Domenik Bork and Elmar J. Sinz

Bridging the Gap from a Multi-View Modelling Method to the Design of a Multi-View Modelling Tool

Effectiveness of comprehensive modelling methods considerably benefits by the availability of appropriate tool support. However, there is a significant semantic gap between a multi-view modelling method and the design of a corresponding modelling tool. The paper at hand contributes to bridge that gap by means of explicitly focusing the early steps in the design process of a modelling tool. The approach presented here comprises three steps: Starting with (1) a modelling scenario, which centres the human modeller, (2) a multi-view modelling principle and use cases of the tool are derived and (3) the conceptual design of a multi-view modelling tool is specified. The approach is introduced in an abstract manner before it is applied to a concrete scenario. This scenario is outlined for the Semantic Object Model (SOM) business process modelling method, depicting a straight way to the design of an appropriate multi-view modelling tool.

1 Introduction

Creating a comprehensive and useful model is rather art than handcraft. Like an artist, who is going to paint a picture of a landscape, a modeller perceives the real world, delimits the part of the world which should be captured by the model, determines an appropriate perspective on the world, and finally reconstructs the selected part of the world as an artefact.

However, unlike an artist, who may use oil paint, brushes and canvas for making the artefact, a modeller is subject to a more complex setting. This setting is called a modelling method (Ferstl and Sinz 2013) which comprises in particular the following constituents:

- A *metaphor* which outlines the general way of perceiving the real world and designing a corresponding model,
- a *meta-model* which defines the modelling language, as well as associated rules and constraints,
- an *architecture model* which helps to cope with complexity by structuring a model into sub-models, views and layers, and
- a *process model* guiding the steps to be taken while creating the model.

Effectiveness of modelling methods considerably benefits by the availability of appropriate tool support which assists the modeller to create and manage the model, to visualise the model, to share the model with a variety of model users, as well as to utilise and process the model for different purposes. However, there is a significant semantic gap between a modelling method and the design of a corresponding modelling tool. This semantic gap lies in the abstract constituents of the modelling method on the one side and the tool functions on the other side. Its bridging is guided by the goal of harmonising the two sides in the mind of the modeller, i.e., providing the modeller with tool functions that are intuitively usable with respect to the modelling method.

This all the more takes place when a tool for a multi-view modelling method has to be designed. In multi-view modelling, each view represents

certain aspects of the system. The union of the views gives the whole system. Here, it is advisable to replace the conventional diagram-oriented modelling approach by a system-oriented one (see section 3.2.1):

- In diagram-oriented modelling, editing of a model is done by applying an operator to a single diagram. The effects of the operator can only be seen in the diagram.
- In system-oriented modelling, an operator is applied to a specific diagram or to the model itself. Its effects can be seen in all corresponding diagrams.

The object of research in this paper is the design of the early steps in the development of a multi-view modelling tool for a multi-view modelling method. This ends up with the functional requirements for the tool implementation. The goal of the paper is to break down the complexity of developing system-oriented multi-view modelling tools into several, manageable steps by explicitly considering these early steps. Using a modelling scenario as the starting point, the approach leads to rich and well-defined functional requirements (non-functional requirements are not regarded here explicitly).

The paper aims at operationalizing these early steps for the design of multi-view modelling tools. The contribution of the paper is twofold: First, the approach is delineated in general terms. Unlike the artist introduced above, the modeller rather acts like an engineer, generally taking several views on the object under construction simultaneously (e.g., top view, front view, side view; see section 3.2.1). Hence, the approach presented here is explicitly directed towards multi-view modelling. It is aimed at facilitating the conceptual design of a software tool for a given method straightforward while considering the relevant restrictions (e.g., consistency, modelling principle). Second, the approach is applied to the Semantic Object Model (SOM), a comprehensive method for business systems modelling with an emphasis on multi-view modelling of business processes. SOM is chosen, because it brings challenging requirements to multi-view modelling. Modelling scenario, multi-view modelling principle, use cases and conceptual design of a software tool for SOM business process modelling are demonstrated.

The remainder of the paper is organised as follows: Chapter 2 gives an overview on multi-view modelling and consistency of views. Chapter 3 outlines the concepts of modelling scenario, multi-view modelling principle, use cases and conceptual design. The approach is applied to SOM business process modelling in chapter 4, leading to the conceptual design of a system-oriented multi-view modelling tool. Finally, chapter 5 reports on experience and gives an outlook on future research and development.

2 Related Work on Multi-View Modelling and Consistency of Views

Over the last decades, modelling of complex systems (e.g., enterprise architectures, software systems, business processes) has evolved from single-view to multi-view modelling (MVM). Single-view modelling is based on a selected modelling technique to capture a certain facet of the real world. In MVM, each view can be based on a different modelling technique, complementing each other to a comprehensive and holistic model (Frank 2002; Kruchten 1995; Lopez-Herrejon and Egyed 2010; Nuseibeh et al. 1994).

Using multiple views, each focusing on a certain aspect of a system (e.g., behaviour, structure) allows more exhaustive modelling of the system. Specialised modelling techniques (e.g., domain-specific languages) can improve the expressiveness and quality of the resulting model. Separation of concerns (using specialised views) also reduces the complexity of the single view, making it easier for a modeller to build correct models. On the other hand, multi-view modelling implies the challenge of integrating views to a comprehensive model and assuring consistency between different views. Although there are methodologies and process models, explicitly requiring

Type of Consistency	Syntactic	Semantic
Horizontal	The class names used in a sequence diagram should appear in the associated class diagram	The events produced in a sequence diagram should not produce states in the state diagrams of the objects which participate in the interaction
Vertical	The methods definition of a class should be consistent in all the abstraction levels in which these methods could be defined	When some child classes are created in a refinement from a parent class, the traces defined by the statemachine of the parent class should be supported by the lower-level statemachines of the child classes

Table 1: Examples of UML consistency problem classification (Lucas et al. 2009)

or tolerating inconsistency (Balzer 1991; Nuseibeh et al. 2001), the focus of the paper at hand is on methodologies, which look upon consistency between views as essential.

Therefore, a brief classification of consistency for multi-view models is introduced. In the context of a MVM method, inconsistency can be described as 'any situation in which two descriptions do not obey some relationship that is prescribed to hold between them' (Nuseibeh et al. 1994, 2001). Paige (Paige et al. 2007) describe multiview consistency checking (MVCC) as the requirement, that 'two or more diagrams, each presenting a different view, do not contradict each other according to a set of (metalevel) rules'. The authors distinguish between multiview consistency checking and model conformance. Model conformance deals with the question, whether the views correspond to their syntactic specification (e.g., described in a meta-model). Meta-modelling platforms can ensure model conformance in most cases, as the meta-model itself is introduced to the platform. As MVCC must be provided by the tool developer, the presented approach has an emphasis on MVCC aspects.

A popular language for MVM is the Unified Modeling Language (UML) (Object Management Group (OMG), Unified Modeling Notation). The UML provides different types of diagrams for modelling views on structure (e.g., class diagrams) and behaviour (e.g., activity diagrams) of a complex (software-intensive) system. Although UML is the de-facto industry standard for the design of object-oriented software, there is a lack of consistency-checking between diagrams (Engels et al. 2001; Gomaa and Wijesekera 2003; Huzar et al. 2005; Mens et al. 2005; Usman et al. 2008).

The paper at hand utilises the business process modelling method of the Semantic Object Model (Ferstl and Sinz 1990). SOM business process models are specified using a multi-view approach which comprises a structure view, a behaviour view as well as views on the decomposition of business transactions and business objects. These views are specified as projections onto an integrated meta-model. Similar to the relational algebra, the projection operator specifies a view definition on the meta-model by selecting the meta-model elements considered in the view.

A lot of research has been done on classifying and dealing with consistency problems in multi-view modelling (see workshop series (Huzar et al. 2005) from 2003 to 2005, (Lopez-Herrejon and Egyed 2010; Mens et al. 2005; Usman et al. 2008). In the following, the most relevant classes of consistency are briefly outlined:

- *Horizontal consistency*, also called *intra-model* consistency, refers to views representing the same abstraction level. Using different views, optionally built with respect to different modelling paradigms, is the basic concept of multi-view modelling. Horizontal consistency problems therefore occur in any multi-view model.

- *Vertical consistency*, also called *inter-model* consistency, refers to views representing different levels of abstraction. When modelling complex systems, model refinement is often used to specify and to extend the model in a stepwise and model-driven manner. Moreover, consistency between models built in different phases of software development (e.g., design, analysis), representing different abstraction levels, belongs to this category.
- *Syntactic consistency* guarantees, that a model conforms to its language definition, usually specified by the meta-model of the method.
- *Semantic consistency* requires the artefacts captured in different views to be semantically consistent to each other. The semantic information, which a modeller gains from different views, must be coherent.

The consistency classes horizontal / vertical as well as syntactic / semantic are assumed to be disjoint in pairs and can be combined mutually. Table 1 exemplifies concrete consistency problems related to multi-view modelling with UML. The consistency problems that come with the use of SOM are intensively discussed in the application scenario of the approach (see chapter 4).

Considering consistency problems in the tool development process is substantial for the usability of the tool and the correctness of the models obtained. Whereas lots of literature can be found on MVM (e.g., Nuseibeh et al. 1994, 2001), model consistency (e.g., Lopez-Herrejon and Egyed 2010; Paige et al. 2005, 2007; Usman et al. 2008), and software development in general, no literature has been found on the process of designing a modelling tool according to a given MVM method and considers the discussed problems in a formal and comprehensive way. Most literature found only deals with isolated solutions for a specific problem-domain (e.g., support for multiple user experience (Huang et al. 2011), multi-view model evolution (Stolz 2010), collaborative modelling tools (Gallardo et al. 2012)) or domain-specific language (DSL) (Grundy et al. 2007; McIntosh et al. 2008; Mens et al. 2005) built with a specific development platform. In the following, an approach tackling this shortcoming is introduced in an abstract manner before its concepts are applied to practical tool design for a concrete method.

3 From a Multi-View Modelling Method to the Conceptual Design of a Multi-View Modelling Tool

The approach to bridge the gap from a multi-view modelling method to the design of a multi-view modelling tool comprises the following steps:

1. Characterisation of the modelling scenario, specifying the situation in which a human modeller acts to create a model of an existing or postulated subarea of the real world. Thereby the modeller is guided by the modelling method and supported by the modelling tool.
2. Specification of the multi-view modelling principle, defining the general way of carrying out multi-view modelling, and use cases, identifying the usage of the modelling tool by a human modeller (actor).
3. Definition of the conceptual design of the modelling tool, specifying the tool from a functional perspective (model representation, modelling operations) as well as the user interface used for interaction with a human modeller.

The three steps are carried out with respect to a given modelling method, i.e., they determine an appropriate consideration of metaphor, meta-model, architecture model and process model in order to improve the usability and utility of the tool. For this reason, these steps establish the most important and critical phase in the process of designing a modelling tool.

In order to reduce the complexity of specifying a conceptual design for a multi-view modelling method, it is advisable to perform the steps in a sequential way. This means, that the different views grounded in the modelling scenario serve as an input for the use case definition. The use cases then are elaborated by enhancing them with

consistency issues, which result in the conceptual design of the modelling tool. This output constitutes the input for the subsequent software development.

3.1 Modelling Scenario

The characterisation of a modelling scenario as the first step of the approach is depicted in Fig. 1. The modelling scenario shows the modeller (or model user) creating a model by means of a multi-view modelling tool. This explication of the modeller's perspective helps to manage the complexity of the modelling task.

The result of the modelling process is an artefact which is perceived to be a model of a certain subarea of the real world (object) according to a mapping function. In contrast to a mathematical mapping, the object is not specified formally. Instead, the model is established by a human modeller (subject) with respect to his/her understanding of the real world and the delimitation of the object within the real world (Wolf 2001). For this reason, the resulting artefact always depends on the modeller's understanding of the real world and the subarea to be modelled. Hence, the artefact is a subjective reconstruction of a subarea of the real world. Different modellers will in general build different artefacts. A modelling tool which is carefully designed to support the modelling method will help to margin the subjective differences and foster an intersubjective understanding of the model.

The role of the modeller within a modelling scenario is as follows: The modeller pursues certain goals which reflect the purpose of the model to be built. The goals refer both to the purpose of the model and the stakeholders to use the model. The modeller is guided by a metaphor, giving the general idea of understanding the modelling language and reconstructing the relevant part of the real world, e.g., the metaphor of a data modelling approach is to model the static data structure of a business system, upon which all functions are defined. With this knowledge in mind, the modeller perceives the real world and delimits the subarea to be modelled. The modeller then reconstructs the object in terms of the modelling language (which in turn is inspired by the metaphor), thereby building the model.

Because of the complexity of models, the modeller may be not able to comprehend the artefact as a whole. Instead, the modeller is provided with several different views on the artefact, altogether representing the model. This approach is covered by the concept of multi-view modelling.

3.2 Multi-View Modelling Principles and Use Cases

After clarifying the general way of carrying out multi-view modelling (see section 3.2.1) use cases depict the interaction of a human modeller and a modelling tool (see section 3.2.2). Use cases represent a delimited functionality of a system which is used by a human or machine actor. The use cases of a system can be described by a use case diagram (e.g., UML use case diagram). Unfortunately, UML use case diagrams do not relate use cases to certain views of a method. The notation of the use cases together with the relevant views in the presented approach is in tabular manner. Non-functional requirements (e.g., performance, stability) are not considered, because they are normally not restricted to a certain set of views. Nevertheless, non-functional requirements should be specified as well in order to provide the tool developer with a complete requirements specification for a modelling tool.

3.2.1 Multi-View Modelling Principles

There are, in general, two different ways of carrying out multi-view modelling: (1) Develop the views separately and then combine them to an integrated and consistent model, or (2) build the integrated model as a whole and derive the views as projections on the integrated model. Type (1) is often performed using drag & drop modelling which is well-known from many graphical editors. Type (2) follows the model-view-controller

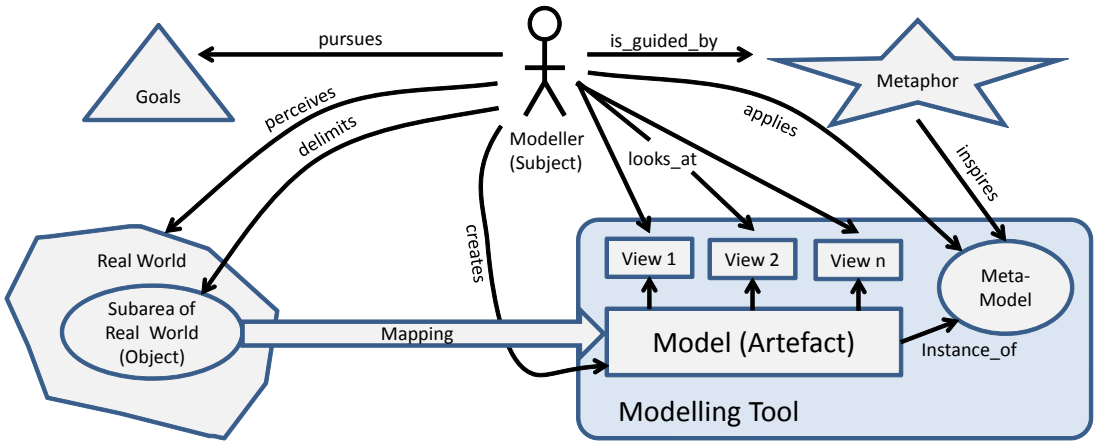

Figure 1: Modelling Scenario

pattern (Reenskaug 1979) which considers several views on a model. The model can be changed directly or via a view. Any change affects the model itself and becomes visible in all respective views. Modelling principle (1) is referred to as diagram-oriented, (2) as system-oriented modelling. Table 2 characterises the two multi-view modelling principles.

The principles of multi-view modelling are illustrated in Fig. 2 using an example from the engineering domain. An engineer is assumed to model a spatial (3D) object, e.g., a simple cube with a drill hole. The 3D object will usually be represented by three 2D views (top view, front view, and side view), each of them as a projection on the integrated 3D model. According to diagram-oriented multi-view modelling, one view is modelled after another and then consistency is verified (see Tab. 2, Model editing). In the example in Fig. 2, each view is consistent on its own, top view and front view as well as top view and side view are consistent in pairs. However, front view and side view are contradictory. By contrast, system-oriented multi-view modelling causes the consequences of an editing operation in one view become immediately visible in all respective views (Tab. 2, Change propagation). Specifying the depth of the drill hole in the front view would cause a change of both the integrated model and the side view. The inconsistency shown in Fig. 2 cannot occur.

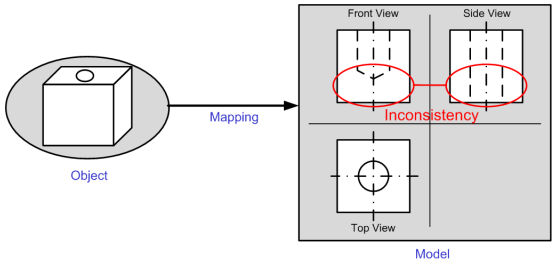

Figure 2: Multi-view modelling of a geometrical object

3.2.2 Use Cases

The next step is to define the use cases for the creation and editing of multi-view models with an appropriate tool. The key concepts of conventional use case diagrams are actors, use cases and relationships between actors and use cases (Object Management Group (OMG), Unified Modeling Notation Superstructure). Include, extend and generalisation relationships can be used to define the relationship between use cases.

The conventional use case diagrams are enhanced with specific information considering the views affected by a use case. The views can be regarded as an interface between an actor and corresponding use cases and follow the model-view-controller paradigm (MVC) (Reenskaug 1979). This means, that a use case can be triggered directly or via

	Diagram-oriented	System-oriented
Model editing	One diagram (view) at a time	Model as a whole, visible via different views
Model refinement	Each diagram represents a view on the model with respect to a certain level of refinement	Refinement of model elements is part of the model
Refinement levels	Fixed	Selective zoom-in and zoom-out
References between views	References between selected model elements of different diagrams	Views as projections on the model
Model consistency	Inconsistencies must be resolved by the modeller	Inconsistencies are prevented by the integrated model
Change propagation	Limited or no propagation of changes	Changes in one view are propagated to all affected views immediately

Table 2: Classification of multi-view modelling principles

a certain view. The results can be seen in each relevant view. Putting the MVC paradigm to the tool development, conventional use cases are enhanced with the information (1) in which view or set of views a use case can be triggered and (2) in which view or set of views the execution of an use case has an effect on. The views itself have been specified in the modelling scenario. In the following, the notion of the enhanced use cases is introduced (for an example see Tab. 3):

Use Case A unique identifier or a number followed by a meaningful and short description for the use case, specifying a set of actions performed by the system.

Triggered in view A set of views (zero to many), the use case can be triggered in. Zero, when a use case can be triggered without being restricted to a certain view (e.g., directly on the model using the context menu of the modelling tool). Many, when a use case may be triggered in several views.

Effect on view For each view of the method, a separate column is listed, depicting whether the execution of a use case has always (value 'Yes'), conditionally (value 'Cond') or never (value 'No') an effect on the specific view. The specification of the effect on a view is part of the conceptual design in the next step. A distinction between (1) a direct effect based on the integrated model and (2) an indirect effect because of the usability of the tool (e.g., visualising some hints for the modeller or giving some guidance during the modelling process) is carried out. (1) is displayed in the 'Effect' column of the use case definition and must be considered later on to ensure model conformance, whereas (2) is not considered in the use case. Effects not considered with the use cases can serve as an input for the definition of the non-functional requirements.

References Use Case A list of references to use cases by means of the include, extend or generalisation relationship. The references are defined by the relationship-type and the identifier or number of the related use case.

3.3 Conceptual Design

The third step of the approach proposed here is the conceptual design of a multi-view modelling tool to support a method. This is done in the following by specifying the tool functions, which are derived from the use cases referring to the modelling scenario. The tool must provide

a set of functions, enabling a human modeller to create a comprehensive model as well as to capture the modelling steps and results via the views on the model. Basic functions (e.g., create model elements, connect model elements) should be supported in an intuitive way. Modelling transactions, representing a sequence of editing operations which transform a consistent state of the model into a new state, which again is consistent with respect to syntax and semantics (Bork and Sinz 2010), have to be defined (e.g., refinement of models, model transformation).

The conceptual design gives a specification of tool functions, particularly considering MVM and view consistency. This provides the input for the subsequent implementation of the tool. The specification should include at least:

Function Denomination of the function.

Object Set of elements of the languages' meta-model on which an operation can be performed (e.g., model element, view), elements of the modelling tool itself (e.g., context menu, window) or the model as a whole.

Operator Signature of the operator.

Effect Consequences of executing the operation on both the view from which the operation is triggered as well as the integrated model and affected views.

Consistency Consistency classes (see chapter 2) concerned by the execution of the function. Abstract and informal description of the consistency problems occuring because of the use case execution. Especially the use cases with conditional effects on a view must be considered extensively. The conditional aspects (i.e., under which condition do which consistency issues come up) have to be regarded in the consistency column.

In contrast to the use cases (see section 3.2.2), an emphasis of the conceptual design is the consideration of the consistency issues coming with a multi-view model. The conceptual design therefore adds the classes of consistency (see chapter 2) concerned by the execution of a function together with some brief information about how the tool developer should ensure model consistency. Detailed consistency preserving mechanisms are not part of the conceptual design, because their implementation depends on the development platform or programming language of the tool. A focus of the conceptual design should be on the conditional effects of the use cases, showing the way the tool should consider them appropriately.

In addition to the definition of the multiple views, the internal representation of the model must be defined. Depending on the modelling principle (see section 3.2.1) of the method, integration of views has to be supported. If an integrated model is used (alternative 2), the projections specifying the views have to be defined, preferably by a mapping between the meta-models (meta-model of the integrated model and meta-model of the view). Algorithms which detect changes on a view and trigger corresponding modifications on both the internal representation of the model and all affected views have to be installed.

Whenever a method allows the definition of models on different abstraction levels, navigation between those levels should also be supported. Vertical consistency (see chapter 2) is concerned about the consistency of models on different abstraction levels. An intuitive and preferably model-driven specification of those abstraction levels has to be supported. Mutual navigation between the different levels seems to be a powerful weapon to tackle model complexity, especially in MVM. Furthermore, a clear separation of functions for navigation (1) and functions for model editing (2) is advisable. (1) has no effect on the integrated model representation, whereas (2) leads to an update on the integrated model and to updates on affected views.

4 From the SOM Method to a SOM Business Process Modelling Tool

In the following, the approach outlined in chapter 3 is applied to SOM business process modelling.

Section 4.1 gives a brief overview on the SOM method with an emphasis on SOM business process modelling. Section 4.2 adopts the steps of modelling scenario, multi-view modelling principle, use cases and conceptual design to the SOM business process modelling method, thus bridging the gap to the specification of a SOM business process modelling tool.

4.1 SOM Method for Business Process Modelling

The Semantic Object Model (SOM) is a comprehensive method for enterprise modelling with an emphasis on business process modelling and application systems specification. The first ideas of SOM have been published in 1990 (Ferstl and Sinz 1990). Since then the method has emerged steadily.

In the following, the focus is on the business process modelling part of the SOM method (Ferstl and Sinz 2005). According to chapter 1, the constituents of the method are:

Metaphor SOM business process modelling follows the metaphor of a distributed system, consisting of autonomous and loosely coupled business objects. Business objects are coordinated by means of business transactions towards the fulfilment of common goals. Thus, the metaphor combines the basic concept of object-orientation with transaction-based coordination.

Meta-model The centre of the meta-model for SOM business process models is built by the concepts of business object and business transaction. A business transaction coordinates two business objects. A business object can be involved in several business transactions. A business object comprises one or more business tasks, which are assumed to operate on the same object. A business transaction is performed by two tasks belonging to different business objects. Tasks belonging to the same object can be connected by an internal event. Moreover, the execution of a task can be triggered by an external event. Each transaction is involved in the coordination of at least one good or service.

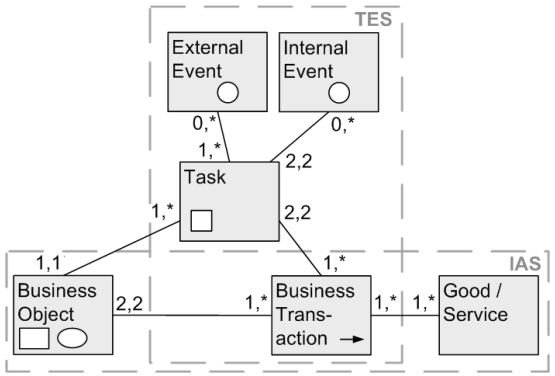

Figure 3: Business process meta-model of the SOM method (Ferstl and Sinz 2013)

A business process model according to the meta-model in Fig. 3 is represented using a structure view called interaction schema (IAS) and a behaviour view called task-event schema (TES). Both views can be derived by projections onto the meta-model (see Fig. 3).

Business objects and business transactions can be decomposed recursively. According to the principle of non-hierarchical coordination, a transaction can be decomposed into subsequent initiating, contracting and enforcing transactions (rule 1). Corresponding to the principle of hierarchical coordination, an object can be decomposed into sub-objects, connected by a control transaction and a report transaction (rule 2). The resulting business objects and business transactions can be decomposed recursively (rule 3 and 4) (see Ferstl and Sinz (2013) for a complete list of the decomposition rules specified in Backus-Naur-Form (BNF)).

(1) $T(O,O') ::= [[Ti(O,O')seq]$
$Tc(O',O)seq]\ Te(O,O')$

(2) $O ::= \{O', O'', Tr(O',O''),\ [Tf(O'',O')]\}$

(3) $O' \mid O'' ::= O$

(4) $Ti \mid Tc \mid Te\ ::= T$

Architecture Model The overall enterprise architecture of the SOM method comprises model

layers for (1) enterprise plan, (2) business process model and (3) resources. A business process model specifies a procedure to carry out a given enterprise plan. Resources (especially human actors and application systems) enable the execution of business processes. Inside a business process model, objects and transactions can be decomposed according to the rules explained above.

Process Model From a logical point of view, a SOM enterprise architecture is assumed to be developed top down, i.e., starting with an enterprise plan, then specifying the corresponding business process model and finally providing the resource models.

4.2 SOM Modelling Scenario

With respect to the modelling scenario introduced in section 3.1, the subarea of the real world (object) is a single business process (e.g., the procurement process of a trading company) or a system of interacting business processes. The resulting artefact is a business process model according to the SOM business process meta-model. The modeller pursues certain goals which refer to the purpose of the model (e.g., building a model to support a redesign of the procurement process to cut cost and throughput time) and the stakeholders using the model (e.g., domain experts, managers). Perception of the real world and delimitation of the object as well as creation of the artefact are guided by the SOM business process modelling metaphor. Figure 4 illustrates the modelling scenario of SOM business process modelling.

SOM multi-view business process modelling is based on four views: interaction schema (IAS) and task-event schema (TES) have been already introduced in section 4.1 as structure view and behaviour view respectively. As decomposition of business objects and business transactions is an integral part of the model, decomposition trees serve as a third and fourth view of SOM business processes models.

4.3 SOM Multi-View Modelling Principle and Use Cases

SOM business process modelling generally utilises the system-oriented principle (see section 3.2.1) for multi-view modelling. This means in turn, that the tool is directed towards consistency preserving instead of consistency checking and recovery. All views are seen as projections on one comprehensive and integrated model. Any modelling operation executed by the modeller is applied to the internal model representation and then all changes are triggered to the affected views. As mentioned in section 3.3, the conceptual design includes the information about the consistency issues of a modelling operation but not the algorithms for ensuring a consistent model. Model refinement must be supported by realising the decomposition rules, given by the method (Ferstl and Sinz 2005). The decomposition itself is a central part of SOM business process modelling. Therefore, selective zooming in and out of the business process model must be realised as modelling operators on the decomposition trees of business transactions and business objects. Applying the zoom operator results in the visualisation of a more (zoom-in) or less (zoom-out) refined business process model without performing any changes on the integrated model. The modeller therefore has the ability, to switch between already specified refinement levels.

Picking up the information gathered in the SOM modelling scenario in the previous step, typical use cases of a modeller can be defined in the tabular manner introduced in section 3.2.2. For each use case and each of the methods view must be decided, whether the execution of a use case has an effect on a view or not. As mentioned before, SOM utilises a system-oriented approach, therefore the integrated model must also be considered during the use case definition. Table 3 illustrates a selection of modelling operations needed to build and refine SOM business process models and their effect on the different views and the integrated model.

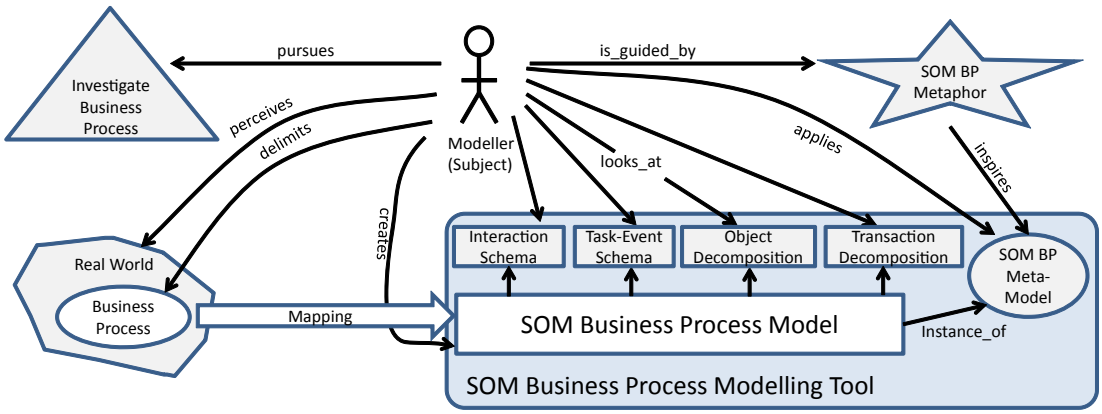

Figure 4: SOM business process modelling scenario

Table 3 shows, that e.g., the decomposition of a business transaction can be triggered within three views (IAS, TES and within the transaction decomposition view) and has an effect on the integrated model and the transaction decomposition view. The use case of increasing the visualised business process level (zooming) can be triggered within transaction decomposition and object decomposition view. Its execution affects only IAS and TES, not the integrated model. Whenever several views are affected, the tool developer has to ensure that all changes are propagated and the considered views are updated accordingly. Simple changes (e.g., the change of an elements name) may be transported through changes on a central repository. If the development platform does not support a repository or the changes are more complex, additional consistency logic hast to be implemented on the platform, e.g., using a programming or scripting language. In the latter case, the conceptual design gives some guidance for the developer by briefly describing the consistency issues of executing a modelling operation.

The table also shows, that the use case of adding a business object (use case 6) is related to the use case 7 (add business transaction) and 11 (Smooth Edges of Transactions) by an include relationship. This is caused by the fact, that any additional business object has to be connected with one existing business object by an additional enforcing transaction. After the new business object and business transaction are introduced to the IAS and TES, the adjustment of the transaction must be performed in order to provide a good visualisation of the business process model for the modeller. The information summarised in Tab. 3 serves as a starting point for the conceptual design of the functional requirements for a multi-view SOM modelling tool in the next step.

4.4 SOM Conceptual Design

In the following, the conceptual design (see section 3.3) of a SOM business process modelling tool is outlined. This section will concentrate on tool functions, modelling transactions and consistency between views. As mentioned before, SOM business process modelling is characterised as a comprehensive system-oriented modelling method. Utilising the method requires complex modelling transactions for model creation, model editing, and model refinement. A modelling transaction is the refinement of the business process model. First, the modeller selects a transaction and triggers the 'Decompose Business Transaction' function in its context menu. After that, the tool provides the modeller with a list of all applicable SOM decomposition rules. The modeller chooses one rule and applies it to the selected transaction. The decomposition trees are updated and enhanced with the new child nodes. The IAS and TES are still on the previous decomposition level. This is an intermediate inconsistency the

Use Case	Triggered in View	Effect on					References
		Interaction Schema	Task-Event Schema	Transaction Decomp.	Object Decom-position	Integrated Model	Use Case
1.Decompose Business Transaction	Interaction Schema Task-Event Schema Transaction Decomp.	No	No	Yes	No	Yes	No
2.Decompose Business Object	Interaction Schema Task-Event Schema Object Decomp.	No	No	No	Yes	Yes	Include(7)
3.Revoke Transaction decomposition	Transaction Decomp.	Cond	Cond	Yes	No	Yes	No
4.Revoke Object decomposition	Object Decomp.	Cond	Cond	Cond	Yes	Yes	Include(9)
5. Zooming	Transaction Decomp. Object Decomp.	Yes	Yes	No	No	No	Include(11)
6.Add Business Object	No specific view	Yes	Yes	No	Yes	Yes	Include(7,11)
7.Add Business Transaction	No specific view	Yes	Yes	Yes	No	Yes	Include(11)
8.Remove Business Object	No specific view	Yes	Yes	No	Yes	Yes	Include(9,11)
9.Remove Business Transaction	No specific view	Yes	Yes	Yes	No	Yes	Include(11)
10.Increase Business Process Level	Interaction Schema Task-Event Sschema	Yes	Yes	No	No	Yes	Include(11)
11.Smooth Edges of Transactions	Interaction Schema	Yes	No	No	No	No	No

Table 3: SOM business process modelling use cases

tool allows in order to give the modeller more flexibility. Second, the modeller has the possibility to manually trigger the function 'Increase Business Process Level' in order to connect the newly created transactions with the business objects, therefore producing a new consistent state of the business process model and finalising the modelling transaction.

Table 4 specifies SOM modelling functions, using the criteria introduced in section 3.3 and based on the information already gathered during the use case definition in the previous step. The conceptual design includes a complete specification of any use case depicted in the earlier step with an emphasis on consistency of the model. The modelling functions are restricted to a certain object of the method or the tool itself and enlarged with a brief description of their effect and consistency issues that have to be considered during tool development.

Although all the features mentioned above are important for the quality of the tool, the internal representation of the model and the consistency between the views deserve some closer attention. Initialisation of the integrated model and verification of consistency between views are more or less complex, depending on the properties of the selected development platform. Some platforms support the utilisation of the model-view-controller pattern (Reenskaug 1979) by a repository approach, which is very helpful for system-oriented methods (see section 3.2.1) with an integrated meta-model. In this case, the integrated model is realised by a repository, all view elements are implemented as references to repository items. Changes on a view (e.g., changing the name of an element) are automatically pushed into the repository and therefore become immediately visible in all affected views.

Besides the discussed conceptual design, the tool developer should have a specification of non-functional requirements as well. As those requirements are not handled significantly different in multi-view modelling, they should be specified additionally to the conceptual design.

5 Lessons Learned and Outlook to Future Research

The approach presented here contributes to bridge the gap from a multi-view modelling method to the conceptual design of a multi-view modelling tool. The constituents of a modelling method are a metaphor, a meta-model, an architecture model, and a process model. The conceptual design of a corresponding modelling tool consists of specifications of the tool functions with an emphasis on multi-view consistency.

What makes this a challenge is the multi-view aspect, i.e., when the starting point is a multi-view modelling method and the end point is a system-oriented multi-view modelling tool. Here, the handling of the multiple views by the modeller has to be adjusted with his/her way of reconstructing the subarea of the real world to be modelled in several views. The cornerstones of the approach are

1. a modelling scenario (the overall perspective—a specification of the modelling method, applied to a real world problem by the modeller),
2. a modelling scenario and use cases (the modeller's perspective—the use cases performed by the modeller and their effect), and
3. tool functions (the tool perspective—modelling functions provided by the tool).

These cornerstones should help to break the semantic gap down to manageable steps and thus gain a better tool support.

The motivation for the presented approach here is the experience gathered with the development of several SOM tools over the last decades. Indeed, the tools were oriented at the modelling method, but they were disregarding the modelling scenario and the use cases. Experience with the new tool, designed according to the approach presented here, includes:

- The documentation of a modelling scenario helps the users to learn about the usage of the tool. Misunderstandings on the usage of the tool can be avoided.

Function	Object	Operator	Effect	Consistency
Decomposition	Business Object	Decompose	The selected object is decomposed according to a selected decomposition rule. Object decomposition view and the integrated model must be updated.	Horizontal & Vertical. If the negotiation principle for object decomposition is applied, the modeller might create additional transactions which must be visible in the transaction decomposition tree.
Decomposition	Business Object	Revoke Decomposition	The selected object and its sibling nodes in the object decomposition tree are deleted. Object decomposition view and the integrated model must be updated.	Horizontal&Vertical If the modeller created additional transactions during the prior decomposition of the selected objects, these transactions have to be deleted. If IAS and TES had showed the selected elements, they must be updated.
Zooming	Business Object/ Business Transaction	Zoom on selected level	IAS and TES are updated to a more or less detailed decomposition level. No changes on the integrated model and the decomposition trees.	Horizontal Attribute values of the created elements must be consistent in all views. The connection of the transactions and objects must be consistent in IAS and TES.
Add Model Element	Business Process Model	Add Business Object	An additional environmental object is created and added to the model. It must be connected with a new enforcing transaction to an existing object. The decomposition trees, IAS, TES and the integrated model must be updated.	Horizontal & Vertical The new elements must be shown in all four views with the correct attribute values. It must be connected to the newly created transaction in IAS and TES.
Remove Model Element	Business Object/ Business Process Model	Remove Business Object	The selected business object is removed together with its child elements from the object decomposition tree, IAS and TES. The integrated model must be updated.	Horizontal & Vertical If transactions hang loose after the object deletion, the modeller must connect them again, using the parent nodes of the deleted objects in the object decomposition tree.

Table 4: Specification of SOM business process modelling functions

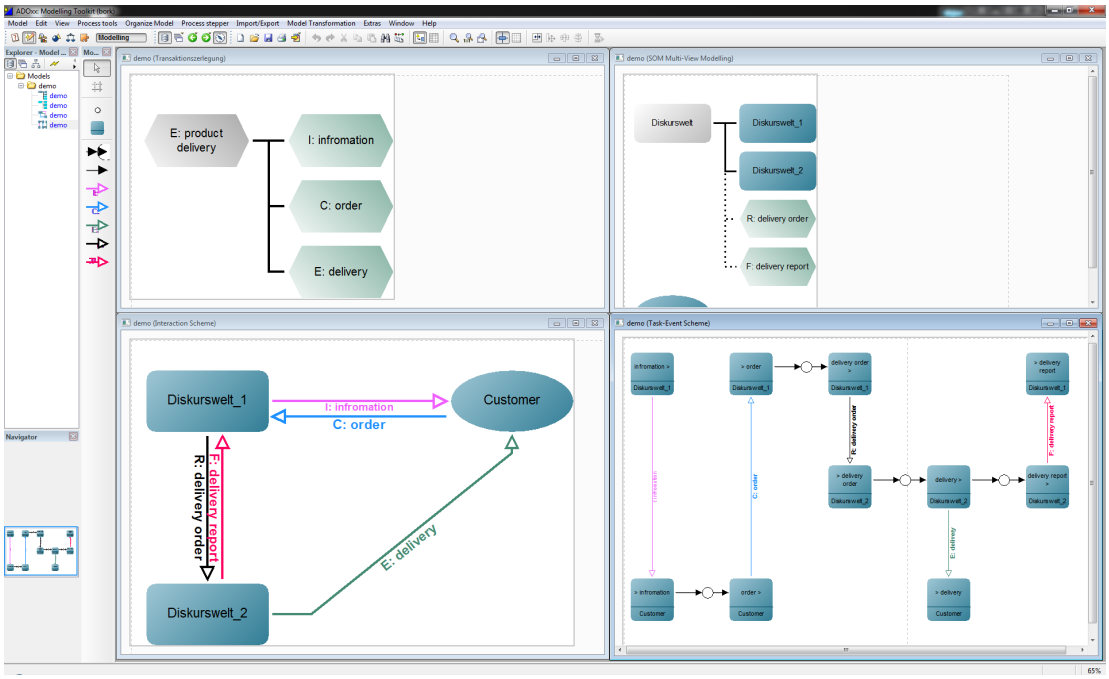

Figure 5: Screenshot of the SOM multi-view business process modelling tool

- The early phases of the design process of the tool are broken down into comprehensible steps. This leads to tool functions balanced with the modelling scenario.
- The consideration of multi-view aspects in the early phases sharpens the requirements and therefore results in a more comprehensive functional specification.
- All in all, the approach appears to be a contribution to requirements engineering of modelling tools.

The approach was applied to SOM business process modelling. SOM brings manifold requirements to multi-view modelling (see Tab. 3 and Tab. 4), e.g., the decomposition of model elements is part of the model itself. An approach working for SOM should work for modelling methods with equal or fewer requirements as well. This is a hypothesis to further research.

The tool is based on the ADOxx meta-modelling platform. Bork and Sinz (2010) report on the design and the implementation of the SOM business process modelling tool on the ADOxx meta-modelling platform in more detail, focussing on the subsequent phases of tool development. A prototype of the tool is available within the Open Model Initiative[1].

Future research will concentrate on generalising the presented approach. It is assumed to be a universal idea, helping to better design modelling tools and thus supporting the proliferation of modelling methods. Therefore, we plan on applying the approach to a completely different multi-view modelling method. Additionally, we think about realising the approach on a meta-modelling platform, therefore guiding method experts in the process of specifying the functional requirements for a modelling tool.

Additional research has to capture some facets of a modelling method which are disregarded up to now, particularly architecture model and process model. All in all, the presented approach should be further investigated as a contribution to requirements engineering for multi-view modelling

[1]The Open Model Initiative, `http://www.openmodels.at`, last visit: 12-02-2013

tools on the one hand and knowledge engineering for multi-view modelling methods on the other.

References

Balzer R. (1991) Tolerating Inconsistency. In: Belady L., Barstow D. R., Torii K. (eds.) Proceedings of the 13th international Conference on Software Engineering. ICSE '91. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 158–165

Bork D., Sinz E. J. (2010) Design of a SOM Business Process Modelling Tool based on the ADOxx meta-modelling Platform. In: de Lara J., Varro D. (eds.) Pre-Proceedings of the 4th international Workshop on Graph-Based Tools. GraBaTs 2010. Enschede, The Netherlands, pp. 90–101

Engels G., Küster J. M., Heckel R., Groenewegen L. (2001) A methodology for specifying and analyzing consistency of object-oriented behavioral models. In: Gruhn V. (ed.) Proceedings of the 8th European Software Engineering Conference. ESEC. ACM, Vienna, Austria, pp. 186–195

Ferstl O. K., Sinz E. J. (1990) Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik 32(6), pp. 566–581

Ferstl O. K., Sinz E. J. (2005) Modeling of Business Systems Using SOM. In: Bernus P., Mertins K., Schmidt G. (eds.) Handbook on Architectures of Information Systems. International Handbooks on Information Systems. Springer, Berlin, pp. 347–367

Ferstl O. K., Sinz E. J. (2013) Grundlagen der Wirtschaftsinformatik, 7th ed. Oldenbourg, München

Frank U. (2002) Multi-perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences. HICSS, p. 72

Gallardo J., Bravo C., Redondo M. A. (2012) A model-driven development method for collaborative modeling tools. In: Journal of Network and Computer Applications 35(3), pp. 1086–1105

Gomaa H., Wijesekera D. (2003) Consistency in Multiple-View UML Models: A Case Study. In: Kuzniarz L., Huzar Z., Reggio G., Sourrouille J. L., Staron M. (eds.) Workshop on Consistency Problems in UML-based Software Development, International Conference on the Unified Modeling Language-the Language and its applications. UML, pp. 1–8

Grundy J., Hosking J., Cao S., Zhao D., Zhu N., Tempero E., Stoeckle H. (2007) Experiences developing architectures for realizing thin-client diagram editing tools. In: Software: Practice and Experience 37(12), pp. 1245–1283

Huang K.-H., Nunes N., Nobrega L., Constantine L., Chen M. (2011) Hammering Models: Designing Usable Modeling Tools. In: Campos P., Graham N., Jorge J., Nunes N., Palanque P., Winckler M. (eds.) Human-Computer Interaction INTERACT 2011. Lecture Notes in Computer Science Vol. 6948. Springer Berlin Heidelberg, pp. 537–554

Huzar Z., Kuzniarz L., Reggio G., Sourrouille J. (2005) Consistency Problems in UML-Based Software Development. In: Jardim Nunes N., Selic B., Rodrigues da Silva A., Toval Alvarez A. (eds.) UML Modeling Languages and Applications. Lecture Notes in Computer Science Vol. 3297. Springer Berlin / Heidelberg, pp. 1–12

Kruchten P. (1995) The 4+1 View Model of Architecture. In: IEEE Software 12, pp. 42–50

Lopez-Herrejon R. E., Egyed A. (2010) Detecting inconsistencies in multi-view models with variability. In: Kühne T., Selic B., Gervais M.-P., Terrier F. (eds.) Proceedings of the 6th European conference on Modelling Foundations and Applications. ECMFA'10. Springer-Verlag, Paris, France, pp. 217–232

Lucas F. J., Molina F., Toval A. (Dec. 2009) A systematic review of UML model consistency management. In: Information and Software

Technology 51(12), pp. 1631–1645

McIntosh B., Giupponi C., Voinov A., Smith C., Matthews K., Monticino M., Kolkman M., Crossman N., van Ittersum M., Haase D., Haase A., Mysiak J., Groot J., Sieber S., Verweij P., Quinn N., Waeger P., Gaber N., Hepting D., Scholten H., Sulis A., van Delden H., Gaddis E., Assaf H. (2008) Bridging the Gaps Between Design and Use: Developing Tools to Support Environmental Management and Policy. In: Jakeman A., Voinov A., Rizzoli A., Chen S. (eds.) Environmental Modelling, Software and Decision Support. Developments in Integrated Environmental Assessment Vol. 3. Elsevier, pp. 33 –48

Mens T., Van Der Straeten R., Simmonds J. (2005) A Framework for Managing Consistency of Evolving UML Models. In: Yang H. (ed.) Software Evolution with UML and XML. Idea Group Publishing, pp. 1–31

Nuseibeh B., Kramer J., Finkelstein A. (1994) A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. In: IEEE Transactions on Software Engineering 20(10), pp. 760–773

Nuseibeh B., Easterbrook S., Russo A. (2001) Making inconsistency respectable in software development. In: Journal of Systems and Software 58(2), pp. 171–180

OMG Object Management Group (OMG), Unified Modeling Notation. http://www.uml.org/. Last Access: (2012-06-14)

OMG Object Management Group (OMG), Unified Modeling Notation Superstructure. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. Last Access: (2012-06-14)

Paige R. F., Kolovos D. S., Polack F. A. C. (2005) Refinement via Consistency Checking in MDA. In: Electronic Notes in Theoretical Computer Science 137(2), pp. 151–161

Paige R. F., Brooke P. J., Ostroff J. S. (July 2007) Metamodel-based model conformance and multiview consistency checking. In: ACM Trans. Softw. Eng. Methodol. 16(3)

Reenskaug T. (1979) Thing-Model-View-Editor – an Example from a planningsystem. http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf. Last Access: (2012-06-14)

Stolz V. (2010) An Integrated Multi-View Model Evolution Framework. In: Innovations in Systems and Software Engineering 6 (1-2), pp. 13–20

Usman M., Nadeem A., Kim T.-h., Cho E.-s. (2008) A Survey of Consistency Checking Techniques for UML Models. In: Advanced Software Engineering and Its Applications, pp. 57–62

Wolf S. (2001) Wissenschaftstheoretische und fachmethodische Grundlagen der Konstruktion von generischen Referenzmodellen betrieblicher Systeme. PhD thesis, University of Bamberg, Shaker Verlag

Domenik Bork, Elmar J. Sinz

University of Bamberg
Department of Information Systems – Systems Engineering
An der Weberei 5
96047 Bamberg
Germany
{domenik.bork | elmar.sinz}@uni-bamberg.de