A MODEL-BASED NARRATIVE USE CASE SIMULATION ENVIRONMENT

Veit Hoffmann and Horst Lichter
Research Group Software Construction, RWTH Aachen University, Ahornstr. 55, Aachen, Germany

Keywords: Requirements, Use Cases, Scenarios, Simulation, Quality Assurance, Model-based Development.

Abstract: Since their introduction use cases are one of the most widespread used techniques to specify functional requirements. Because low quality use cases often cause serious problems in later phases of the development process the simulation of use cases may be an important technique to assure the quality of use case descriptions. In this paper we present a model based use case simulation environment for narrative use cases. At first we motivate core requirements of a simulation environment and an underlying execution model. Moreover we describe our model based simulation approach and present some first experiences.

1 INTRODUCTION

Defects in requirements specifications often lead to severe problems. They cause projects to exceed the planned budget or even lead to a complete failure (The Standish Group, 2003). As a consequence quality assurance of the requirements is an essential and crucial task.

Empirical studies (Weidmann, Hoffmann & Lichter 2009) show that since their introduction by Ivar Jacobson in 1986 (Jacobson 1987 & 2004) use cases are one of the most important requirements techniques to specify the functional behavior of a system. Many publications (Cockburn 2000, Armour & Miller 2001, Kulak & Guiney 2003, Bittner & Spence 2003) address use case modeling which is nowadays an integral part of modern development processes like the Unified Process (Jacobson, Booch & Rumbaugh 1999).

Despite of its frequent usage in practice, use case modeling has two principal weaknesses:

1. Each use case describes multiple scenarios at once. Stakeholders, especially domain experts, report problems to evaluate the correctness of use case descriptions because the scenarios described by the use cases are often scattered over one basic and several alternative behavioral fragments.
2. Since each use case only specifies one specific system function or even a part of it, it is difficult to evaluate the global system behavior resulting from the interactions of the use cases. Hence, traditional inspection techniques are not well suited to identify inconsistencies between use cases or missing behavior (Drenger & Peach 2004).

Simulation is a proven technique and an intuitive means to assess the behavior of a system. Therefore simulation of use cases may help to overcome those problems. By means of simulation stakeholders can experience the behavior by entering simple stimuli and thus evaluate the modeled functionality by stepping through the specified scenarios. Moreover, simulation may be used proactively during the specification process to support step-wise refinement. The results of different use case simulation runs may be analyzed to deduce information about the structure of the described behavior.

In this paper we present

- the requirements of an integrated use case simulation environment supporting use case quality assurance,
- and a simulation approach for so called narrative use cases based on petri-net semantics.

The remainder of this paper is organized as follows: In sections 2 and 3 we introduce central terms used in this paper and give an overview to related research. Section 4 contains a brief introduction to narrative use cases. In section 5 we present the core requirements for a use case simulation environment. We describe an execution model for the simulation of narrative use cases and a model based transforma-

Hoffmann V. and Lichter H. (2010).
A MODEL-BASED NARRATIVE USE CASE SIMULATION ENVIRONMENT.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 63-72
DOI: 10.5220/0003006900630072

tion approach in section 6. Section 7 presents an example of our simulation approach and section 8 introduces the developed simulation environment. Finally we evaluate our approach in section 9 and give some remarks as well an outlook to future research in section 10.

2 DEFINITIONS

According to Jacobson a **use case** is a set of sequences of actions (including variants) that a system performs to yield an observable result of value to an actor. A use case defines of set of coherent scenarios which resemble the same goal. A **scenario** is a single behavioral path determined by set of consecutive events that lead to a defined result. A **scenario instance** is a concrete instantiation of a scenario containing all decisions and runtime information needed to specify a run through a scenario from its start to its end.

Simulation of behavior is typically based on formal **execution models**. An execution model is represented by a graph defining the simulation semantics. In this paper we distinguish two concepts of execution models. The **topology** of an execution model is its graph structure. It consists of the nodes of the graph and their connections. The **dynamics** addresses the runtime information of an execution model needed to control an execution sequence on a given topology.

3 RELATED WORK

Many publications (Sutcliffe et.al. 1998, Lee, Cha & Kwon 1998, William et. al. 2005, Glinz, Seybold & Meier 2007) have discussed the importance of simulation as a means to support modeling and analyzing behavioral requirements descriptions. Furthermore several approaches defining execution semantics for use cases have been published. These approaches can be distinguished into two classes:

(1) Approaches using a formal executable specification language (Whittle & Jayaraman 2006, Jorgenson & Bossen 2004).
(2) Approaches relying on a "structured" textual notation which is transformed "semi"-automatically to a notation with defined execution semantics (Somé 2006, Zhao & Duan 2009).

We consider the latter more suitable and promising because in industry text-based use case notations are highly accepted (Weidmann, Hoffmann & Lichter 2009).

Our approach is strongly related to the work of Somé (2006) who has proposed a state-machine based formalization of structured textual use cases and a tool support for the simulation of those state machines. Nevertheless our approach has two main differences compared to Somé's work. First our approach does not suffer from the problem that the simulation allows unspecified scenarios and second the topology of our execution model is more robust against changes. Because changes have to be made very often during the specification process, this is an important advantage. Kwon et. al. (1998) describe a simulation approach based on a special extension to petri-nets. Their work relates on a manual transformation and focuses on the analysis of parallel behavior, whereas our approach uses an automated transformation algorithm.

4 NARRATIVE USE CASES

A **narrative use case** is a structured textual description of a use case. We have proposed a narrative use case meta model (Hoffmann, Lichter & Nyssen 2009) that allows to describe use case behavior in a flow oriented textual manner. In a **narrative use case model** each single use case step is described in unconstrained natural language.

To model the control flow inside a use case a couple of control flow elements are used. Figure 1 shows two narrative use cases (taken from the well known ATM example).

In a narrative use case model every use case is represented by a **narrative description**. Each narrative description consists of a set of **flows** describing the use case's behavior by means of a set of actions and anchors.

Actions are behavioral atoms written in natural language and each action represents a single step performed by an actor or the system itself.

Anchors mark spots within a flow where behavior from another flow may be inserted. Different kinds of anchors exist to represent the different kinds of relationships between flows - i.e. inclusion and extension of flows.

Finally, flows are equipped with **contexts** specifying situations and conditions in which the behavior described in a flow is applied. Narrative use case models support inclusion and extension between flows as well as exceptional relationships. Each of those relationships is represented by an individual

context. Moreover special **interaction contexts** represent associations between a primary actor and a use case. They are used to identify those flows being the interaction starting points of a scenario.

Narrative Description	**Withdraw Cash**
Flow:	*Main Flow*
Associated Actors:	*Customer*
Contexts:	
Interaction Context	*triggered by Customer*
Flow of Events:	
1. Enter Card	*The Customer enters the card*
2. Include	*Authenticate::Log in*
3. Enter Amount	*The Customer enters the amount to withdraw*
4. Dispense Money	*The System dispenses the selected amount of money*
5. Return Card	*The System returns the card*

Narrative Description	**Authenticate**
Flow:	*Log in*
Associated Actors:	*Customer*
Contexts:	
Inclusion Context	*Incl. by Flow: Withdraw Cash*
Flow of Events:	
1. Alternative Anchor	*{Waiting for PIN}*
2. Enter PIN	*The Customer enters the PIN.*
3. Alternative Anchor	*{PIN Evaluation}*
Flow:	*Check PIN*
Contexts:	
Extension Context	*Extending {PIN Evaluation} if invalid PIN entered.* *Return to {PIN Evaluation} if PIN valid* *Return to {Waiting for PIN} if PIN invalid*
Flow of Events:	
1. Check PIN	*The System checks the PIN*

Figure 1: ATM narrative use cases.

5 SIMULATION ENVIRONMENT REQUIREMENTS

Simulation is a well-known prototyping technique and an intuitive means to assess the behavior of a system. The effectiveness of simulation to support modeling and quality assurance of use case based requirements specifications highly depends on the tool support (the use case simulator) and on the execution model that is the basis of the simulation.

In the following we briefly describe the core requirements on a use case simulator as well as on its underlying execution model.

5.1 Simulator Requirements

A use case simulator should be usable for multiple purposes. It should enable all stakeholders to perform step-wise inspections of the use cases and offer automatic analyses of the simulation information. Moreover the simulator should support regression based validity analyses of scenario instances after the use case model has been changed (as proposed by Glinz et. al. (2007)). We have identified the following major requirements:

- **Stakeholder-specific Visualization**
  The simulator should be usable by different stakeholders, especially domain experts and customers who are typically unaware of a formal simulation model. Therefore the simulation environment should provide a customizable visualization of the simulation runs. Moreover it should include perspectives for different kinds of stakeholders.
- **Flexible Integration with other Tools**
  During requirements engineering use cases and various other requirements artifacts are developed in parallel. Furthermore use cases are needed in later phases of the development process, like testing or GUI-design. Consequently the simulator should allow the integration of other requirements engineering tools and models and it should provide extension points to support specific development tasks in all phases of the development process.
- **Precise Execution of the Modeled Behavior**
  The simulator should execute the behavior specified in the use cases as precise as possible. It should be able to perform all legal scenarios and prohibit the simulation of scenarios that are not specified in the use case model.
- **Simulation of Incomplete Models**
  As use case specifications vary in their degree of completeness and formality the simulator should provide appropriate mechanisms to handle missing or informally specified information.
- **Simulation of Global System Behavior**
  Defects in the functional behavior are often caused by the interaction of use cases. Thus, the simulator should support the execution of multiple consecutive use cases to enable the analysis of the system's global behavior.

- **Reuse of Simulation Data**
  Scenario instances resulting from simulation runs should be used for static analyses, documentation of scenarios or as input to other development tasks like testing. Thus, whenever a simulation run is performed the simulator should be able to store all runtime information of the simulated scenario instances.
- **Analysis of the Simulation Runs**
  The simulator should support static analyses of simulations runs. This includes quantitative analyses like coverage measures as well as analyses of the structure of the execution model e.g. the identification of inconsistencies in behavior descriptions or the determination of change impacts. Additionally the simulator should support the revalidation of scenarios instances after the simulated use cases have been incrementally refined.

5.2 Execution Model Requirements

As denoted before, we prefer a structured textual use case notation because of the acceptance in industry. Although narrative use case descriptions have a defined structure, the resulting behavior specifications have no formal execution semantics. Therefore a formal execution model must be developed that matches the narrative use case concepts best. We see the following important requirements regarding the execution model:

- **Finite, Scalable Model**
  Because the simulator should be usable in real world projects the transformation of a narrative use case model to an execution model should always result in a finite model even if the specified behavior contains iterations or recursive definitions. Besides, the size of an execution model and the space and time complexity of the transformation algorithm should scale for big real world projects.
- **Robust Model Topology**
  The simulator should be usable from the early phases of use case development onwards. Therefore the topology of the execution model should be robust against changes. When additional information like a new condition or a new step is added or a sequence of steps is changed, the change should only have local effects to the execution model's topology. This is a prerequisite to perform impact analyses of a change to information derived from the execution model e.g. test cases or GUI prototypes.
- **Composition of Execution Models**
  The execution model should offer mechanisms to compose different partial models to one integrated model and to decompose a model into fragments. Ideally composition is done without copying the partial models to the integrated one.

6 SIMULATION CONCEPTS

In this section we describe our approach to simulate narrative use case descriptions based on the requirements described in section 5. The core of our simulation approach is a formal petri-net based execution model. To simulate a narrative use case model it is transformed to a respective execution model by a recursive transformation algorithm. During a simulation run all information about the simulated scenario instances is stored in a dedicated trace model. Thus different kinds of static analyses on the use case model, the corresponding execution model and the trace model as required in section 5 are possible.

6.1 Execution Model Concepts

The proposed execution model is based on colored petri-nets. First we briefly introduce colored petri-nets then we describe the topological and dynamic elements offered by the execution model.

6.1.1 Colored Petri-Nets

Petri-nets are a well known formalism to specify step-wise processes that include choice, iteration, and concurrent execution. They are bipartite directed graphs consisting of places, transitions, and directed arcs. Arcs run from a place to a transition or vice versa. The execution semantics of petri-nets is based on tokens, which are passed along the arcs of the net. Petri-nets, unlike other popular execution formalisms, have exact mathematical execution semantics, together with a well-developed mathematical theory for process analysis.
Colored petri-nets are an extension to standard petri-nets proposed by Jensen (1997, 2003). In colored petri-nets the tokens are distinguishable and the decision whether a transition can fire depends on the nature of the tokens. To be more explicit the tokens contain attributes and the values of those attributes are considered to decide whether a transition may fire or not. Although the expressiveness of colored petri-nets is identical to standard ones the topology

needed to express a certain behavior is noticeably smaller than the one of an equivalent standard petri-net.

6.1.2 Modeling the Topology

As a consequence to use colored petri-nets, our execution model consists of **transitions** and **places**. Both elements may have a reference to an element of the respective narrative use case model. Additionally we have introduced a concept to simplify the structure of the petri-nets. All transitions have an **incoming-** and an **outgoing-mode** describing their execution semantics. Both modes may either be "AND" or "XOR". Incoming-mode means that a transition can fire in "XOR"-mode if one incoming place has a token, or if all incoming places have tokens in "AND"-mode.

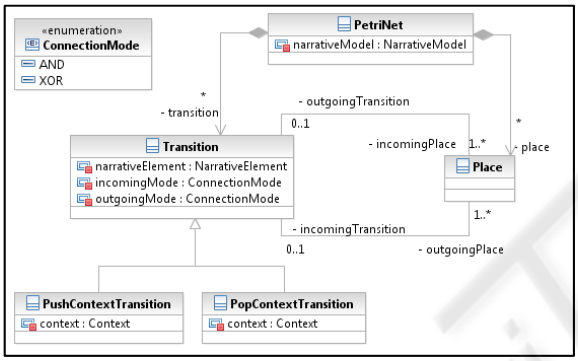

Figure 2: Execution model: Part Topology.

The outgoing-mode specifies how many token are created after the execution of a transition. A transition creates one token in exactly one of the outgoing places "XOR" in XOR-mode or a token in every outgoing place in "AND"-mode. Although colored petri-nets don't comprise incoming and outgoing mode, those modes don't contradict the theoretical concepts of colored petri-nets. Moreover the execution model contains two specific kinds of transitions called **push context transition** and **pop context transition.** The semantics of these elements will be explained in the next section. Figure 2 depicts the central topological elements of our execution model and their relationships.

6.1.3 Modeling Dynamic Behavior

The core elements of the dynamics are **tokens**. Tokens may have an arbitrary number of **attributes**. Each attribute has a name, a type, and a value. Additionally each token has a **context stack**. The context stack contains references to contexts from the narrative use case model (see section 4) that have been traversed before the creation of the token. The stack is used to determine where to continue the simulation after the end of a flow has been reached.

This concept (usage of runtime information) is needed because the decision where to continue can't be made only based on attribute values. For example, if we introduce an additional use case with an include-relationship to the existing use case "Authentication" in the ATM example, the simulator has to decide where to continue after the "Authentication" use case was completed successfully.

The context stack of a token may only be modified by the aforementioned Push- and PopContext-Transitions. A push context transitions adds a context to the stack, if a new context is reached during the simulation. A pop context transition removes the context from the stack after the end of the behavior in a certain context has been reached. Figure 3 depicts the elements of the execution model's dynamic view.

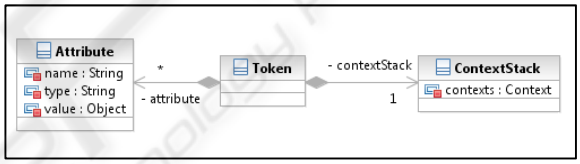

Figure 3: Execution model: Part Dynamics.

6.2 Generating the Execution Model

The transformation of a narrative use case model to a corresponding execution model is done recursively. The core idea of the transformation is that only actions are **behavioral atoms**. All other elements (e.g. flows) of narrative use case models are considered **behavioral frames** enclosing at least one **behavioral fragment**. Each behavioral fragment consists of a set of narrative use case model elements. This means: flows enclose their contained events, contexts enclose the flows whose usage they describe, and anchors enclose the contexts they are connected to.

The transformation algorithm itself is based on a set of generic **transformation strategies**. For each narrative model element type the respective transformation strategy defines the execution model elements that are created. That way, every narrative use case model element is transformed separately according to the respective transformation strategy into one or more transitions connected by places.

The transformation of a narrative use case model starts at the interaction starting points of its scena-

rios. Then it moves along the scenarios to their ends. At first the transformation algorithm connects all interaction contexts of the narrative use case model to an initial execution model frame. Then it transforms the behavior described in the flows of the narrative use cases in a depth first manner. The flows that are directly connected to an interaction context are transformed first. Other flows are processed when respective anchors to the flows are reached.

During the transformation every behavioral frame is transformed into an initial and a final transition. The initial transition is the predecessor of the enclosed behavioral fragments and the final transition is the successor. Thus the execution model evolves from outside-in, since in each transformation step elements are added to the beginning as well as to the end of the execution model.

During the transformation every element of the narrative use case model is transformed exactly once. Whenever an element is reached for a second time it is not transformed again. Instead, a reference to its execution model representation is generated.

Thus the transformation of a behavioral fragment terminates when the last event of a flow is transformed or when an element is reached that already has an execution model representation. Since a behavioral frame may contain several behavioral fragments, its transformation terminates when all behavioral frames are transformed. Then the transformation algorithm moves on to the next element of the narrative use case model.

Thus the transformation algorithm terminates after each flow reachable from any scenario of the narrative use case model was processed once. After completion of the transformation all elements are transformed and all connections between the elements are established. Therefore all scenarios described in a narrative use case model can be performed based on the respective execution model.

In the following we briefly explain the transformation strategies for the narrative use case element types. Figure 4 shows the caption used in the figures of this section.

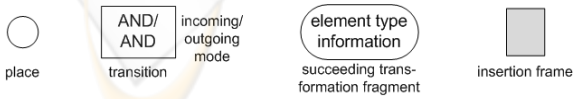

Figure 4: Caption of the transformation strategies.

White colored elements are created during the transformation of a behavioral frame. Stroked elements depict sections that are transformed in a succeeding transformation step. Finally the gray boxes in the left and right corners of a transformation fragment represent the frame into which the elements are inserted.

6.2.1 Initial Execution Model

The initial execution model contains all scenarios of the narrative use case model. Thus it represents the global behavioral frame. Its initial transition is called start delimiter and its final transition is called end delimiter of an execution model (see Figure 5). The start delimiter is connected to the initial place, which holds the first token when a simulation run is started. The end delimiter is connected to the final place that holds the last token at the end of a simulation run. Moreover the start and the end delimiter are connected by a place to support the execution of multiple use cases consecutively.

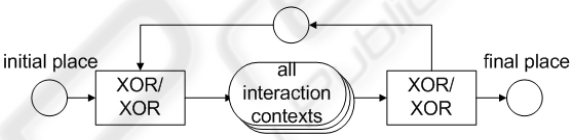

Figure 5: Initial execution model.

6.2.2 Flows

A flow encloses all its contained events. Since it may be executed in several different contexts the initial transition of a flow has an XOR-incoming mode. The final transition has an XOR-outgoing mode because the execution may only continue in the context which the flow was called from in the first place.

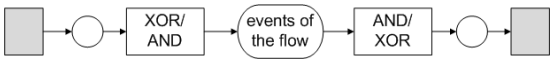

Figure 6: Flow transformation fragment.

6.2.3 Contexts

The execution model representation of all kinds of contexts is identical. Its initial transition is a push context transition and has an XOR-incoming mode, since a context may be connected to several incoming anchors. The final transition is a pop context transition with an XOR-outgoing mode, since the execution may continue at only one outgoing anchor.

Figure 7: Context transformation fragment.

6.2.4 Events

Actions are behavioral atoms and can be represented by single transitions, because they don't contain a behavioral fragment.

Inclusion- and extension anchors are on the contrary behavioral frames.

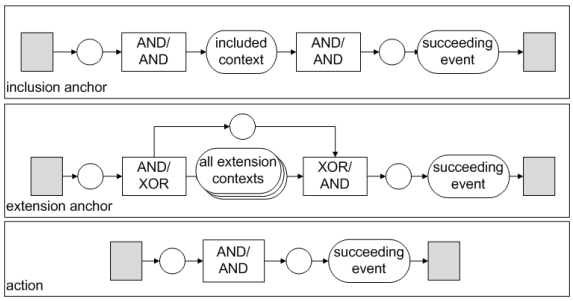

Figure 8: Event transformation fragments.

An inclusion anchor is connected to only one flow which always must be executed when the inclusion is reached. An extension anchor on the contrary may be connected to several flows which may or may not be executed depending on the conditions that are specified in the extension contexts of the connected flows. As a consequence the initial and the final transition of extension anchors are connected directly and they have an XOR-outgoing mode to decide which extension to select (Figure 8).

6.2.5 Additional Transformation Concepts

As denoted before each element is transformed only once. This approach guarantees that the transformation creates a finite model which scales with the size of the narrative use case model. The reference to an already transformed element is established by creating two petri places to connect the behavioral frame to the existing element (Figure 9).

Finally, so called null-symbols are inserted whenever the end of a recursion is reached. This is the case if the end of a flow is reached or in any situation where information is missing (e.g. a missing inclusion context at an inclusion is replaced by a null-symbol).

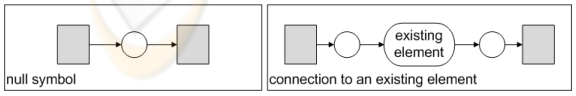

Figure 9: Additional concepts.

6.3 Use Case Simulation

The simulation of a narrative use case model is performed using its execution model. A simulation run is started by initializing the execution model with a single token at the start delimiter. During the simulation run the token is passed along the petri-net. At points where multiple paths could be traveled, the decision is made based on the values of the attributes in the current token. The simulator can check whether a decision can be made automatically based on formalized information available at the decision point. If this is not possible the user has to decide where to continue the simulation. A simulation run is finished after all tokens are passed to the end delimiter of the execution model. During a simulation run all runtime information is stored in a trace model which we present briefly in the following.

6.3.1 Trace Model

Figure 10 depicts the elements of the trace model. A **trace model** instance usually contains information about several simulation runs of different scenario instances that have been performed together – e.g. a set of scenarios that have been inspected during a use case review. This enables analyses of the correlation of simulation runs like coverage measurements. In the trace model the simulation information of each scenario instance is stored in a separate **trace**. All traces consist of a set of **trace steps** containing the details of a simulation run. Every trace step holds a reference to an element of the topology of the execution model and a reference to a token. Thus a simulation run is fully qualified by a trace and can be executed again based on the trace information. Moreover all trace elements may hold references to **simulation information** elements. Simulation information elements are used to add special purpose information to a trace – e.g. review findings.

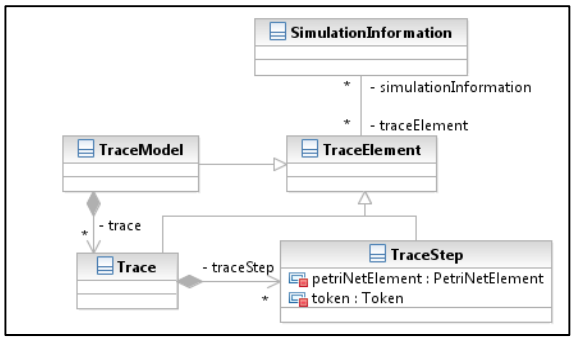

Figure 10: Elements of the trace model.

7 EXAMPLE

In this section we point up our simulation approach alongside the ATM example introduced in section 4. Figure 11 shows the execution model for the two narrative use cases. The transitions are labeled according to their referenced narrative use case model elements. The dotted arrows visualize a simulation run on the execution model. Additionally the context stack of the simulation run is denoted at the Push- and pop context transitions of the execution model. Figure 12 depicts the trace recorded during the simulation run.

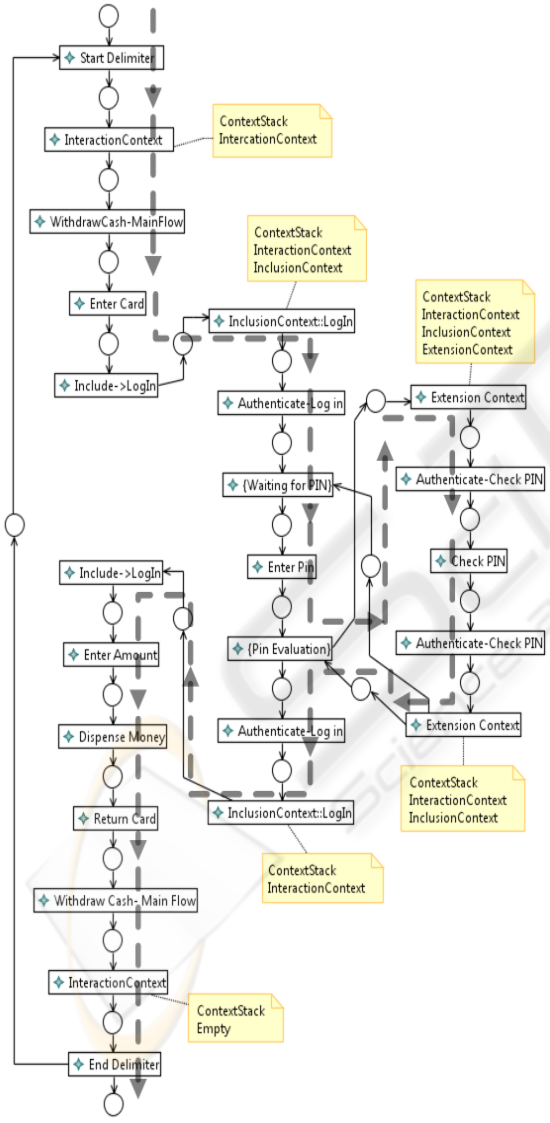

Figure 11: ATM execution model & simulation run.

8 TOOL SUPPORT

We have integrated a simulation environment for narrative use case models called USE (Use case Simulation Environment) into NaUTIluS, the use case editor of the Eclipse based development platform ViPER (ViPER, n.d.). USE is based on an EMF-implementation (EMF, n.d.) of the execution and the trace model. USE consists of a simulator and a set of views to visualize simulation information.

After a narrative use case model is transformed to a corresponding execution model the simulator can run simulations.

Simulation Step	Decision	Token ContextStack
Start		Empty
InteractionContext::Withdraw Cash::Main Flow		IC::Withdraw Cash
Enter Card		IC::Withdraw Cash
Include::Authenticate::Log In		IC::Withdraw Cash IN:: Authenticate::Log In
InclusionContext::Withdraw Cash::Main Flow		IC::Withdraw Cash IN:: Authenticate::Log In
{Waiting for PIN}	none	IC::Withdraw Cash IN:: Authenticate::Log In
Enter PIN	Manual set (PIN-NR 1142)	IC::Withdraw Cash IN:: Authenticate::Log In
{PIN Evaluation}	Automatic check (PIN not valid)	IC::Withdraw Cash IN:: Authenticate::Log In
ExtensionContext::Log In::Check PIN		IC::Withdraw Cash IN:: Authenticate::Log In EX:: Log In::Waiting for PIN
Check PIN	Manual set (PIN valid)	IC::Withdraw Cash IN:: Authenticate::Log In EX:: Log In::Waiting for PIN
ExtensionContext:: Log In::Check PIN	Automatic check (PIN valid)	IC::Withdraw Cash IN:: Authenticate::Log In
{PIN Evaluation}		IC::Withdraw Cash IN:: Authenticate::Log In
InclusionContext::Withdraw Cash::Main Flow		IC::Withdraw Cash
Enter Amount		IC::Withdraw Cash
Dispense Money		IC::Withdraw Cash
Return Card		IC::Withdraw Cash
InteractionContext::Withdraw Cash::Main Flow		Empty
End	Manual set (Finish)	Empty
Terminated		

Figure 12: Trace of a scenario instance.

Whenever a simulation reaches a decision point (different scenarios are possible) the simulator decides automatically based on the information stored in the tokens or asks the user to decide which scenario should be pursued. During a simulation run the simulator stores trace information and provides feedback to the user about the current simulation via standard Eclipse notification mechanisms. Moreover USE defines several extension points to customize the simulator for specific tasks.

Figure 13 shows a screenshot of USE with some visualization views.

9 EVALUATION

The herein proposed simulation approach completely meets the requirements motivated in section 5. The execution model has a precisely defined execution semantic, its topology is robust against changes and supports incremental refinement of the use cases, because changes only have local effects.

The explained recursive transformation algorithm scales for real world narrative use case models and is able to handle missing information or incomplete behavior by inserting so called null-symbol placeholders.

The simulations can be performed using the simulation environment USE which stores all runtime information in a dedicated trace model for later analyses.

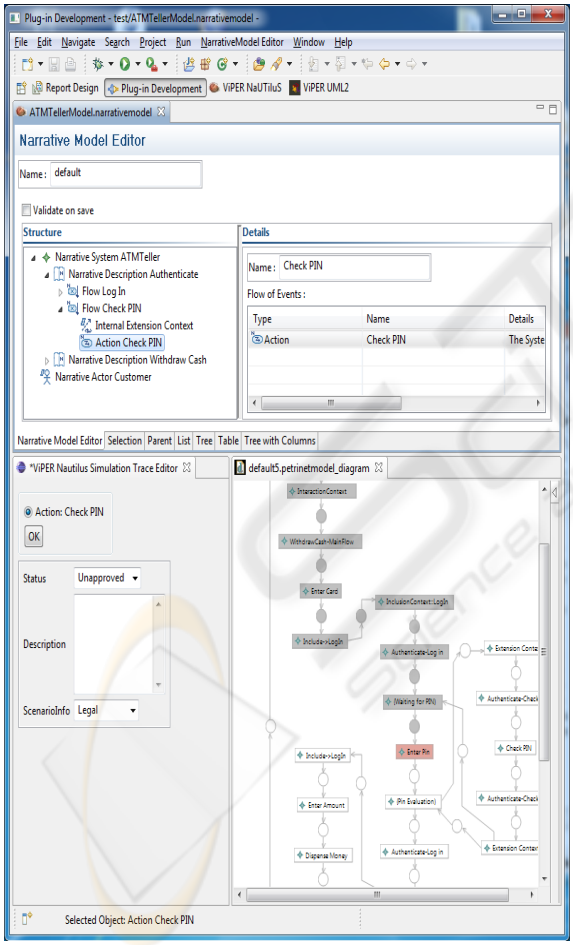

Figure 13: USE screenshot.

Finally, USE is easily connectable with other requirements models as it relates on Eclipse framework standards and provides highly flexible mechanisms to configure custom visualization.

Our simulation approach has three outstanding advantages compared to state machine centered techniques.

1. The concept of petri-nets and its token semantic strongly reflects the step-by-step idea of use cases. While state machines rely on states that are not explicitly modeled in use cases, our approach maps the steps of a narrative use case directly to transitions of a petri-net. Thus the transformation algorithm is straightforward and a comprehensible visualization of the behavior is possible.
2. Our execution model is context aware since the execution contexts are stored in the tokens during the simulation. Therefore our approach can handle situations where the control flow is not directly derivable form the system state. State machines would have to introduce implicit states to cover context aware behavior. Those implicit states are difficult to derive and increase the complexity of the respective state machine.
3. The topology of colored petri-nets is robust against changes in the referenced narrative use case model because changes only have local effects.

We have evaluated the presented use case simulation approach alongside with USE in some academic projects performed in student's master thesis. The simulation has been well accepted and first experience shows that behavior simulation is an intuitive way to analyze narrative use case models. The simulation-based approach has especially proven useful in projects with very detailed use case descriptions containing many (complex) alternatives to the normal use case behavior.

10 CONCLUSIONS AND OUTLOOK

In this paper we have introduced a model-based simulation approach for narrative use cases. We have pointed out the most important requirements of a useful and efficient use case simulation environment. Furthermore we have presented a petri-net based execution model, a model-based transformation algorithm and a narrative use case simulation strategy. Finally we have sketched the simulation environment that we have developed based on the presented concepts. First experiences of our simulation approach in student projects are promising.

Currently we are evaluating the usefulness and acceptance of the presented approach in different small and medium size projects we perform at the faculty and with different industrial cooperation partners.
Until now we are using the simulation environment only to support the inspection of use cases. In the future we plan to enlarge our approach to an integrated requirements specification technique. Besides we are currently doing first experiments to generate GUI prototypes based on the simulation information and to generate test specifications based on instance scenarios created with the simulation environment.

Finally we are currently developing tool support for static analyses on the narrative use case model as well as on the corresponding execution model.

REFERENCES

The Standish Group, 2003. Chaos chronicles v3.0 Technical report.

Weidmann, C., Hoffmann, V. & Lichter, H., 2009. Einsatz und Nutzen von Use Cases - Ergebnisse einer empirischen Untersuchung. In Softwaretechnik-Trends, Band 29, Heft 2, pp. 62 -67.

Jacobson, I., 1987. Object-oriented development in an industrial environment. OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications, *ACM Press*, New York, pp. 183-191.

Jacobson, I., 2004. Use Cases - Yesterday, today, and tomorrow. Software and System Modeling, vol 3, pp. 210-220.

Cockburn, A., 2000. Writing Effective use cases, Addison-Wesley.

Armour, F. & Miller, G., 2001. Advanced use case Modeling Volume One, Software Systems. Addison-Wesley Longman Publishing Co., Inc., Boston.

Kulak, D. & Guiney, E, 2003. Use Cases: Requirements in Context. Addison-Wesley Longman Publishing Co., Inc., Boston.

Bittner, K., & Spence, I., 2003. Use Case Modeling. Addison-Wesley Longman Publishing Co., Inc., Boston.

Jacobson, I., Booch, G. & Rumbaugh, J., 1999. The Unified Software Development Process, Addison-Wesley Longman Publishing Co., Inc., Boston.

Denger, C. & Paech, B., 2004. An Integrated Quality Assurance Approach for Use Case Based Requirements, *Proceedings Modellierung* 2004, Marburg, pp. 307–308.

Sutcliffe, A.G., Maiden, N.A.M., Minocha, S. & Manuel, D., 1998. Supporting Scenario-Based Requirements Engineering, *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1072-1088.

Lee, W.J., Cha, S.D. & Kwon, Y.R., 1998. Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering, *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp 1115-1130.

Williams, C., Kaplan, M., Klinger, T. & Paradkar A., 2005. Toward Engineered, useful use cases. Journal of Object Technology, Special Issue: Use Case Modeling at UML-2004, vol 4, pp. 45-57.

Glinz, M., Seybold, C. & Meier, S., 2007. Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models. Dagstuhl-Workshop Modellbasierte Entwicklung eingebetteter Systeme (MBEES 2007). Informatik-Bericht 2007-01, TU Braunschweig, pp. 103-112.

Whittle, J. & Jayaraman, P.K., 2006. Generating Hierarchical State Machines from use case Charts, RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06), *IEEE Computer Society*, Washington, DC, pp. 16-25.

Jorgensen, J.B. & Bossen, C., 2004. Executable Use Cases: Requirements for a Pervasive Health Care System. *IEEE Software*, 21(2), pp. 34-41.

Somé, S., 2006. Supporting use case based requirements engineering. Information and Software Technology 48, pp. 43-58.

Zhao J. & Duan Z.: Verification of Use Case with Petri Nets in Requirement Analysis, ICCSA (2), pp. 29-42.

Hoffmann, V., Lichter, H. & Nyßen, A., 2009. Towards the Integration of UML-and textual Use Case Modeling, Journal of Object Technology, vol. 8, no. 3, pp. 85-100.

Jensen, K., 2003. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol.1 (Monographs in Theoretical Computer Science. An EATCS Series). *Springer*.

Jensen, K. 1997. A Brief Introduction to Coloured Petri Nets, Tools and Algorithms for the Construction and Analysis of Systems, pp. 203-208.

ViPER project site, http://www.viper.sc.

Eclipse Modeling Framework project site, http://www.eclipse.org/modeling/emf/