally, the time it took to realize additional auxiliary functions (such as different levels of notational details; see section 4.4.2) varied with the nature of the function. In the future, we will continue to add language concepts of further MEMO languages to the tool (in particular, ITML concepts) and further enhance the existing implementation with additional features and constraint checks.

4.4 Tool Application and Case Study

On the basis of the MEMO modelling languages, the MEMO4ADO tool provides a platform to describe various organisational and technological facets of an enterprise in an integrative manner, enabling various kinds of reflective analyses. The basic way of using the MEMO4ADO tool consists in creating different *core diagrams* that describe selective abstractions of the enterprise in question (i.e., goal systems, organisational structures, and business process control flows) and to subsequently interrelate them by means of additional *integrative diagrams* and *references* between the diagrams. Dependent on the specific MEMO-based method followed, the diagrams may be created in different partial orders (e.g., top-down considerations starting with goal systems, or operational analyses starting with business process control flows; see also sections 2.3 and 3.2). However, it is constitutive of MEMO that the models are not to be taken as a single-blow approach to specify and afterwards implement parts of an organisation (i.e. to "engineer" an enterprise in a narrow sense). Instead, the models are intended to serve as a means of analysing, reflecting on, and perhaps rethinking ways of working in iterative, collaborative processes. The full scope of possible application scenarios for the languages cannot be presented here. For details, see the process models and examples in the respective publications ([12], [13], [26]). To offer an overview of the modelling tool, we first present the implemented diagram types and illustrate their basic use by means of an example (section 4.4.1). Subsequently, we describe auxiliary functions that have been implemented to aid language use (section 4.4.2).

4.4.1 Basic Functionality and Diagram Types

Figure 10 shows an overview case of the different diagram types that can be created with the MEMO4ADO tool. The figure also illustrates the interrelations between the different diagram types and exemplifies how model elements defined in one diagram may be referenced in another. The diagram types are described successively below.

Goal System Diagram (core diagram type; top left). This is the essential MEMO GoalML [26] [30] diagram type to establish, investigate, and restructure an organisation's goal system. The diagram type permits to describe goals, of which the GoalML offers two kinds, and their various possible relationships at a high level of detail. *EngagementGoals* describe goals which are mainly established for specific organisational units and whose attainment can be measured after a given period. *SymbolicGoals* are intended to describe goals that serve broader purposes of motivation and inspiration. Each kind of goal possesses a variety of attributes to describe goal components and properties. For many attributes, dedicated notational symbols are offered whose appearance is automatically adjusted based on the current attribute value (e.g., *absolutePriority*, *generalDirection*, and others). Between all kind of goals, a variety of relationships can be defined, including *decompositional*, *means-ends*, *causal*, *mathematical*, and *prioritizing* relationships. All of these relationships can be further qualified by attribute values (e.g., conflicting vs. complementary means-ends relationships). The qualifications are visually represented using different supplementary symbols. For further details on this diagram type and the possible analyses it supports, see [26, pp. 241–247] [30, pp. 4–7].

Organisational Structure Diagram (core diagram type; middle left). This diagram type, which is based on the MEMO OrgML [12], offers a means of describing and analysing static aspects of a (formal) organisation. In essence, it can be used to create more elaborate variants of traditional organisational charts consisting of, and interlinking, elements such as *organisational units*, *positions*, *roles*, and *committees*. In contrast to conventional organisational charts found in textbooks, however,

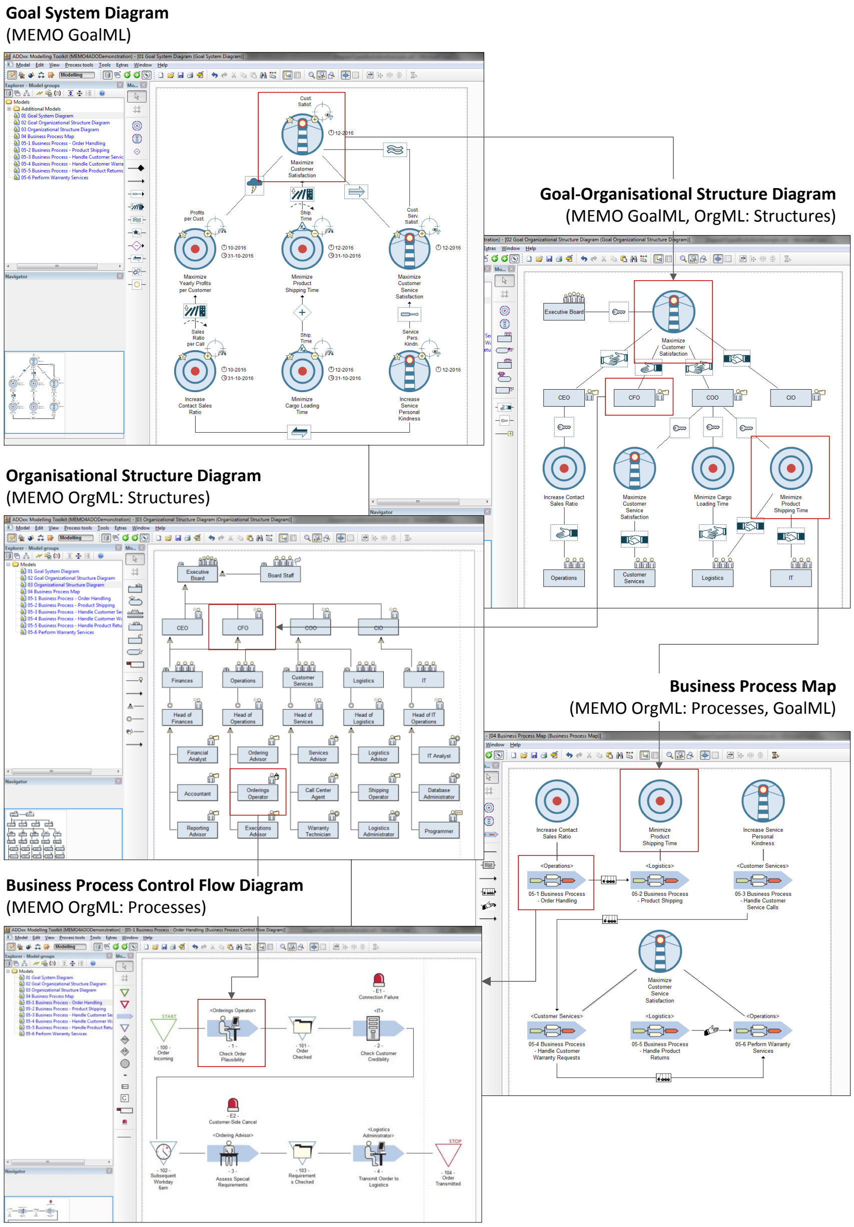

Goal System Diagram
(MEMO GoalML)
Goal-Organisational Structure Diagram
(MEMO GoalML, OrgML: Structures)
Organisational Structure Diagram
(MEMO OrgML: Structures)
Business Process Map
(MEMO OrgML: Processes, GoalML)
Business Process Control Flow Diagram
(MEMO OrgML: Processes)
Maximize Customer Satisfaction
Maximize Yearly Profits per Customer
Minimize Product Shipping Time
Maximize Customer Service Satisfaction
Increase Contact Sales Ratio
Minimize Cargo Loading Time
Increase Service Personal Kindness
Executive Board
Board Staff
CEO
CFO
COO
CIO
Finances
Operations
Customer Services
Logistics
IT
05-1 Business Process - Order Handling
05-2 Business Process - Product Shipping
05-3 Business Process - Handle Customer Service Calls
05-4 Business Process - Handle Customer Warranty Requests
05-5 Business Process - Handle Product Returns
05-6 Perform Warranty Services
Check Order Plausibility
Check Customer Credibility
Assess Special Requirements
Transmit Order to Logistics

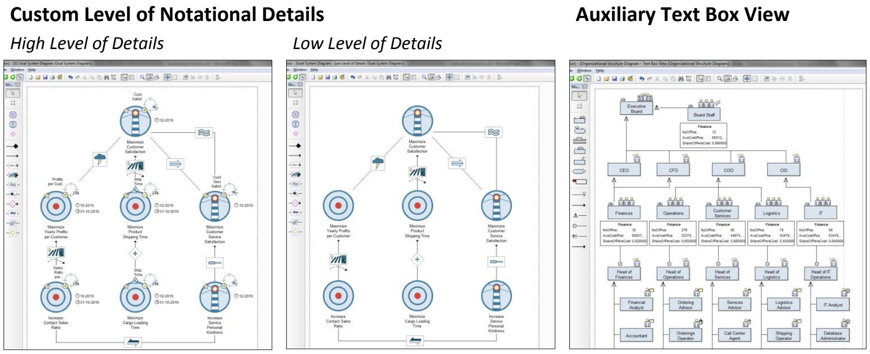
Fig. 10: MEMO4ADO diagram types

the semantics of the notational elements and relationships are well-described. This is reflected in a variety of relationships types, including *composed of*, *supervised by*, and *subordinated to*. In addition to describing basic organisational structures, the language permits to record a host of advanced properties for the elements. This includes attributes to describe the *position type* (‘Sales’, ‘Technical’, and others), *averagePerformance*, or the *requiredQualification*. For several attributes, again, the current values are displayed dynamically in the form of visual symbols. For further details on this diagram type, its variants, and possible uses, see [12, pp. 64–72].

Goal-Organisational-Structure Diagram (integrative diagram type; upper right). This diagram type enables to integrate elements from an existing organisational structure diagram with goals taken from an goal system diagram (see above). To achieve consistency among existing diagrams, the diagram type uses concepts that reference elements defined in other diagrams. For example, when adding a new *SymbolicGoalReference*, the goal ‘Maximize Customer Satisfaction’ defined in the existing goal system diagram (top left in figure 10) can be referenced (this is implemented using ‘Interref’ attributes). The goal’s name and an inter-diagram link are adopted automatically. The same can be done to reference organisational units. Having established these referential elements, the relationship *InitiationRelation* can be used to express which unit or position has initialized or mandated the specification of a certain goal. The *AccountabilityRelation*, in turn, permits to define which unit is responsible for achieving that goal (different degrees of commitment can be defined using the attribute *commitment*). In sum, this diagram type allows to clarify interrelations between goal systems and organisational units, to the effect that they can also be traced transparently (by clicking on the links automatically attached to the symbols in the diagram, or using the ADOxx function ‘Follow’). See [26, pp. 247–251] and [30, pp. 7–8] for further remarks on this diagram type.

Business Process Control Flow Diagram (core diagram type; bottom left). This diagram type, based on the second part of the MEMO OrgML [13], provides the ability to specify business process control flows. The essential concepts are *subprocesses* and *events*. For both concepts, different types can be selected (e.g., manual, IT-supported, and fully automated sub-processes; and time-triggered, message-triggered, or generic change-triggered events). Again, changes of these and other attribute values are signified visually. The organisational units or positions which are responsible for fulfilling a given sub-process can be specified using the *in charge of* attribute, representing another inter-diagram relationship (see figure 10). Subprocesses and events have to be related alternately by means of the central *subsequent* relationship. More complex control flow structures can be defined using parallelization (concept *Fork*) or exclusive path decision points (concept *Decision*). For further details and examples regarding this diagram type, see [13, pp. 89–95].

Business Process Map (integrative diagram type; lower right). This diagram type distinguishes itself from control flow diagrams in that it provides an overview of different business process types in an enterprise rather than describing one business process type in detail. The central concept, *business process*, offers a reference attribute which allows to link it to a control flow diagram. As a result, business process maps can be used as a starting point to navigate to richer descriptions of particular business process types. Business processes may also be related on a macro-level, such as by means of the relationships *supports*, *may trigger*, and *similar to*. Finally, because business processes may also be taken as reference objects of organisational


Custom Level of Notational Details

High Level of Details

Low Level of Details

Auxiliary Text Box View

Fig. 11: Auxiliary function examples

goals, this diagram type enables to add references from business processes to existing goals (using the same concepts as in goal-organisational-structure diagrams). For example, the organisational goal 'Minimize Product Shipping Time' might refer to the business process type 'Product Shipping' (middle of the business process map in figure 10). Again, the elements in the diagram contain clickable links to directly navigate to the source diagrams. See [13, p. 89] for further notes on business process maps and [26, p. 253] for general remarks on goal reference object diagrams.

Taken together, the above diagram types represent a coherent and tightly integrated subset of MEMO diagrams. Using the capabilities of the ADOxx environment, interrelationships between different areas of an enterprise can be recorded and explored dynamically. Possible application scenarios, which could only be indicated here, range from strategic goal planning processes to organisational restructuring efforts to process bottleneck analyses. For more details on these and further capabilities of the languages, refer to the cited language documentations.

4.4.2 Auxiliary Functions and Support in Language Use

While the basic diagram types offer a ground for creating and interlinking various diagrams, building and using comprehensive enterprise models is still a challenging task. In order to ease language use and to improve model versatility and interpretability for different target groups, we are continuously adding auxiliary functions to the environment. Three of these are briefly outlined below.

Constraint Checks. The MEMO meta models are accompanied by a number of constraints to prevent semantically inconsistent or nonsensical models. We implemented a subset of these constraints. Two kinds of constraints exist (see [13, pp. 57–63]). Ad hoc constraints can be recognized as soon as an invalid model part is created. For instance, it is not possible that two positions are the superior of each other at the same time. When trying to define such a relation, an error message containing an explanation is shown. Other constraints can be checked only once a model is considered complete. For example, a complete business process control flow model needs to include a start and a stop event. For these kinds of constraints, we implemented the feature 'Check Model Validity' (found in the menu). When calling this function, a model is evaluated and possible errors and confirmation messages are displayed in the form of a user dialog. Please note that while we are intending to further extend the scope of constraint checks, the tool does not check the full set of constraints of the MEMO languages.

Different Levels of Notational Detail. When considering MEMO diagrams such as those shown in figure 10, it can be found that these diagrams exhibit a level of notational detail which might be overly complicated for some purposes. To improve diagram legibility and clarity for these scenarios, we implemented a function that allows to switch between different levels of detail. This feature is currently available for the GoalML and perhaps will be added for other languages in the future. The function is found in the menu. Figure 11 (left-hand side) illustrates a goal system diagram with a high level of detail as compared to one with a lower level of detail.

Auxiliary Overview Textboxes. Attribute values of model elements can be specified and investigated using the ADOxx notebook dialog. Sometimes, it may also be considered helpful to see values of attributes at a glance while interpreting a diagram. For this purpose, we implemented various additional text box views for each language. These views attach a basic text box to each model element in which the values for selected attributes are listed textually. Some modes focus on general overview attributes, whereas others focus on special topics such as financial aspects. Figure 11 (right-hand side) shows an example.

Building on the underlying diagram types, the auxiliary functions are meant to additionally enhance the usability and convenience of the MEMO4ADO tool. Further enhancing the scope of these functions, as well as adding further modes of accommodating tool use are ongoing tasks on our agenda.

5 Conclusion

MEMO is an enterprise modelling method whose development was initiated as early as two decades ago and which continues to be a subject of active research. Studies into ways of modelling organisational action systems and information systems have resulted in an integrated and still growing set of comprehensive modelling languages. ADOxx is a valuable addition to the modelling environments that have been used in the history of MEMO. First, ADOxx is highly accessible, both for language users and meta modelers. For language users, the ADOxx Modelling Toolkit offers a clear user interface which hides many unnecessary technical (meta) modelling details. Initial application in Bachelor's-level modelling courses indicate that students swiftly learn how to use the MEMO4ADO tool, and that they appreciate the modelling support it offers. Compared to our Ecore- and Eclipse-based tools, we have the impression that ADOxx is regarded as more intuitive and ergonomic. We also see that ADOxx facilitates the developer's task, as the pre-defined meta level concepts provided in the Development Toolkit represent a convenient basis to develop modelling languages for two-level language architectures. We furthermore appreciate that ADOxx represents a stable and mature environment. Finally, ADOxx provides a basis for implementing a host of additional features, drawing on its capabilities for model analysis and the incorporated ADO script language. The benefits of these features are exemplified in this and the other contributions in this volume.

At the same time, investigations into the nature of reconstructing professional languages by means of conceptual (meta) models have led us to conclude that traditional two-level language architectures do not suffice to satisfy advanced modelling requirements. These insights have, as a first step, stimulated the development of advanced meta modelling concepts for three-level language architectures (e.g., "intrinsic features", discussed in this contribution). These concepts could not directly be translated to the ADOxx language architecture. Furthermore, our work on the specification of modelling languages confirmed the assumption that a flexible number of classification levels promotes reusability and flexibility of models and languages. In addition, the simultaneous use of a programming language that also features an arbitrary number of classification levels allows the common representation of models and code. Because an account of an arbitrary number of classification levels is beyond the scope of most current modelling tools, including ADOxx, we are now working on an XMF-based tool that in fact provides the ability to create multilevel models and multilevel software systems that share the same representation [16]. The further development of this tool as well as the reconstruction of existing MEMO languages for a corresponding architecture is where our future research is heading.

The MEMO4ADO tool presented in this chapter is meant to provide a platform to dynamically create and explore enterprise models on a coherent, limited scope. The tool focuses mainly on concepts to model aspects of organisational action systems (e.g., goals, structures, and processes). By utilizing the capabilities of ADOxx, it especially enables to recognize and trace links between different areas of an enterprise. We intend to use the tool in the context of a Bachelor's-level modelling course as a laboratory in which students are asked to investigate an existing multi-perspective enterprise model and to extend it with further partial models, such as new business processes. Future development of MEMO4ADO will include the implementation of further auxiliary functions and the integration of additional concepts from other MEMO languages. For example, we wish to add concepts from MEMO ITML [23] to provide the ability to describe information systems infrastructures and to enable additional diagram types in which IT concepts can be interrelated with elements of the organisational action system.

References

1. CIMOSA: Open system architecture for CIM. Springer, Berlin, Heidelberg, New York (1993)
2. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software & Systems Modeling **7**(3), 345–359 (2008)

3. Bock, A.: Beyond Narrow Decision Models: Toward Integrative Models of Organizational Decision Processes. In: D. Aveiro, U. Frank, K.J. Lin, J. Tribolet (eds.) Proceedings of the 17th IEEE Conference on Business Informatics (CBI 2015). IEEE Press, Los Alamitos (2015)
4. Clark, T., Sammut, P., Willans, J.: Applied Metamodelling: A Foundation for Language Driven Development, 2 edn. Ceteva (2008). URL http://www.eis.mdx.ac.uk/staffpages/tonyclark/Papers/Applied%20Metamodelling%20%28Second%20Edition%29.pdf
5. Clark, T., Willans, J.: Software language engineering with XMF and XModeler. In: M. Mernik (ed.) Formal and Practical Aspects of Domain-Specific Languages, pp. 311–340. Information Science Reference (2012)
6. Dietz, J.L.G.: Enterprise Ontology: Theory and Methodology. Springer, Berlin, New York (2006)
7. Ferstl, O.K., Sinz, E.J.: Modeling of business systems using SOM. In: P. Bernus, K. Mertins, G. Schmidt (eds.) Handbook on Architectures of Information Systems, pp. 347–367. Springer, Berlin and Heidelberg, New York (2006)
8. Fill, H.G., Karagiannis, D.: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. Enterprise Modelling and Information Systems Architectures **8**(1), 4–25 (2013)
9. Frank, U.: Multiperspektivische Unternehmensmodellierung: Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung. Oldenbourg, München (1994)
10. Frank, U.: The MEMO Meta-Metamodel. Research Report of the Institute for Business Informatics 9, University of Koblenz, Koblenz (1998)
11. Frank, U.: MEMO Organisation Modelling Language (1): Focus on Organisational Structure. ICB Research Report 48, University of Duisburg-Essen, Essen (2011)
12. Frank, U.: MEMO Organisation Modelling Language (2): Focus on Business Processes. ICB Research Report 49, University of Duisburg-Essen, Essen (2011)
13. Frank, U.: The MEMO Meta Modelling Language (MML) and Language Architecture: 2nd Edition. ICB Research Report 43, University of Duisburg-Essen, Essen (2011)
14. Frank, U.: Domain-specific modeling languages - requirements analysis and design guidelines. In: Reinhartz-Berger, Iris, Sturm, Aron, Clark, Tony, Wand, Yair, Cohen, Sholom, Bettin, Jorn (eds.) Domain Engineering: Product Lines, Conceptual Models, and Languages, pp. 133–157. Springer (2013)
15. Frank, U.: Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Software & Systems Modeling **13**(3), 941–962 (2014)
16. Frank, U.: Multilevel modeling: Toward a new paradigm of conceptual modeling and information systems design. Business and Information Systems Engineering **6**(6), 319–337 (2014)
17. Frank, U.: Power-modelling: Toward a more versatile approach to creating and using conceptual models. In: Proceedings of the Fourth International Symposium on Business Modelling and Software Design, pp. 9–19 (2014)
18. Frank, U., Strecker, S.: Open reference models – community-driven collaboration to promote development and dissemination of reference models. Enterprise Modelling and Information Systems Architectures **2**(2), 32–41 (2007)
19. Frank, U., Strecker, S.: Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems. ICB Research Report 31, University of Duisburg-Essen, Essen (2009)
20. Goldstein, A., Frank, U.: Components of a multi-perspective modeling method for designing and managing it security systems. Information Systems and e-Business Management (2015, in press)
21. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley, Amsterdam (2009)
22. Gulden, J., Frank, U.: MEMOCenterNG – a full-featured modeling environment for organisation modeling and model-driven software development. In: Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development (FTMDD 2010) (2010)
23. Heise, D.: Unternehmensmodell-basiertes IT-Kostenmanagement als Bestandteil eines integrativen IT-Controllings. Logos, Berlin (2013)
24. Kirchner, L.: Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung. Logos, Berlin (2008)

25. Koch, S., Strecker, S., Frank, U.: Conceptual modelling as a new entry in the bazaar: The open model approach. In: E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto, G. Succi (eds.) Open Source Systems, pp. 9–20. Springer, New York (2006)
26. Köhling, C.A.: Entwurf einer konzeptuellen Modellierungsmethode zur Unterstützung rationaler Zielplanungsprozesse in Unternehmen. Cuvillier, Göttingen (2013)
27. Object Management Group: Meta Object Facility (MOF) Core Specification: Version 2.0 (2006). URL http://www.omg.org/spec/MOF/2.0/
28. Object Management Group: Object constraint language: Version 2.2 (2010). URL http://www.omg.org/spec/OCL/2.2/
29. Odell, J.J.: Power types. Journal of Object-Oriented Programming **7**(2), 8–12 (1994)
30. Overbeek, S., Frank, U., Köhling, C.A.: A language for multi-perspective goal modelling: Challenges, requirements and solutions. Computer Standards & Interfaces **38**, 1–16 (2015)
31. Sandkuhl, K.: Enterprise Modeling: Tackling Business Challenges with the 4EM Method. The Enterprise Engineering Series. Springer, Berlin (2014)
32. Schauer, H.: Unternehmensmodellierung für das Wissensmanagement: Eine multiperspektivische Methode zur ganzheitlichen Analyse und Planung. VDM, Saarbrücken (2009)
33. Scheer, A.W.: Architecture of Integrated Information Systems: Foundations of Enterprise Modelling. Springer, Berlin, New York (1992)
34. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2 edn. Addison-Wesley, Upper Saddle River (2009)
35. Strecker, S., Frank, U., Heise, D., Kattenstroth, H.: MetricM: A modeling method in support of the reflective design and use of performance measurement systems. Information Systems and e-Business Management **10**(2), 241–276 (2012)
36. The Open Group: TOGAF Version 9. The Open Group Series. Van Haren, Zaltbommel (2009)
37. The Open Group: ArchiMate 2.0 Specification: Open Group Standard. The Open Group Series. Van Haren, Zaltbommel (2012)
38. Zachman, J.A.: A framework for information systems architecture. IBM Systems Journal **26**(3), 276–292 (1987)