MEMOCenterNG – A full-featured modeling environment for enterprise modeling and model-driven software development

Jens Gulden and Prof. Dr. Ulrich Frank

University Duisburg-Essen,
Universitaetsstr. 9, 45141 Essen, Germany
{jens.gulden,ulrich.frank}@uni-duisburg-essen.de
http://www.wi-inf.uni-duisburg-essen.de/FGFrank/

Abstract. We present MEMOCenterNG[1], an integrated, full-featured modeling environment containing 11 built-in modeling languages together with meta-modeling support for creating new languages and corresponding diagram editors. Tools for analysis, model transformation and code generation are included.

Key words: Modeling, Model-driven software development, Multi-perspective modeling, Enterprise modeling, Meta-modeling

1 A comprehensive multi-language modeling tool

Large modeling projects typically use multiple modeling languages simultaneously to express a variety of aspects of a modeled system. This holds true in diverse types of modeling projects, especially in business process modeling scenarios, which unite conceptual business aspects with technical realizations. Model-driven software development benefits from multiple interrelated modeling perspectives to gain a coherent and as complete view as possible on the system to be developed.

To foster large modeling projects, an integrated modeling environment tool is desirable to reduce efforts in combining multiple model editors operating on shared model content. Semantic integrity is an indispensable requirement for multi-perspective modeling [7], that means, when different model editors reference the same concept (e.g., a person displayed both in an organization structure diagram and in a process diagram), the model editor components need to share common information about its unique identity. The need for semantic integrity requires model editors to work on top of a common set of meta-concepts [8] which ensures semantically valid relations among model data.

With MEMOCenterNG, we present a comprehensive modeling environment which integrates an extensible set of modeling languages on the basis of

[1] MEMOCenterNG is named after the MEMO enterprise modeling method [7]. "NG" stands for "Next Generation", MEMOCenterNG is the successor to an earlier MEMOCenter prototype application.

a consolidated common meta-model API. The provided modeling languages are suitable to express knowledge about a system on three different levels of abstraction from multiple interdependent perspectives.

The available levels of abstraction provided by built-in modeling languages in MEMOCenterNG are:

1. A meta-modeling layer which allows for creating new modeling languages and corresponding diagram editors, and also internally defines other built-in modeling languages.
2. A set of built-in modeling languages for modeling the organizational environment of involved actors, their goals, behaviour and involved resources. This organizational abstraction layer is provided by domain-specific modeling languages of the MEMO modeling method [7].
3. An implementation model layer which offers general purpose modeling languages to express a software system's inner perspective in terms of, e.g., classes, attributes and relationships.

Figure 1 gives an overview on the component architecture of MEMOCenterNG, as it evolves from applying the MEMO Meta Modeling Language (MML) for language creation.

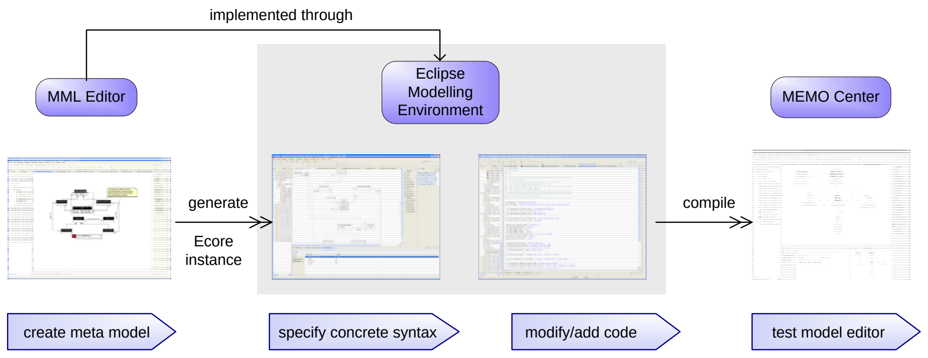


Fig. 1. MEMOCenterNG component architecture

MEMOCenterNG contains a total of 11 modeling languages on all three abstraction layers. This set is composed of 7 domain specific languages as proposed by the MEMO method [7], made up of *Activity Diagrams*, *Allocation Diagrams*, *Organization Diagrams*, *Process Control Flow Diagrams*, *Process Decomposition Diagrams*, *Resource Diagrams* and *Strategy Diagrams*. 3 implementation-level generic purpose modeling languages covering *Data Flow Diagrams (DFD)*, *Entity-Relationship-Models (ERM)* and class-diagrams of the *Unified Modeling Language (UML)*, and the *MEMO Meta-Modeling Language (MML)*. These initially available languages can be enhanced by any number of specific modeling languages created with the MML. Supplemented by third-party tooling components, models can be analyzed, model-transformations can be carried out, and code-generation mechanisms are available to generatively create software in a model-driven development process. Altogether, MEMOCenterNG forms a comprehensive environment for modeling on multiple layers of abstraction from multiple perspectives, store and manage interrelated models in a common environment, and further process model data inside the same platform. In model-driven software development projects, generated artifacts can furthermore be edited, compiled and packaged within the same tool. The platform is based on the Eclipse [5] environment which can additionally be enhanced by a multitude of third-party supplementary components for software development.

By means of the included MEMO Meta Modeling Language (MML, [8]), new modeling languages can efficiently be created, and appropriate diagram editors for using the languages are automatically created from MML meta-models (see sect. 3).

Since all generated components run in the same environment as MEMOCenterNG, models and generated software components may even reflectively refer to MEMOCenterNG's models. This allows models to be integrated into software components at run-time as part of a self-referential information system architecture [9].

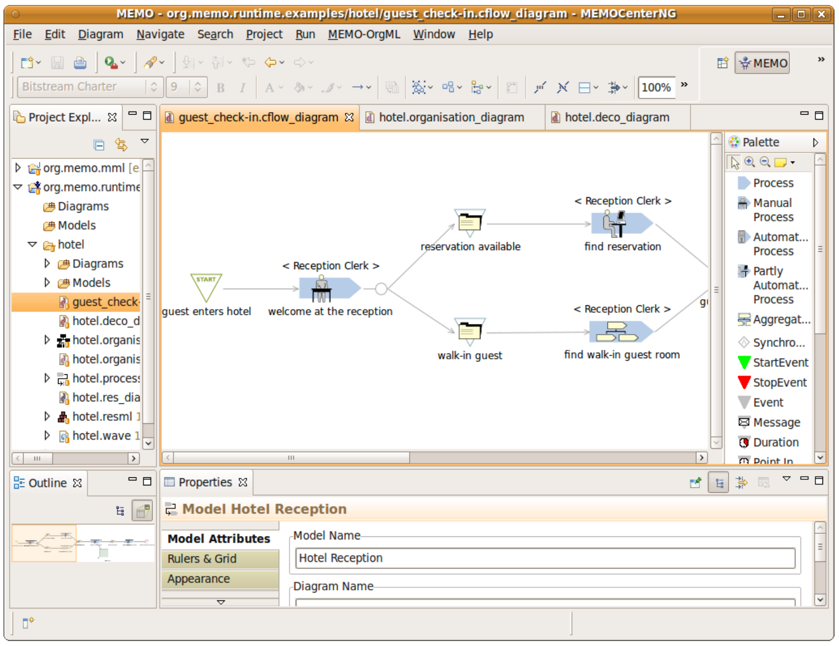
2 Languages for organization modeling

MEMOCenterNG comes pre-packaged with interrelated modeling lang-uages of the MEMO OrgML [7] that cover organization modeling. To express the outer context of an incident to be modeled, organization modeling languages are used to capture people's goals, their behavior, structure and resources of the modeled organization. Such types of models play an important role in enterprise modeling (EM, [7]), especially to express types of business processes that are performed in an organization with their corresponding responsible actors and involved resources. Besides business contexts in a narrow sense, any organizational setting and projects with shared goals among groups of people can generically be expressed with the semantic modeling concepts provided by these modeling languages.

The languages included in MEMOCenterNG are the *Organization Diagram* language for modeling organizational structure, and the *Process Control Flow Language* [7], which allows to express semantically rich process model descriptions of business processes and other methodical procedures in organizations. The Process Control Flow language is enhanced by the *Process Decomposition Language* which is used for specifying static decomposition relationships among process steps, i.e., expresses which process steps are further described by more fine-grained process models. Finally, the *Strategy Diagram* and *Activity Diagram* languages for expressing strategy, goals and actions from a high-level strategic view are part of MEMOCenterNG.

To model physical and non-tangible resources in business contexts, the *ResML* [13] is included in the set of modeling languages, accompanied by the *Allocation Diagram* language which is responsible to express the mappings between process steps and resources.

Figure 2 shows an example Process Control Flow model edited in MEMO-CenterNG.


MEMO - org.memo.runtime.examples/hotel/guest_check-in.cflow_diagram - MEMOCenterNG

Model Hotel Reception

Model Attributes

Model Name: Hotel Reception

Diagram Name:

Fig. 2. Example *MEMO Process Control Flow* model

3 MEMO Meta-Modeling Language MML

The *MEMO Meta Modeling Language* (MML, [8]) included in MEMOCenterNG has especially been designed to efficiently enhance existing languages, and to automatically generate deployable diagram editors from meta-model descriptions of modeling languages. The MML puts special focus on interlinking multiple modeling languages. Concepts from other, previously existing MML models, can be referenced. Every concept is classified by a unique graphical symbol which indicates to which language it belongs.

Full-featured diagram editors can automatically be generated by a single mouse-click from MML models within the user-interface of MEMOCenterNG's MML editor. These generated diagram editors are fully downwards-compatible to the Eclipse Modeling Framework (EMF, [17]) and Graphical Modeling Framework (GMF, [10]) components, which allows for applying any additional EMF / GMF technology component to MML-specified model editors and corresponding model instances, and also use model-instances for analysis, transformation and code-generation (see sect. 5).

Figure 3 gives an example of an MML model edited in MEMOCenterNG.

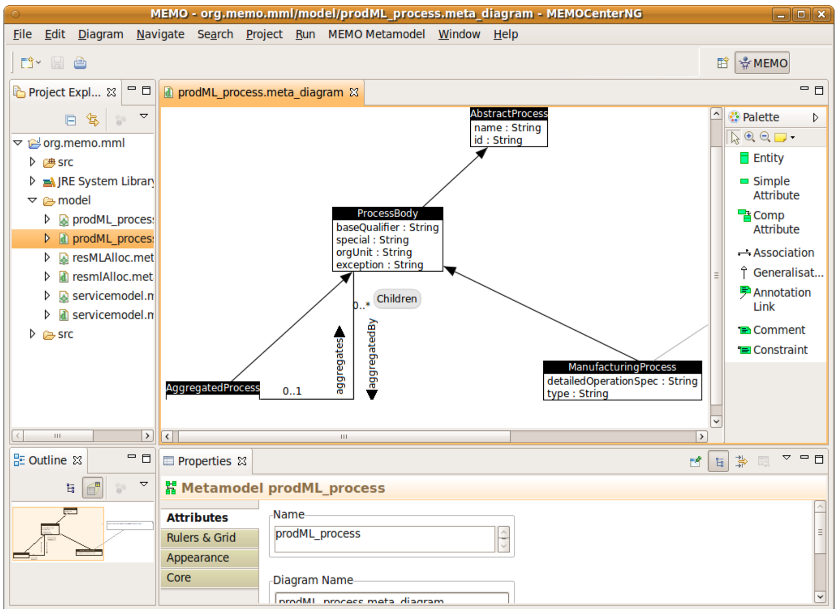


Fig. 3. Example of an MML model edited in MEMOCenterNG

In addition to prevailing meta modeling concepts, the MML features a few concepts that account for specific requirements. The concept of an intrinsic feature, which is similar to the concept of a clabject [1] accounts for a well-known problem: The specification of a meta concept refers to features of corresponding types. However, sometimes a type implies the existence of features that apply to instances only. For example: The specification of the meta concept "Process" may include attributes such as "maximum execution time", which apply to a corresponding type, e.g. "Order Management". In addition to features on this level, it is intrinsic to every process type that its instances have certain attributes such as "started at" or "terminated at". Attributes that are marked as intrinsic must not be initialized on the type level. The code generated from meta types that include intrinsic features accounts for this constraint. A model editor presents intrinsic features of model elements with their types. It does not, however, allow for initializing them. Hence, intrinsic features are intended to be used only after a further step of instantiation. This could apply to code that is generated from a model. In addition to that, intrinsic features can be used for visualizing instance-level data using the same graphical representation as on the conceptual level. Take, for instance, an organizational chart: On the conceptual level, it would include position types only, not the names of the corresponding employees. For some purposes, it might be more useful to represent an organizational chart with the actual people that are in charge of the presented positions. In this case, the representation of the model needs to be extended in order comprise an additional layer of abstraction, in this case the employees that hold the presented positions. Figure 4 shows an excerpt of a meta model of the MEMO OrgML and corresponding instantiations on the type and the instance level.

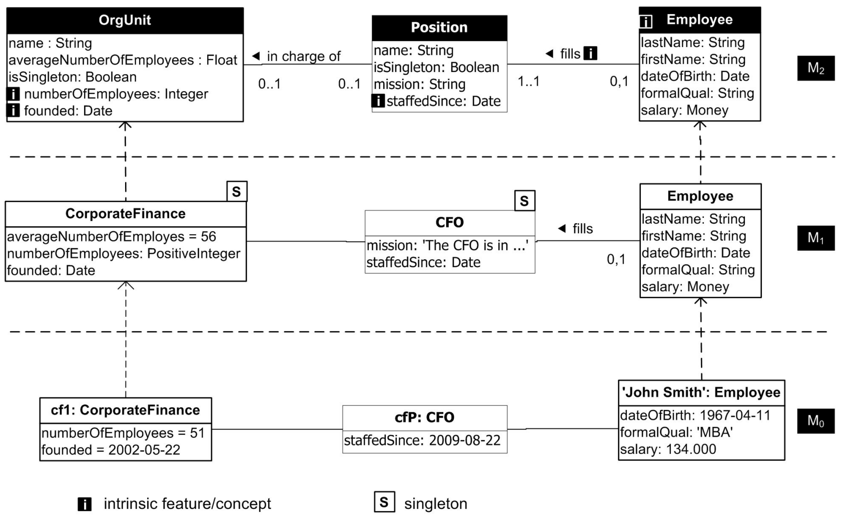


Fig. 4. Illustration of intrinsic features

Figure 5 displays an excerpt of an organizational chart that combines type level information (on positions) with instance level information (on particular employees).

The examples in Fig. 4 and Fig. 5 indicate that values of some attributes – either on the type level such as average number of employees or on the instance level such as the name of an employee might be obtained from external sources, e.g. a database. If this is the case, an attribute of a meta concept can be characterized as derivable. In the current implementation, the tool allows for marking attributes as derivable.The syntactic notation symbol of a small “d” in a square marks an attribute as derived, which is shown in Fig. 6 together with notation symbols for intrinsic attributes.

Two types of derivable attributes can be distinguished. An attribute may either be derived internally from another value or a calculated value expression in the same model instance or a referenced instance. To specify such an inter-

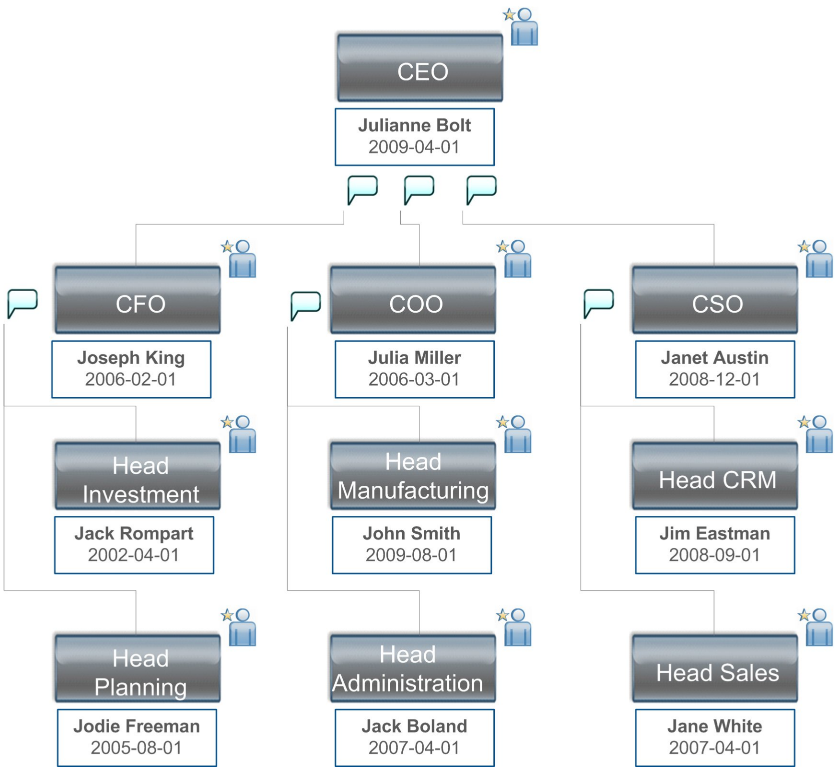


Fig. 5. Excerpt of organizational chart covering two levels of abstraction

nal derivation, an expression in the Object Constraint Language (OCL) can be attached to a derived attribute in MEMOCenterNG's MML editor. This expression will be evaluated on the current model element instance when the value of the derivable attribute is read.

Deriving attributes from an external source, e. g. a database, is the more general second case of realizing derivable attributes. MEMOCenterNG allows to declare database references which specify relational databases as external data sources. These may be relational SQL databases or any kind of database system with a textual query language, as long as a standard-compliant JDBC [12] driver class for the database system is available. When attributes are declared to be derivable from an external data source, a query language statement can be attached to the attribute declaration in the MML editor, which will be used to query a referenced database at run-time when the externally derivable attribute is accessed for reading. The result of the database query statement will be delivered as the value of the externally derivable attribute.

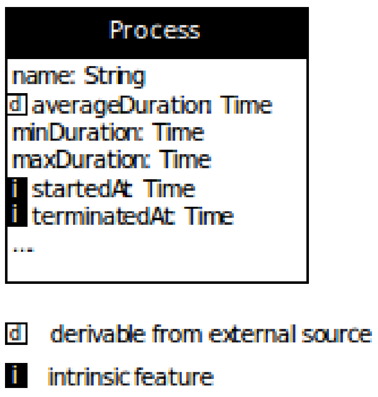


Fig. 6. Illustration of derivable and intrinsic features

The constraints that apply to the meta meta model are defined through OCL expressions in order to foster the creation of a tool for editing meta models that automatically ensures semantic validity to the greatest extent possible.

MEMO features an extensible set of integrated domain-specific modeling languages. Language integration is accomplished through the common metamodeling language MML. Further domain-specific languages can be added by specifying additional meta models, which are integrated through concepts they share with existing languages. The abstract syntax and semantics of the MML are defined with a meta meta model. The concrete syntax was specified to allow for clearly distinguishing meta models from similar models on the object level, such as UML class diagrams. Fig. 7 illustrates the MEMO language architecture and its relationship to MEMOCenterNG.

Semantic integrity is ensured through the use of domain specific modeling languages. Applying general purpose modeling languages for this task would be insufficient, since only domain specific languages allow to incorporate knowledge about common principles of the modeled domain already in the phase of language design. During language design, general principles of the modeled domain are carefully analyzed by scientific researchers, who take decisions which knowledge about the domain to incorporate into the languages. The resulting language concepts provide a high degree of methodological guidance for grasping knowledge about organizations, while at the same time they provide the flexibility for expressing any possible real-world constellation that characterizes individual enterprises and organizations.

An example excerpt of an MML meta model that specifies an enterprise modeling language for organization modeling is shown in Fig. 8.

4 Languages for implementation-level modeling

In order to take in a traditional modeling view on a low abstraction level, MEMOCenterNG contains three classical system modeling languages, *Entity Relationship Models* (ERM, [3]), *Data Flow Diagrams* (DFD, [4]) and object oriented class diagrams from the *Unified Modeling Language* (UML, [2]). Selected concepts of these languages can be referenced from elements in organization models to trace implementation details from a high real-world abstraction level down to technical details.

Each of the implementation-level modeling languages comes with pre-packaged analysis and code-generation functionality. The ERM model editor allows to generate a relational database schema from ERM models as a sequence of executable SQL data declaration language (DDL) statements, which subsequently may be executed from inside the MEMOCenterNG platform to deploy an initial database. The DFD editor comes with basic analysis capabilities, and from UML class diagrams, the source code for Java classes can be generated. The UML class diagram editor component is integrated from one of the authors' Web Application Visual Environment project (WAVE, [16]).

Together with the software development features of the underlying Eclipse [5] platform, MEMOCenterNG forms a fully integrated model-driven software development environment. An example of integrating between an organization model and an implementation model is displayed in Fig. 9.

5 An Example Model-Driven Software Development Scenario

All components of MEMOCenterNG are based on the Eclipse Modeling Project [10] components EMF and GMF, and make use of the Ecore language [6] through the MML. As a consequence, supplementary components that exist for the Eclipse Modeling Project can be applied in MEMOCenterNG. This allows

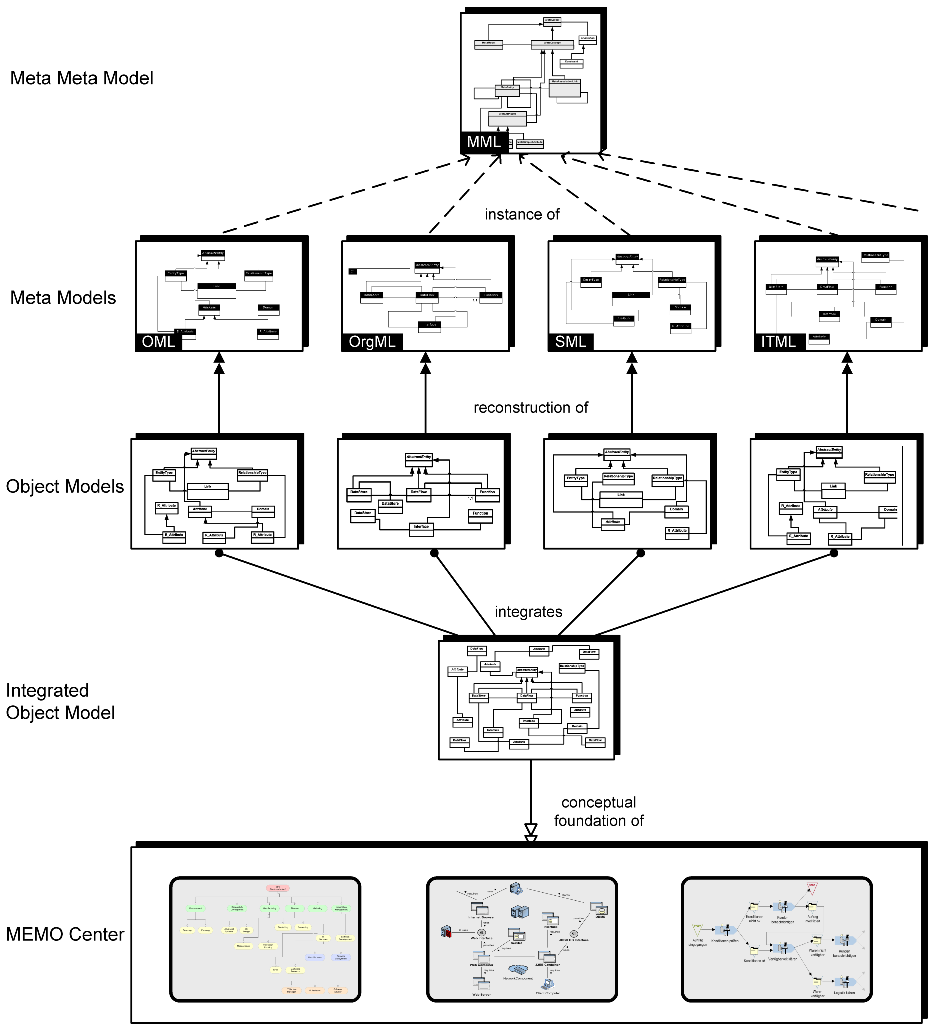
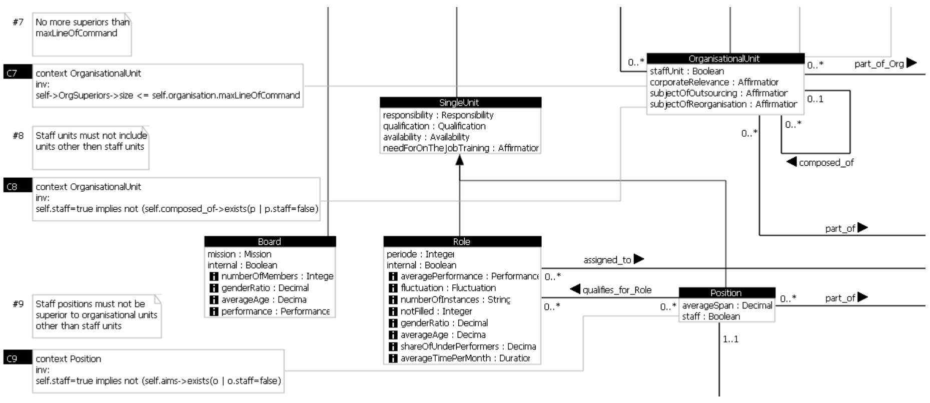


Fig. 7. MEMO Language Architecture

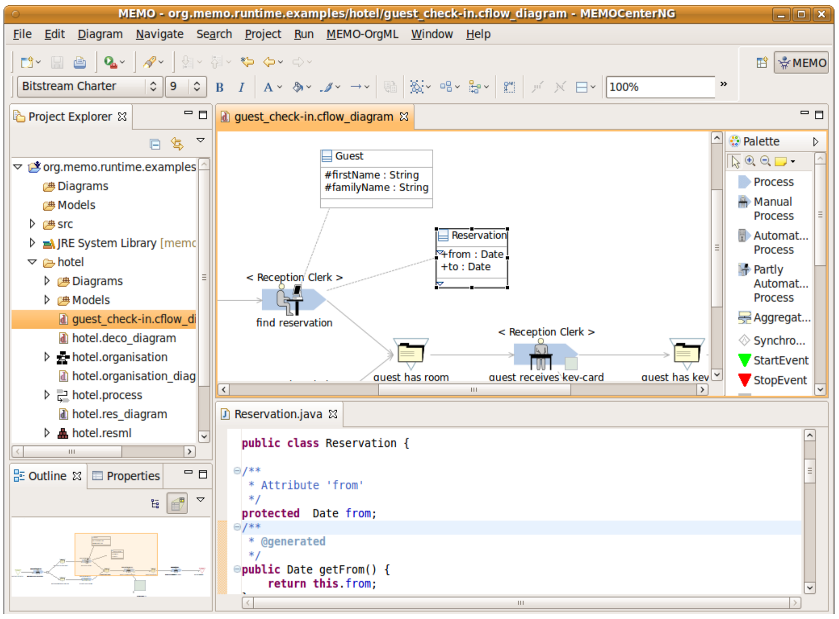

#7
No more superiors than maxLineOfCommand
C7
#8
Staff units must not include units other then staff units
C8
#9
Staff positions must not be superior to organisational units other than staff units
C9
SingleUnit
responsibility : Responsibility
qualification : Qualification
availability : Availability
needForOnTheJobTraining : Affirmation
OrganisationalUnit
staffUnit : Boolean
corporateRelevance : Affirmation
subjectOfOutsourcing : Affirmation
subjectOfReorganisation : Affirmation
part_of_Org
composed_of
part_of
Board
mission : Mission
internal : Boolean
numberOfMembers : Integer
genderRatio : Decimal
averageAge : Decimal
performance : Performance
Role
periode : Integer
internal : Boolean
averagePerformance : Performance
fluctuation : Fluctuation
numberOfInstances : String
notFilled : Integer
genderRatio : Decimal
averageAge : Decimal
shareOfUnderPerformers : Decimal
averageTimePerMonth : Duration
assigned_to
qualifies_for_Role
Position
averageSpan : Decimal
staff : Boolean
0..*
0..1
1..1

```
context OrganisationalUnit
inv:
self->OrgSuperiors->size <= self.organisation.maxLineOfCommand
```

```
context OrganisationalUnit
inv:
self.staff=true implies not (self.composed_of->exists(p | p.staff=false))
```

```
context Position
inv:
self.staff=true implies not (self.aims->exists(o | o.staff=false))
```

Fig. 8. Meta model excerpt created with the MML



```
public class Reservation {

/**
 * Attribute 'from'
 */
protected  Date from;
/**
 * @generated
 */
public Date getFrom() {
    return this.from;
```

Fig. 9. Integrating between an organization model and an implementation model

for seamlessly integrating supporting technologies such as the Xtend / Xpand transformation languages [6], or other code generation techniques such as Velocity [15] or the QVT [14] language. By default, the Eclipse components for executing Xtend / Xpand transformations are included in the MEMOCenterNG environment.

With this environment available, ambitious model-driven software development projects become possible to be realized on top of MEMOCenterNG as a common development platform. As an example for a possible future approach of generating self-referential software applications from enterprise models, a prototypical 3-phase model-to-model transformation method has been set up as sketched in Fig. 10.

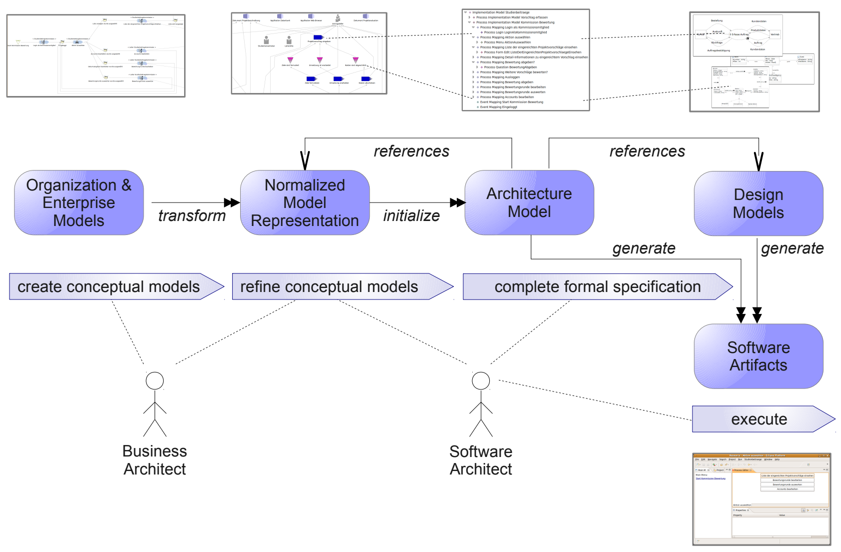


Fig. 10. 3-phase model-to-model transformation method to derive software from enterprise models

Such a sketched approach allows to derive running software systems from enterprise models, by combining conceptual organization models with technical models of the involved information systems, and enrich these models with additional design decisions that are required to specify executable software. Since enterprise models describe the conceptual architectures of organizations and their information systems, enterprise models needs to be conceptually enhanced by implementation-specific detail information required to derive running software from the enterprise models. Such an enrichment can be performed in a collaboration of human business architects and software system architects with support of a 3-phase model-transformation method. In such a setting, a first model-to-model transformation generates a *normalized model* representation from conceptual business processes and other organization model perspectives hat are contained in the enterprise model. Such a normalized representation can, e. g., be given in the Business Process Modeling Notation (BPMN, [11]) language.

Since conceptual business process models differ in the level of abstraction compared to executable software descriptions, there cannot be a fully automatic transformation that creates executable software descriptions from business process models. This is why a refinement phase of the normalized representation is required after the first model-to-model transformation. During this refinement phase, business architects and software system architects cooperate closely to gain a common understanding of the underlying business processes and the desired software requirements. The cooperation between these groups of stakeholders is supported by the visual notation of the underlying model representations.

To further provide software architects with methodical support for enhancing the normalized business process models with implementation-specific details, a second model-to-model transformation is consulted by the sketched method. This is used to add additional information to the process models, which are technically required to execute software e. g. as workflow models by an automatic workflow engine, or as compiled software from generated source code. Using the second model-to-model transformation, a default *architecture model* is derived. The architecture model is formulated in a domain specific modeling language which has been designed to associate additional detail information with elements of the conceptual models. This information is needed to either generate or interpret running software derived from enterprise models. The second model-to-model transformation creates a default architecture model which contains references between all parts of the enterprise model that need additional design decisions to be taken in order to derive running software. References to implementation specific information may be initialized with heuristically derived default values, or they may be placeholders that are required to be manually filled in by software architects. In a third model-transformation step, artifacts for executing the software system are generated from the architecture model and the models it references. This generation phase may create complete sets of source-code files in a target programming language, or alternatively may output configuration documents for an execution mechanism such as a workflow engine, to let the resulting software be interpreted from the generated artifacts.

The sketched approach delivers a method for extracting as much knowledge as possible from enterprise models to prepare necessary design decision that are required for creating running software systems out of multi-perspective enterprise models using the MEMOCenterNG platform as integrated development environment (IDE). While naturally not all transformation steps to a running software system can be performed automatically, this methodical approach allows for efficiently guiding developers through the process of taking relevant design decision and provide manually developed source-code where necessary. In addition, the sketched approach provides a mechanism to decide whether all required decision have yet been taken, i. e., it can automatically be decided if the architecture model is complete with respect to all conceptual components introduced by the enterprise model on the one hand, and associated design decisions and technical artifacts from design models such as UML class-diagrams or data-flow diagrams on the other hand. Missing design decisions resulting from an incomplete architecture model or from modifications that have been made to the underlying models thus can be detected automatically, and appropriate tool support for a model driven development method in the proposed manner would be able to automatically guide developers through a sequence of steps required to complete of an architecture model. The approach thus provides a highly supportive procedure for an advanced model driven software development method.

6 Availability

A beta-version of MEMOCenterNG is available for download at `http://www.wi-inf.uni-duisburg-essen.de/FGFrank/download/memo/`. The current distribution version is still an intermediate release under development. Until a more mature version will be released under an open-source license, please request password information from the authors to download the current version.

7 Conclusion and Outlook

We have presented a modeling tool that offers multiple modeling languages in an integrated environment, based on a meta-model supported language architecture and enriched by an easy-to-use meta-model editor for specific language enhancements.

A common language architecture ensures the semantic integration of concepts across multiple languages. By incorporating meta modeling and the creation of new modeling languages into the feature spectrum of the modeling tool, a new degree of flexibility and adaptability to future requirements is achieved by the application. This makes MEMOCenterNG a comprehensive, full-featured integrated modeling environment for a broad range of modeling projects, including model-driven software development approaches. Currently, the tool is successfully used for teaching purposes and has prototypically been tested in medium-sized business projects carried out in cooperation with our research group.

Apart from refining and extending existing model editors, our future work on MEMOCenterNG is focusing on two topics. Currently, the tool supports creating, analyzing and transforming models. It does not, however, guide these activities with respect to certain objectives. For this purpose, one would need to somehow represent a modeling method, i. e. to augment modeling languages with corresponding process models. With regard to the wide range of different projects to be supported, it seems adequate to provide support for method engineering, i. e. to add features that allow for the convenient and safe customization of methods. For this purpose, we plan to extend MEMOCenterNG with a language for creating process models (not to be confused with business process models). To prepare a project, a user could use a corresponding model editor, create a process model and assign the required modeling languages to each process stage. In a further step, the model of the resulting method could serve as a conceptual foundation for a customized project management tool. The second topic relates to the observation that conceptual models are mainly used during build-time only. However, they would be useful at run-time, too: Augmenting an information system with an enterprise model would enrich the system with a representation of the action system it is embedded in. That would not only provide a powerful foundation for knowledge management and decision support, it would also promote IT business integration. To express that the use of enterprise models at run-time enable enterprise software to not only refer to their conceptual foundation but also to their surroundings, we speak of self-referential enterprise systems [9].

References

1. Atkinson, C. and Kühne, T.: Reducing accidental complexity in domain models. In: Software and Systems Modeling. Online First, June 2007
2. Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Modeling Language Reference Manual. Addison-Wesley, Reading (1999)
3. Chen, P.: The Entity-Relationship Model – Toward a Unified View of Data. In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN 0362-5915, pp. 9–36 (1976)
4. DeMarco, T.: Structured Analysis and System Specification. Prentice Hall, Upper Saddle River (1979)
5. Eclipse Foundation: Eclipse Platform. `http://www.eclipse.org/` (2010-07-04)
6. Efftinge, S., Friese, P., Haase, A. and others: openArchitectureWare User Guide. `http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/index.html` (2008)
7. Frank, U., MEMO: A Tool Supported Methodology for Analyzing and (Re-)Designing Business Information Systems. In: Ege, R., Singh, M., Meyer, B. (eds.): Technology of Object-Oriented Languages and Systems, pp. 367–380 (1994)
8. Frank, U.: The MEMO Meta Modeling Language (MML) and Language Architecture. ICB-Research Report No. 24, Institute for Computer Science and Business (ICB), University Duisburg-Essen (2008)
9. Frank, U., Strecker, S.: Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems. ICB-Research Report No. 31, Institute for Computer Science and Business (ICB), University Duisburg-Essen (2009)
10. Gronback, R. C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Longman, Amsterdam (2009)
11. Grosskopf, A., Decker G., Weske, M.: The Process: Business Process Modeling using BPMN. Meghan Kiffer Press, Tampa (2009)
12. Sun Microsystems: Java Database Connectivity (JDBC). `http://http://java.sun.com/javase/technologies/database/index.jsp` (2010-07-04)
13. Jung, J.: Entwurf einer Sprache für die Modellierung von Ressourcen im Kontext der Geschftsprozessmodellierung. Logos, Berlin (2007)
14. Object Management Group: Meta Object Facility Query / View / Transformations. `http://www.omg.org/spec/QVT/1.0` (2008)
15. Apache Software Foundation: The Apache Velocity Project. `http://velocity.apache.org/` (2010-07-04)

16. Gulden, J.: Web Application Visual Environment (WAVE). `http://wave.berlios.de/` (2010-07-04)
17. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework (2nd edition). Addison-Wesley Longman, Amsterdam (2009)