AMABULO - A Model Architecture for Business Logic

Tobias Brückmann, Volker Gruhn
University of Leipzig
Applied Telematics / e-Business Group
Klostergasse 3, 04109 Leipzig, Germany
{brueckmann, gruhn}@ebus.informatik.uni-leipzig.de

Abstract

Models are widely used for communication and documentation purposes. They also tend to be used as parameters for code generation. Because these models have to be complete, consistent and correct we have to support modellers in keeping their models clean. The modeller should have the choice to select the preferred view and the needed level of detail for his modelling and model maintenance tasks. This paper proposes a model architecture named AMABULO for a model driven development process of business logic for information systems. The model architecture consists of a meta model, corresponding visual diagrams and an interchange format. With the use of AMABULO the development process for business logic is supported from analysis until code generation.

1 Introduction

Models are becoming more and more important artifacts in software development processes. From the early days of programming, software developers have to build and maintain software systems. To handle the dependencies between parts of complex systems, visual models are used as an abstraction of the program code for documentation and communication purposes. With the help of these models, selected aspects and the underlying structure can be easily depicted without the need to read the whole source code of a system. At least since the idea of a model driven development process is discussed and supported by the Model Driven Architecture (MDA) [1] of the Object Management Group (OMG), the use of models in software processes changes from a communication and illustration artifact to a detailed specification artifact used as input for code generation. If models are used as parameter for code generation, they have to be complete, consistent and correct in relation to the software requirements.

Comparable to the program code, the complexity of models used in a model driven development process increases and the modellers have to create and maintain large software models with hundreds or thousands of elements. The need for such large models is due to the level of detail of models which is required for code generation purposes. Because the manual creation and maintenance process of large models is error-prone and the modelling errors directly result in code errors, we have to support the user in keeping his models clean. For this reason, different views in different layers of abstraction to a model have to be supported. The modeller has the choice to select the preferred view for his modelling and model maintenance tasks [2].

Many modelling concepts for any aspect of software models are developed and matured. Structural modelling languages are for example Entity-Relationship-Diagrams [3], different UML structural diagrams [4] such as class diagrams, component diagrams, and deployment diagrams. Languages for modelling systems behaviour are UML behaviour models, Petri nets [5], process modelling languages such as BPMN [6], UML activity diagrams and EPC [7], state based modelling concepts such as labelled transition systems (LTS) and modelling languages such as UML state diagrams and scenario based modelling languages such as Message Sequence Charts (MSC) [8] or UML sequence diagrams.

The use of only one modelling concept often meets the requirements. This applies to a model driven development approach which only generates selected parts of a system, such as the data base model or parts of the user interface. A huge number of modelling tools and academic research projects support the generation of program or XML code for different domains.

But for a more holistic approach to model driven development, a model architecture is needed, which provides different well-defined views to a model and a well-defined integration of these different views in one meta model. A model driven development process of business logic requires the use of different views to a system. A business process usually contains several execution paths, which are

determined either by users or automatically. The structure of business objects have to be refined using structure models. Also internal states of business objects have to be considered when determining an execution path through a process and when executing state transitions.

The most common approach of an integrated view onto the same model is the unified modelling language UML. The UML 2.1 standard document defines 13 different diagram types which are defined on the same meta model [1]. But the UML diagrams provide neither a formal definition nor a well-defined semantic integration between them. Several academic work groups, such as the UML 2 Semantics Project [9] and publications as [10] propose formal foundation parts of the UML standard, but the UML definition itself does not contain a clear semantics definition.

Not just the semantics of single diagrams are important, but the integration of different diagrams as views onto one model and their semantic dependencies must also be well-defined. There are several approaches to connect different modelling concepts at a semantic level which are discussed in the subsequent section.

In the following sections, after the discussion of related approaches (Sect. 2), we present our model architecture (Sect. 3). The detailed description of the meta model (Sect. 4) and the proposed diagrams (Sect. 5) are followed by the model interchange format (Sect. 6). We conclude with a discussion and an outlook on future research opportunities (Sect. 7).

2 Related Work

The most commonly known and well-understood relation between models is the connection of class-based models to state-based models, where the states are values of class attributes and the transitions from state to state define possible changes from attribute values.

Many research activities are going on in the interaction between scenario modelling and state modelling. In [11], a survey of 21 different model synthesis approaches between state-based and scenario-based models is presented and discussed. All these approaches are developed in the field of reactive systems such as pump controller software and other technical controllers. These approaches are rather unsuitable to support the development of interactive information systems, because a scenario usually contains only one execution path through a process. For information systems complex business processes and the influence of the user interaction to execution paths have to be modelled. Each possible path requires its own scenario and all scenarios have to be consistent, which results in a huge number of diagrams.

Another approach is to relate process-based models to state-based modelling with the intention to provide consistency between process models and object life cycles using the states as constraints for the execution of actions [12]. The proposed mapping from the process model to a state diagram does not support different processes and business objects in the same model. For the support of a development process, we cannot constrain the numbers of processes or business objects. An additional mapping from different object live cycles to a process model was presented in [13].

The key concept of all these approaches is the connection between modelling concepts from meta model of a source diagram to the meta model of the target diagram. This has some disadvantages. Between each of the linked diagrams a transformation for each diagram and for each direction has to be specified. For more than three views onto a model, the numbers of needed mappings will grow enormously. And if the model information is distributed across differed diagrams, a model transformation process has to collect all the needed information from each single diagram. An approach that tries to tackle these disadvantages is the use of a semantic meta model with defined mappings to concrete diagrams. In [14], a set of diagrams and a mapping to a meta model is proposed for formal validation and code generation purposes. But the provided modelling concepts and notation elements do not support the modelling and refinement of business logic using process and state diagrams.

These approaches do not provide a model architecture which supports a model driven development process for business logic. For a structural model and transformation to program code, only one modelling concept such as entity-relationship or UML class diagrams is needed. These diagrams support a modelling concept that can be consistently refined from analysis to implementation.

If we want to support behaviour modelling and generation of program code for systems behaviour, we have to consider the business process model as a result of the analysis and provide the possibility to refine this model. We have to consider business objects, which are manipulated by actions of the process and which can influence the path through a process, and we have to consider the difference between actions that are executed by a human user and actions which are automatically executed. Generation of source code from a behaviour model demands to connect the generated source code (representing business logic) to the persistency and user interface layer.

3 Approach

We propose a model architecture to support the business logic development for information systems in a model driven development process from analysis to implementation. Satisfying the following requirements, the whole design process from analysis time with modelling business processes until the refined model, which is detailed enough for the code generation should be supported by:

- Providing a meta model that defines all required model elements, which can be addressed by a model transformation language,
- providing different views onto the model for structural and behavioural modelling, providing well-defined integration of the different views in the meta model,
- providing well-defined connections of the diagrams at the level of their notation elements, and
- providing a defined interface to transformation engines, which allows our model architecture to be the source or the target model of a model transformation.

In this sense, a "model architecture" consists of a meta model, which provides the semantics of core model elements, a set of diagrams providing different views onto a model and a model interchange format as an interface to model transformation engines. Each diagram in a model architecture is defined by its abstract syntax, which maps the meta model to diagram elements, and the concrete syntax, which defines the visual representation of a model element.

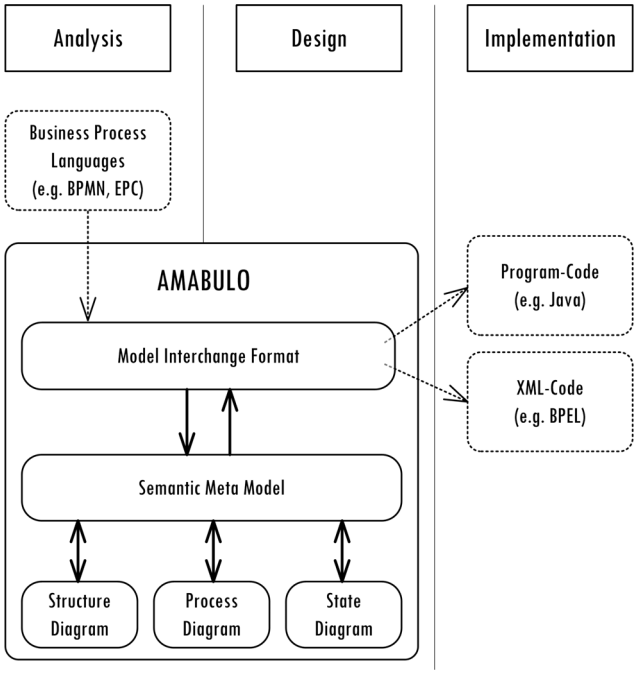


Figure 1. Overview over Model Architecture

Figure 1 gives an overview of the model architecture named AMABULO that we propose to support business logic. The supported phases in the software development are analysis and design. The dashed lines in Figure 1 show an example for a possible transformation at analysis time, where a business process model is the source and our model architecture the target of the transformation. Another possible transformation uses our meta model as source to generate program code at implementation time.

4 Meta Model

The proposed meta model shown in Figure 2 comprises three modelling concepts, which are described in the following subsections. Elements for process modelling are needed to define structured sequences of functions describing the business logic. The elements for business object modelling are needed to define business objects and their properties. These objects are used to control the flow through a process. They can be manipulated by functions. The elements for state modelling are used to model the object life cycle.

We reduced the elements of the AMABULO meta model to a minimum set that still contains all information needed for code transformation. Modelling elements providing only "syntactic sugar" are not part of the meta model.

4.1 Process Modelling

The meta model elements for process modelling are hierarchically structured using abstract concepts. The key concept "ActivityConcept" is the abstract generalisation of the concepts "Process" and "Function". (An instance of an "ActivityConcept" is called activity in the following transition.) An activity can be executed if defined preconditions are satisfied. Afterwards defined postconditions have to be satisfied. The preconditions as well as the postconditions are modelled as "Constraints". Furthermore, an activity can only be executed if the required input parameters are available. After execution, the specified output parameters are guaranteed. Input and output parameters of an activity are instances of "Parameters".

A succession relation between activities can be defined using the "Succession" element. This is needed to model the order of processes and functions. A succession relation contains one activity as predecessor and maybe several activities as successors. Each succession relation can be refined by a constraint. This constraint, named "guard", is needed to decide at runtime which of the possible successors have to be executed.

The element "Process" is a specialisation of "ActivityConcept". A process is a container for activities and used for structuring the process model supporting unlimited nesting of "Processes". Each process has one or more initial nodes, which denote activities which have to be executed first.

A "Function" is an atomic specialisation of "ActivityConcept", which can be executed. Only a function is able to

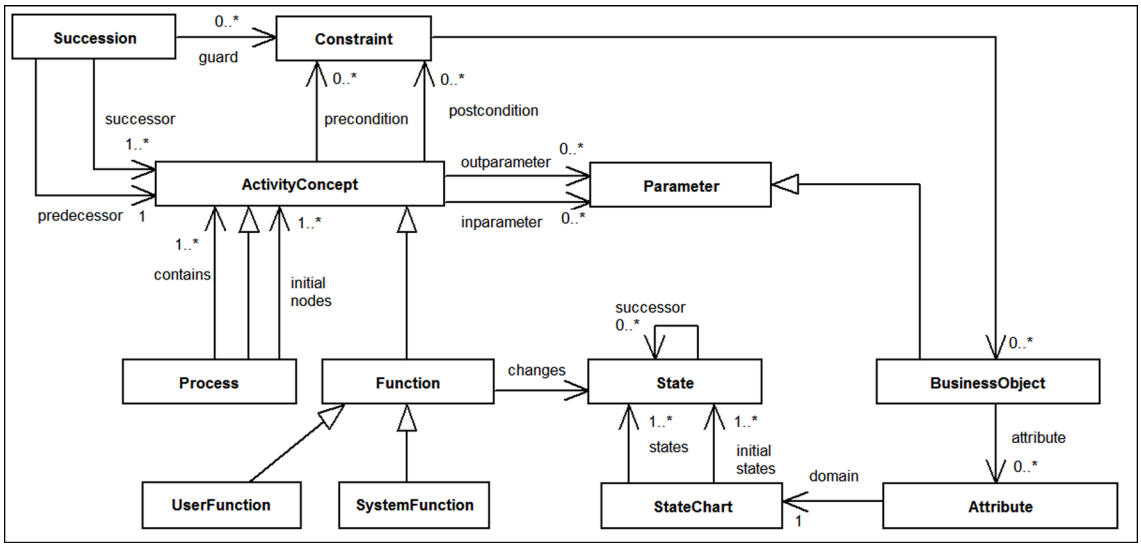


Figure 2. Meta Model

change the state of an attribute of a business object. A function cannot be refined. The meta model distinguishes between automatic functions (of type "SystemFunction") and manual functions (of type "UserFunction"). This distinction is important, since manual functions require the generation of glue code to user interfaces (or even the generation of the user interfaces).

We specialise "ActivityConcepts" into "Processes" and "Functions" to distinguish atomic elements ("Functions") from elements containing one ore more execution paths which have to be calculated at runtime ("Processes"). Furthermore, we allow only functions as atomic "ActivityConcepts" to change states of business objects. These are two constraints of our meta model we use for code generation purposes to have a clear separation of container elements and functions.

4.2 Business Object Modelling

An instance of "BusinessObject" represents a domain object. It can be manipulated by the execution of functions, it can constrain and influence the paths through a process (the state or the value actually can determine which path of the process has to be chosen), and it can directly determine which activities have to be executed. Each business object can be associated with a number of "Attribute" elements. In principle, all modelling concepts of UML class diagrams can be employed for the business object modelling. In the current version of the AMABULO meta model, we focus only on classes and attributes for reducing the complexity of the meta model, and we focus on the behaviour and the connection between business objects and business processes.

4.3 State Modelling

For state modelling, the AMABULO meta model provides simple modelling concepts. A "StateChart" element represents a simple state chart which is associated with an attribute of a business object. The state chart describes the domain of an attribute with all provided values. A "State" is a part of a state chart with a concrete value in a defined range. The order of the states is modelled using the successor relation between states. Each state can contain a list of possible successor states. If a function tries to change the state of a state chart, the new state can only be a state of the successor list. Comparable to processes, state charts contain a set of initial states. The elements of this set are the possible initial values of the associated attribute.

5 Diagrams

Our model architecture supports three different diagram types, but a later extension of the set of supported diagrams is not excluded. We use diagrams provided by the UML[4] because it is a widely-used software modelling language and supports notation elements for all required modelling concepts. Furthermore, the UML provides extension and restriction concepts (stereotypes and profiles), which allow the definition and use of tailored diagrams. The diagram types namely are activity diagram, class diagram, and state diagram.

Figure 3 displays the list of needed notation elements and their mapping to the meta model. This list contains the notation elements of the diagrams which are directly or indirectly mapped to the meta model. Elements such as pseudo

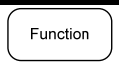
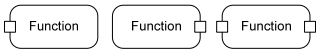
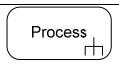
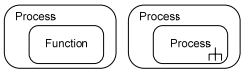
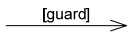
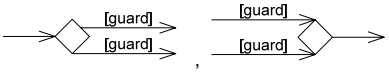
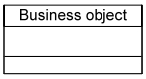
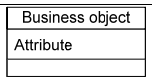
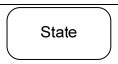
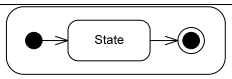
Diagram Type	Used UML Model Elements	Corresponding Element in AMABULO Meta Model	Notation Elements used in UML Diagrams
Activity diagram	Activities:: Action	Function	
Activity diagram	Activities :: Action	Parameter	
Activity diagram	Activities :: Activity	Process	
Activity diagram	Activities :: Activity, Activities :: Action	Process :: contains	
Activity diagram	Activities :: ActivityEdge :: Guard	Constraint	
Activity diagram	Activities :: InitialNode	Process :: initital node	
Activity diagram	Activities :: DecisionNode, Activities :: MergeNode, Activities :: ActivityEdge, Activities :: ActivityEdge :: Guard	Activity Concept :: successors	
Class diagram	Classes :: Class	Business Object	
Class diagram	Classes :: Property	Attribute	
State machine diagram	State Machines :: Pseudostate	State Chart :: initialState	
State machine diagram	State Machines :: State	State	
State machine diagram	State Machines :: StateMachine	State Chart	
State machine diagram	State Machines :: Transistion	State :: successors	

Figure 3. Elements in Visual Model

states or final nodes are only part of the syntax of diagram types.

Figure 4 gives a simple example of a modelled business process, a four eyes principle. For example, an insurance offer can only be accepted if it passes a four eyes principle decision process. This means, only if both deciders accept an offer, it is bound. But if the first decider declines, a second decision is not needed. Only if the offer is accepted, a message is automatically sent by the system. The model contains three different views. An activity diagram provides the process view and the class diagram contains the needed business object "Offer". The third view, a state machine diagram, provides the different states of decision for an "Offer".

5.1 Process Modelling

The UML activity diagram represents the behaviour model. Activities are used to group actions and encapsulate them. This feature is important for refinement purposes during the modelling process. We use UML actions to represent the functions and UML activities to represent a process of the AMABULO meta model. The actions in the diagram can be stereotyped as "user" or "system" actions. This stereotyping is mapped to user and system functions of our meta model. The sequence of functions and processes, defined by the successor relation in the meta model, is modelled using the ActivityEdge, MergeNode, and DecisionNode elements of the activity diagram. The Guard element is used to provide the needed preconditions and postconditions. An example model of the four eyes principle is shown

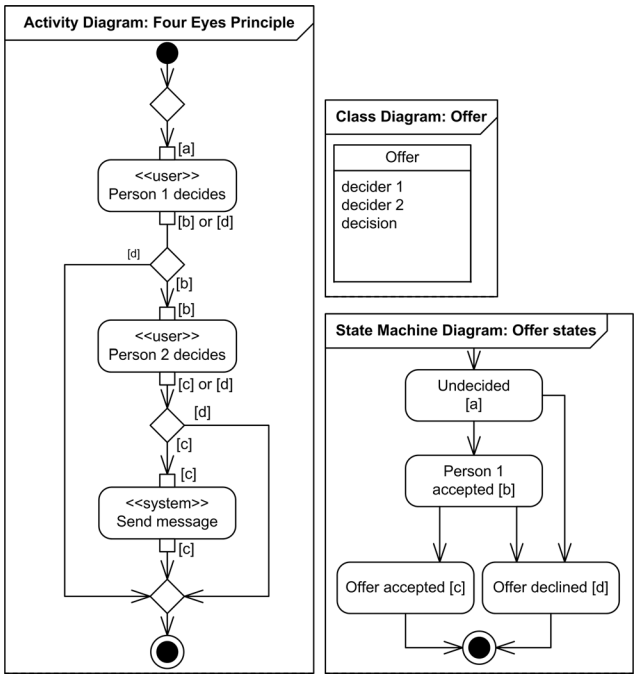


Figure 4. Four Eyes Principle as Example Model

in Figure 4. In this example, each action in the diagram is modelled with labelled incoming and outgoing pins.

These pins form the connection between process modelling and state modelling. Each label on a pin refers to a state of a state diagram. If a state is modelled on an incoming pin, the function or the process only can be executed if an attribute of a business object is in a required state.

5.2 Business Object Modelling

To provide an integrated view of structure and behaviour, business objects are needed as part of the model. A business object is modelled using the concepts of UML class diagrams. Because we want to focus on behaviour modelling, we only use the classes and their attributes for our modelling purposes and leave out further class diagram elements. As an example, we would model the insurance offer of our illustration model as a business object, because this object influences the process flow and can be manipulated by functions of the process. As shown in the class diagram in Figure 4, only three attributes are modelled which are relevant for the four eyes process.

5.3 State Modelling

For each attribute modelled for a business object, the possible values and their dependencies can be modelled using a state diagram. There is no limitation on the number of attributes having an associated state diagram. The initial value is marked as the initial state and the modelled states and their transitions define the possible attribute values and their alteration. As depicted in Figure 4, the example state diagram consists of four states. The initial state is "undecided", the possible successor states are "Person 1 accepted" if the first decider accepts the offer, and "Offer declined" if the first decider declines. From state "Person 1 accepted" the possible following states are "Offer accepted" and "Offer declined", which are both final states.

6 Model Interchange Format

For the use of AMABULO in a model driven development process, the information exchange from AMABULO to a model transformer as well as the information exchange from a modelling environment to the model have to be supported by a defined interchange format. This format contains the elements of the meta model in an XML-based language.

Figure 5 provides an overview of the XML schema using the visual representation of the Altova XMLspy application [15]. The whole schema source code as well as the legend of the schema diagram can be accessed on the project website [16].

The current version of the schema only focuses on the interchange of logical data, but information about the visual alignment of notation elements can be integrated in future versions of the interchange language.

The structure of the XML language follows the structure of the meta model and contains processes, business objects and state charts. Each process contains user functions and system functions as well as processes. Each of them can be identified using a unique identifier. The attributes are part of business objects, and states are part of a state chart. The relations between model elements are realised using identifier references. According to the diagram example in Figure 4, Figure 6 contains the corresponding model described by the XML interchange format, which can be used as the input for transformation engines.

7 Discussion

This paper proposes a model architecture for a model driven development process of business logic for information systems named AMABULO. Our model architecture consists of a meta model and a corresponding visual model. With the use of this model architecture the development process for business logic is supported from analysis until code generation. A key feature of AMABULO is the integration of different views onto a model in a meta model and

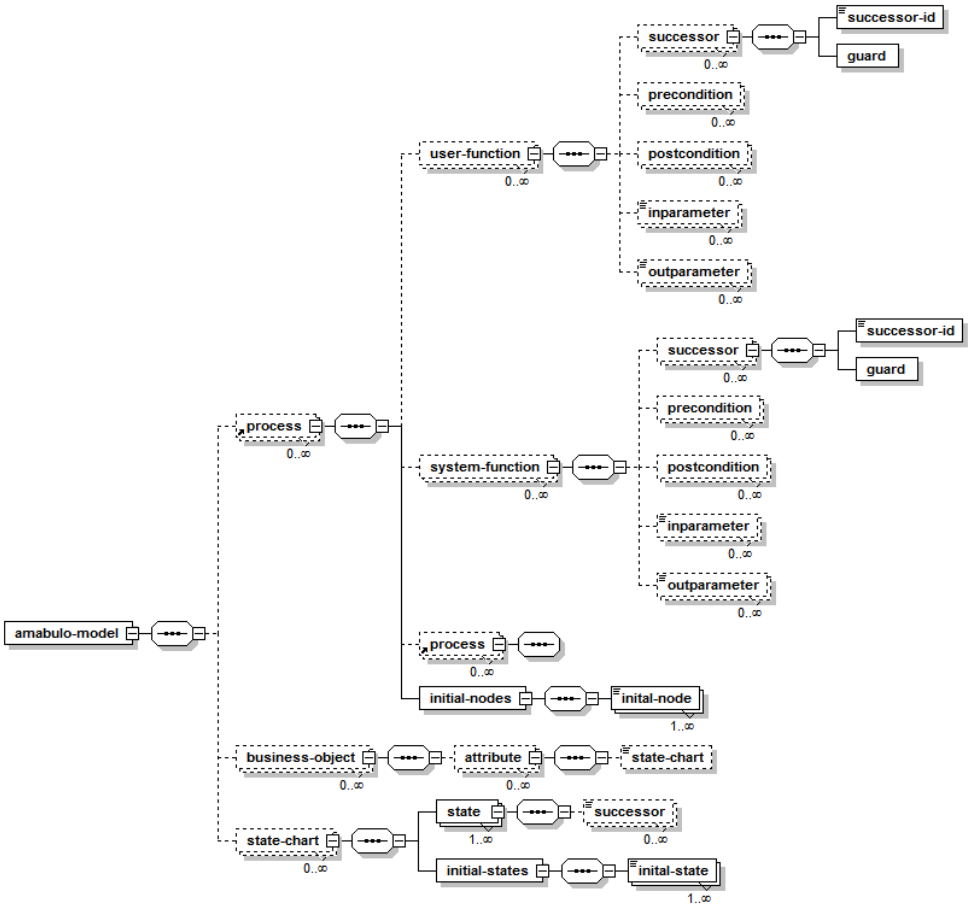


Figure 5. AMABULO Interchange Format

the mapping of the meta model to commonly known modelling elements. The model can be refined step by step until all needed information for code generation is modelled.

Further, we propose an XML interchange format as part of our model architecture. This interchange format proposes an interface from and to AMABULO models which is required for a seamless integration into a model driven development environment. Existing business process models can be transformed to an AMABULO model. Our model provides the features to refine models until the level of detail satisfies the requirements for the generation of program code. If the model is used for communication and documentation purposes only, a transformation to further models is not necessary. But if the XML orchestration of a service oriented architecture (SOA) environment or Java program code is the required output format, a transformation of an AMABULO model to these target models can be defined using the XML interface as an input format.

As part of our future work, we have to describe the elements of the meta model more formally and specify a detailed mapping from the AMABULO meta model to the used diagrams. Furthermore, dependencies of business logic on the user interface and persistency layers have to be discussed in more detail. Another current need is a concrete example of the use of our model architecture in model driven development process including transformation of business models to AMABULO and the generation of program code from an AMABULO model.

References

[1] "MDA Guide Version 1.0.1," Object Management Group (OMG), 12th June 2003 2003. [Online]. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01, as of October 21th, 2007

[2] D. Waddington and P. Lardieri, "Model-centric software development," *IEEE Computer*, vol. 39, no. 2, pp. 28–29, 2006.

[3] P. P.-S. Chen, "The entity-relationship model-toward a unified view of data," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9 – 36, 1976.

```
<?xml version="1.0" encoding="UTF-8"?>
<amabulo-model>
  <process name="four-eyes-principle" uid="four-eyes-principle">
    <user-function name="Person 1 decides" uid="Person-1-decides">
      <successor>
        <successor-id>Person-2-decides</successor-id>
        <guard>offer::decision::b </guard>
      </successor>
      <pre-condition>offer::decision::a</pre-condition>
      <post-condition>offer::decision::b OR offer::decision::d</post-condition>
      <inparameter>Offer</inparameter>
      <outparameter>Offer</outparameter>
    </user-function>
    <user-function name="Person 2 decides" uid="Person-2-decides">
      <successor>
        <successor-id>Send-message</successor-id>
        <guard>offer::decision::c</guard>
      </successor>
      <pre-condition>offer::decision::b</pre-condition>
      <post-condition>offer::decision::c OR offer::decision::d</post-condition>
      <inparameter>Offer</inparameter>
      <outparameter>Offer</outparameter>
    </user-function>
    <system-function name="Send message" uid="Send-message">
      <pre-condition>offer::decision::c</pre-condition>
      <post-condition>offer::decision::c</post-condition>
    </system-function>
    <initial-node>
      <successor>Person-1-decides</successor>
    </initial-node>
  </process>
  <business-object name="Offer" uid="Offer">
    <attribute name="decider 1" uid="decider-1"/>
    <attribute name="decider 2" uid="decider-2"/>
    <attribute name="decision" uid="decision"/>
  </business-object>
  <state-chart name="Offer states" uid="Offer-states">
    <state name="undecided" uid="a">
      <successor>b</successor>
      <successor>d</successor>
    </state>
    <state name="Person 1 accepted" uid="b">
      <successor>c</successor>
      <successor>d</successor>
    </state>
    <state name="Offer accepted" uid="c"/>
    <state name="Offer declined" uid="d"/>
    <inital-state>a</inital-state>
  </state-chart>
</amabulo-model>
```

Figure 6. Four Eyes Principle in XML

[4] *Unified Modeling Language: Superstructure, Version 2.1*, Object Management Group (OMG) Std., 2005. [Online]. Available: http://www.omg.org, as of October 6th, 2007

[5] C. A. Petri, *Communication with Automata*, ser. Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik an der Universität Bonn 2. Bonn: Rheinisch-Westfälisches Institut für instrumentelle Mathematik an der Universität Bonn, 1962, vol. 2.

[6] *Business Process Modeling Notation 1.0 (BPMN)*, Object Management Group (OMG) Std., 2006. [Online]. Available: http://www.bpmn.org, as of October 6th, 2007

[7] G. Keller, M. Nüttgens, and A.-W. Scheer, "Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)"," *Veröffentlichungen des Institutes für Wirtschaftsinformatik, Universität des Saarlandes*, vol. 89, pp. 2–30, 1992.

[8] *Message Sequence Charts (MSC)*, ITU-T Std., 2004. [Online]. Available: http://www.itu.int/rec/T-REC-Z.120/en, as of October 18th, 2007

[9] UML 2 Semantics Project, 2006. [Online]. Available: http://www.cs.queensu.ca/ stl/internal/uml2, as of October 22th, 2007

[10] M. Broy, M. V. Cengarle, and B. Rumpe, "Semantics of UML. Towards a System Model for UML 1-3," Technische Universität München, Tech. Rep., 2006-07.

[11] H. Liang, J. Dingel, and Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches," in *5th Intl.Workshop on Scenarios and State Machines (SCESM)*. Shanghai, China: ACM Press, 2006, pp. 5–12.

[12] K. Ryndina, J. M. Küster, and H. Gall, "Consistency of business process models and object life cycles," in *MoDELS 2006 Workshops*, ser. LNCS, vol. 4364. Genova, Italy: Springer, 2006, pp. 80–90.

[13] J. M. Küster, K. Ryndina, and H. Gall, "Generation of business process models for object life cycle compliance," in *International Conference on Business Process Management (BPM)*, ser. LNCS. Springer, 2007, pp. 165–181. [Online]. Available: http://www.zurich.ibm.com/ ryn/

[14] B. Braatz, "An integration concept for complex modelling techniques," in *Workshop on Multi-Paradigm Modeling: Concepts and Tool (MPM06)*, H. Giese and T. Levendovszky, Eds., vol. 2006/1. Genova, Italy: BME-DAAI Technical Report Series, 2006, pp. 81–93. [Online]. Available: http://avalon.aut.bme.hu/mpm06/Workshopprogram.html

[15] "Xmlspy," Altova, Inc., 2007. [Online]. Available: http://www.altova.com, as of October 12th, 2007

[16] AMABULO, "AMABULO - A model architecture for business logic," 2007. [Online]. Available: http://www.amabulo.com, as of November 4th, 2007