Electronic Communications of the EASST
Volume 58 (2013)

Proceedings of the
12th International Workshop on Graph Transformation
and Visual Modeling Techniques
(GTVMT 2013)

Interactive Strategy-Based Validation of Behavioral Models

Ralf Teusner, Gregor Gabrysiak, Stefan Richter, and Stefan Kleff

12 pages

Guest Editors: Matthias Tichy, Leila Ribeiro
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Interactive Strategy-Based Validation of Behavioral Models

Ralf Teusner, Gregor Gabrysiak, Stefan Richter, and Stefan Kleff

{firstname.lastname}@hpi.uni-potsdam.de
Hasso Plattner Institute for Software Systems Engineering,
University of Potsdam, Germany

Abstract: When behavioral models are derived automatically based on observed stakeholder interactions, requirements engineers need to validate whether the stakeholders agree with the synthesized behavioral models. Allowing stakeholders to experience such models through simulation and animation allows them to comment on, amend to and correct these models. However, to ensure an efficient stakeholder validation, the simulation has to be guided instead of confronting the user with random situations over and over again.
In this paper, we present a strategy-driven simulator capable of guiding the execution of behavioral models based on graph transformations. By analyzing either the overall structure of a partial state space (look ahead) or by performing an in-depth analysis of the states therein, the simulator is able to determine which transformations should be executed next to continue on the most promising path through the overall state space. The discussed implementation is illustrated with a case study.

Keywords: Requirements Simulation, Stakeholder Validation, Simulation Strategy

1 Introduction

Capturing requirements and scenarios in formal models such as graph transformations introduces a communication barrier between requirements engineers and stakeholders. Nevertheless, the latter have to understand what was elicited and specified to identify conflicts, find errors or provide feedback. Especially in highly collaborative scenarios involving multiple stakeholders and their individual perspectives on what is to be accomplished, gathering knowledge about the stakeholders' scenarios is inherently complex. In our earlier work, we presented a combined animation [GGS09] and simulation [GGS10, GHG12] approach capable of deriving graph transformations that represent observed activities (cf. *generalization* in [Hec06]) within such collaborative scenarios. Based on the differences between two succeeding states *prior* and *after* a stakeholder activity was observed, a behavioral specification is derived which encapsulates the changes associated with the activity. For this specification, *story patterns* [KNNZ00] are used. Furthermore, these story patterns can later on be rearranged and replayed to simulate any stakeholder activities that were already observed. By executing the corresponding graph transformations of stakeholders, their actions and interactions can be simulated and visualized for stakeholders participating in the simulation. Eventually, the activities of all stakeholders have been observed and captured in story patterns. Then, the requirements engineers end up with story patterns which, when executed, can simulate the commonly agreed upon scenarios that can unfold for the stakeholders.

Being able to replay specific activities of any of the roles involved in the explored scenarios enables individual stakeholders to participate as one of the roles. As a participant, this stakeholder is then able to experience and validate, whether the behavioral models that are executed are correct, consistent from his point of view and complete, i.e. covering at least all of his known information for the considered scenario.

To ensure that the paths chosen during the simulation of a scenario do not always lead to the same, already validated situations, the decisions of the simulator, i.e. of the individual roles that are simulated, need to be guided in different directions to provoke the participant to play-in alternative scenarios. By navigating through the state space, the simulator can specifically guide the stakeholder through a conflicted scenario. A scenario has to be considered as conflicted if it contains states which yield possibly wrong outcomes or unwanted effects. Examples are traces leading to dead ends and situations that are impossible in reality such as having a negative amount of goods or inconsistencies such as an inquiry that is both accepted and rejected. Thereby, unidentified constraints which the participants can point out afterwards are then be captured as "forbidden situations" that are avoided during future simulations. By generally avoiding dead ends, on the other hand, the simulator can ensure that the stakeholder experiences a scenario that can successfully terminate.

Such a scenario simulation can be guided more effectively if the consequences of the available decisions are known. If the complete state space, i.e. information about all possible states and the transitions between them, is explored, all eventually occurring consequences and their side-effects are known. However, it is not feasible to compute all possibilities beforehand, especially since the state space might be infinite. Still, to avoid solely relying on impromptu or even random decisions, a limited preview of the state space (i.e. *look ahead*) is already helpful. Moreover, the state space that can be explored depends on the story patterns which were already captured, i.e. played in by participants. Thus, the state space supports a guided simulation only in cases in which the participant's actions were known in the form of expected inputs for the simulation. Since participants can interact with the simulation within their visualization, they can directly change the state of the simulation (cf. *play-in* [HM03]). In such cases, the simulator is confronted with inputs that were unexpected for the current state. Still, such an input may be equivalent to a sequence of other activities which implies that the actual follow-up state of the simulation is already included in the state space. Consequently, the participant played in an alternative which would result in a new transition within the state space. Still, if the participant's activity was new and without an equivalent sequence of other activities, then the simulator would be in a follow-up state which was not covered by any state space that might have been calculated beforehand. In such a situation, a feasible guidance of the simulation can only be provided, if a look ahead can be computed fast enough at runtime so that the simulation is still perceived as interactive for the participating stakeholders.

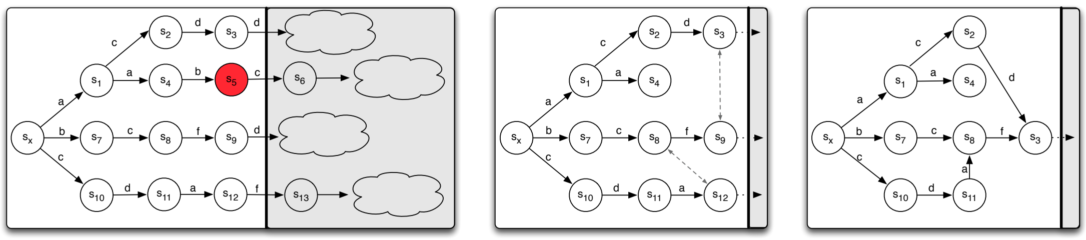
The stakeholders' perspective of the simulation is discussed in [GGS09]. This partial perspective on the simulation is not sufficient for the requirements engineer, who needs a more technical perspective on the simulation, so he should be supplied with an interactive visualization of not only the current state of the simulation, but also of its look ahead. The conceptual contribution of this paper is a bounded look-ahead for graph transformation systems which incorporates strategies and cost functions to explore the potentially infinite state space more effectively. The main contribution is an Eclipse-based simulation environment, that supports engineers to interactively validate the requirements by automatically guiding the simulation. The concepts underlying the strategy-based simulation approach are presented in Section 2. Then, Section 3 illustrates the implementation using a steelworks case study. Afterwards, related work is discussed in Section 4 which focusses on graph transformation approaches. Finally, conclusions are drawn and future work is outlined in Section 5.

2 Concepts

2.1 Navigation and Reduction

To fulfill the outlined requirements, several different concepts are employed. An interactive navigation is only feasible, for technical as well as user-interface reasons, if the graph-structure in scope is kept small enough so that the time for the computation of what to do next and the actual execution take less than one second. Nevertheless, the regarded state space has to be complete and correct concerning the already established constraints. The user's main focus of interest lies in all future states outgoing from the current one. Since the amount of future states might be unlimited, possible future states are limited by the introduction of a maximal look ahead depth, after which the simulation of further states is simply aborted. Another reduction is achieved by aborting the simulation behind states that violate predefined conditions as illustrated by s_5 in Figure 1a. Despite that, the growth of states still correlates exponentially with the chosen look ahead depth. By merging two or more instances of an identical states into one, the exponential growth can at least be postponed (cf. Figures 1b and 1c). This results in a potentially cyclic graph, but reduction of branches enhances lucidity and performance significantly. The effectiveness of this technique is determined by the depth, at which the merging takes place (the earlier the better).

Stakeholders should not encounter situations which are known to be impossible or invalid, e.g. situations in which a budget is exceeded or a time constraint is violated. Usually, such situations can be specified as partial states or as a sequence of states which must not occur. To avoid these situations, the simulator has to check whether one of them can be matched in one of the states of the look ahead. If so, the corresponding path can be excluded, i.e. *cut-off* as early as possible [HHV10], from the options that are available to the stakeholder from the current state of the simulation onwards.



(a) A look ahead (depth=3) which contains a forbidden state (s_5) and the infinite state space (b) Simplified look ahead with a depth of 3 (c) Final look ahead after merging (s_3, s_9) and (s_8, s_{12})

Figure 1: Steps of reduction before scoring the look ahead

2.2 Guidance and Scores

If the available story patterns are underspecified, they will be matched and executed more often than necessary. Consequently, states with many available story patterns might imply that some of these patterns are underspecified and need to be revised. However, only a stakeholder can decide which of the patterns should match in the corresponding state and which patterns require a stricter precondition. To support a stakeholder in identifying such patterns, a strategy that scores the number of alternatives is required which chooses the path leading to the most alternatives for further execution. This can be determined based on the look ahead: by scoring the structure of the look ahead, e.g. by adding up the number of outgoing transitions of each of its states, the simulator can simply choose the sequence of transitions which provides the most alternatives.

To enable a participating stakeholder to resolve a specific situation (e.g., *What do you do if you are over-budget?*), the simulator has to reach a corresponding state. The requirements engineer needs to create a strategy that scores the value of the corresponding resource, in this case "budget", negatively which, in turn, favors all activities that decrease the value of this resource. Scoring is based on increasing and decreasing amounts of resources, which are manipulated through actions, i.e. the application of story patterns. All scores of states in the look ahead are relative to the the simulation's current state s_x. By assigning a weight w_i to each individual resource r_i, the sum of all resources times their individual weight is a suitable scoring function for each state. This score is based on the value of the individual resources of the current state s_x of the simulation. Equation 1 illustrates this function for a state s_y with r_i^x representing the value of resource r_i in state s_x. If, for instance, a resource r_1 named "budget" would have a value of 5,000 in s_x, a value of 3,000 in a potential follow-up state s_y would be scored higher since it is 2,000 times w_1 closer to the goal of the strategy.

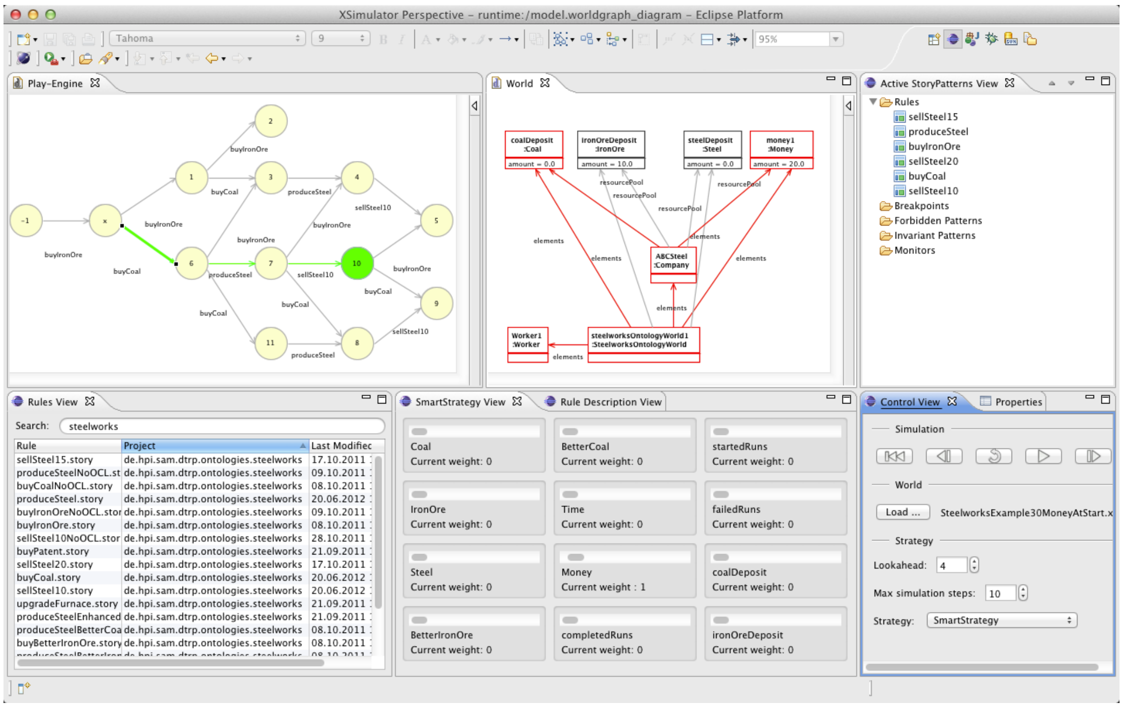
$$score(s_x \rightarrow s_y) = \sum_{i=1}^{|Resources|} w_i \times (r_i^y - r_i^x) \qquad (1)$$

In the overall state space, only some specific paths represent valid, successful scenarios capturing how stakeholders interact to achieve their common goal. By looking at the complete (if not infinite) state space, the simulator can find all states in which this goal is satisfied. Knowing these states, the simulator is able to choose the path that satisfies the stakeholders' overall goal best. Since computing capabilities are limited, the simulator has to decide which path is most likely to lead to success based on the feasibly available look ahead.

3 Implementation and Performance-Evaluation

The implementation of the discussed concepts is structured into three layers. All scoring, monitoring and decision-making is located on the topmost layer in the form of strategies. The second layer computes the look ahead, thereby reducing it in compliance with the aforementioned concepts of state-merging and pruning of states that violate the predefined constraints. Such constraints can be expressed as *forbidden* states which have to be avoided by the simulator. Additionally, these state specifications can also include OCL[1] statements which have to hold. The

[1] Object Constraint Language (http://www.omg.org/spec/OCL/)



Rules View

Search: steelworks

Rule	Project	Last Modifiec
sellSteel15.story	de.hpi.sam.dtrp.ontologies.steelworks	17.10.2011
produceSteelNoOCL.st	de.hpi.sam.dtrp.ontologies.steelworks	09.10.2011
buyCoalNoOCL.story	de.hpi.sam.dtrp.ontologies.steelworks	08.10.2011
produceSteel.story	de.hpi.sam.dtrp.ontologies.steelworks	20.06.2012
buyIronOreNoOCL.stor	de.hpi.sam.dtrp.ontologies.steelworks	09.10.2011
buyIronOre.story	de.hpi.sam.dtrp.ontologies.steelworks	08.10.2011
sellSteel10NoOCL.stor	de.hpi.sam.dtrp.ontologies.steelworks	28.10.2011
buyPatent.story	de.hpi.sam.dtrp.ontologies.steelworks	21.09.2011
sellSteel20.story	de.hpi.sam.dtrp.ontologies.steelworks	17.10.2011
buyCoal.story	de.hpi.sam.dtrp.ontologies.steelworks	20.06.2012
sellSteel10.story	de.hpi.sam.dtrp.ontologies.steelworks	20.06.2012
upgradeFurnace.story	de.hpi.sam.dtrp.ontologies.steelworks	21.09.2011
produceSteelEnhanced	de.hpi.sam.dtrp.ontologies.steelworks	21.09.2011
produceSteelBetterCoa	de.hpi.sam.dtrp.ontologies.steelworks	08.10.2011
buyBetterIronOre.story	de.hpi.sam.dtrp.ontologies.steelworks	08.10.2011

SmartStrategy View | Rule Description View

Coal Current weight: 0	BetterCoal Current weight: 0	startedRuns Current weight: 0
IronOre Current weight: 0	Time Current weight: 0	failedRuns Current weight: 0
Steel Current weight: 0	Money Current weight : 1	coalDeposit Current weight: 0
BetterIronOre Current weight: 0	completedRuns Current weight: 0	ironOreDeposit Current weight: 0

Control View | Properties

Simulation

World

Load ... SteelworksExample30MoneyAtStart.x

Strategy

Lookahead: 4

Max simulation steps: 10

Strategy: SmartStrategy

Figure 2: Screenshot of the Eclipse perspective of the implementation

bottom layer is responsible for matching and executing story patterns on specific states. For this task, it relies on Giese et al.'s Story Diagram Interpreter [GHS09]. Further, this layer supplies caching algorithms and incremental transitions between states to speed up the most expensive part of the simulation, the computation of the look ahead.

For the requirements engineers overseeing individual stakeholder sessions, the implementation offers an Eclipse perspective (cf. Figure 2). In this Eclipse perspective, parameters of a simulation such as the depth of the look ahead or the scoring value of specific resources can be changed at run-time to adjust the strategy employed by the simulator. The engineers' visualization of the look ahead and the proposed path through the look ahead are updated accordingly. To avoid repeated computations, the look ahead that was previously explored is cached and re-used, if an overlap between the old and the new look ahead exists. States that are located before the currently active state and are not part of the actual path taken to the active state (for example alternative states to s_x after state s_{-1} in Figure 2), are not cached to reduce memory consumption. The merging process uses a simple fingerprinting approach to reduce the necessary amounts of computation: the count of inner objects and relations of a state are multiplied by different prime numbers and summed up afterwards. A more detailed and expensive *equals* comparing individual objects is only executed if the fingerprints match. While we are aware that the fingerprinting algorithm could be more sophisticated to further improve the performance gains, this approach already suffices for our requirements.

3.1 Example Scenario: Running a Steelworks

The aforementioned principles will be shown on a deliberately small example without distracting elements. To produce and sell steel, iron ore and coal are required. Since both cost money, it is necessary to plan accordingly to be able to produce steel and sell it with a profit. In this case study, four story patterns are available: *buy coal*, *buy iron ore*, *produce steel* and *sell steel*. While both resources cost 10k$ for ten units each, selling the resulting steel yields 30k$. The domain model is illustrated in Figure 3, Figures 4 and 5 shows the story patterns which realize the process.

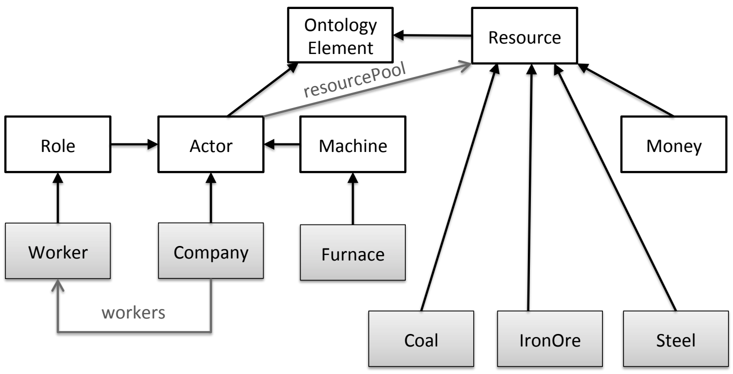


Figure 3: Metamodel of the case study

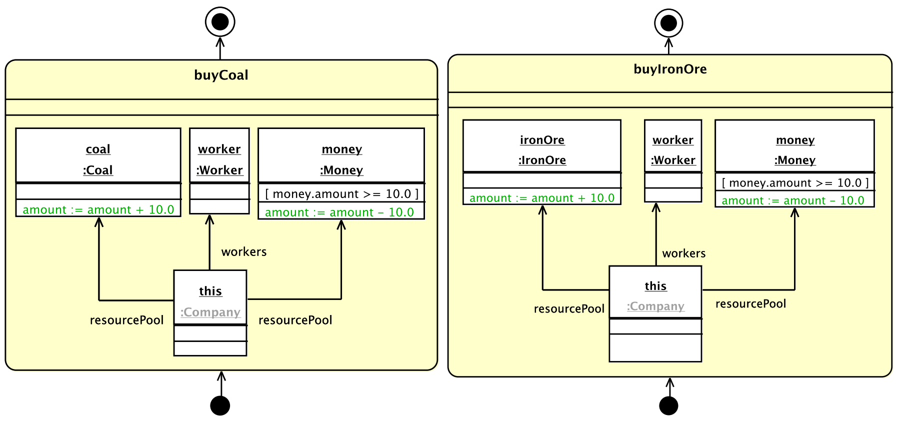


Figure 4: Story patterns for buying coal and iron ore

3.2 Strategies for a Steelworks

After the simulation was initialized using an initial state (s_x in Figure 8), the simulator computes a look ahead with a default depth of four to explore which transitions are possible. Repeatedly buying only one resource leads to states in which nothing else can be bought and nothing can

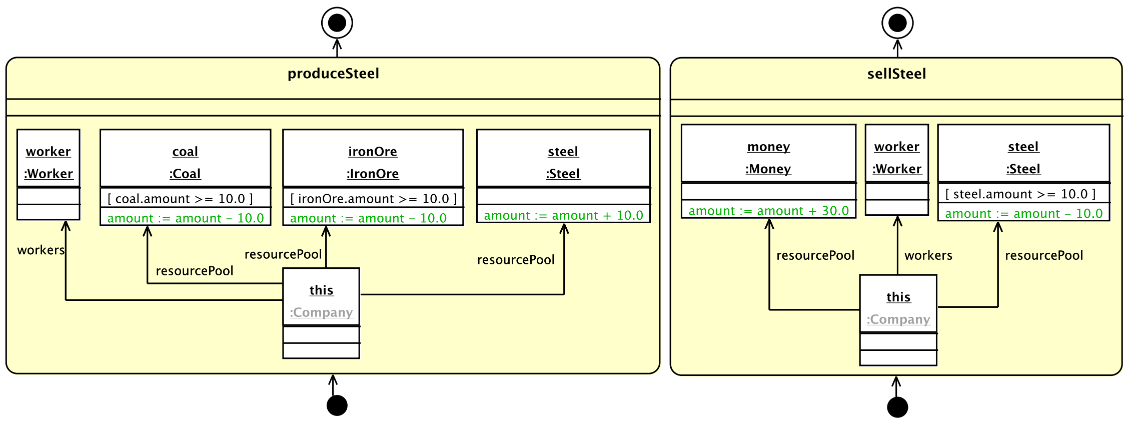


Figure 5: Story patterns for producing and selling steel

be produced or be sold. Consequently, these states (s_3 and s_{13} in Figure 8) are dead ends. Since the order of purchases is irrelevant in this scenario, buying both resources once leads to identical states which can be merged (s_6).

After the look ahead is computed, all potential follow-up states and their paths can be scored to allow the simulator to choose the path most suitable to fulfill the goals of the current simulation strategy. The goal of the steelworks scenario is to maximize the profit, i.e. the amount of money they have. In this case, the user would set the parameters to assess the resources so that money is the only resource of interest (i.e. by giving it a weight of 1 and all other resources a weight of 0). Thus, a weight can be assigned to each subclass of `Resource` that is defined in the metamodel representing the domain (cf. Fig. 3). The simulator chooses the transformations which maximize the score based on these weights. Consequently, a sequence of buying only the required resources, producing and then selling the steel is the most promising one. Although both buying actions are scored negatively, all paths containing sale of steel have an higher overall score than the starting state. In the simulated look ahead, s_{10} yields the highest score and is therefore proposed by the strategy. In more complex scenarios with more options available, the depth of the look ahead would have to be increased. Since long sequences of actions might not pay off if their benefit is not covered within the look ahead, such sequences would never be started.

The implemented Eclipse perspectives (Fig. 2) contains the following views for supervising the simulation: In the upper left view is an illustration of the sequence of the already visited states and the look ahead calculated from the current state (referred to as s_x). In this view, the optimal path is highlighted in green. The requirements engineers can select transitions from the look ahead to inspect them in detail in the upper middle view which presents the current state of the simulation or any state of the look ahead from which this transition is triggered.

In the lower left view, the rules that were already captured by observing stakeholders are listed. These can be included in any simulation session by dragging and dropping them into the upper right view under *Rules*. Also, forbidden situations and other variants, which are not within the scope of this paper, can be included for a simulation session.

The simulation is initialized by providing an initial state, setting a depth of the look ahead, and an optional maximum number of simulation steps after which the simulator should stop. This is done in the lower right view, which also illustrates a list of all available strategies to choose from for the simulation session. To influence the selected strategy, the lower middle view shows all resources next to corresponding sliders which enable the requirements engineers to assign weights to the individual resources.

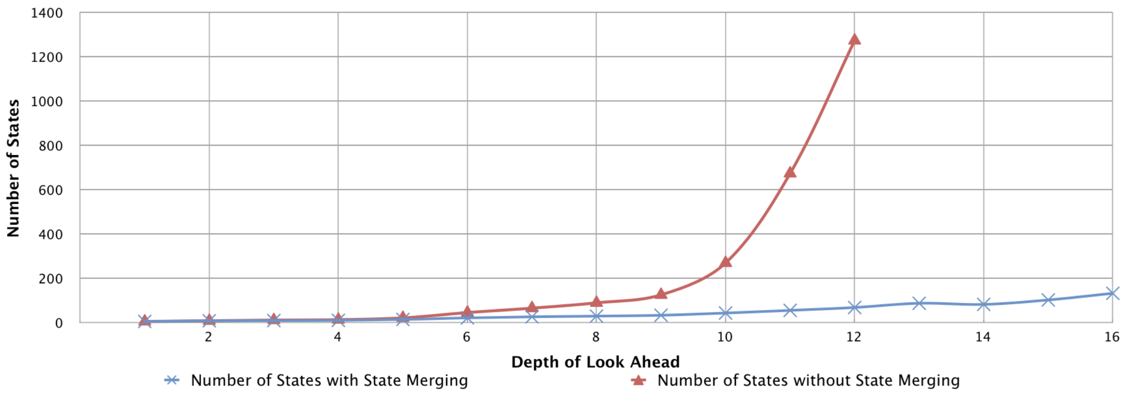


Figure 6: Number of states for different look ahead values – with and without state merging

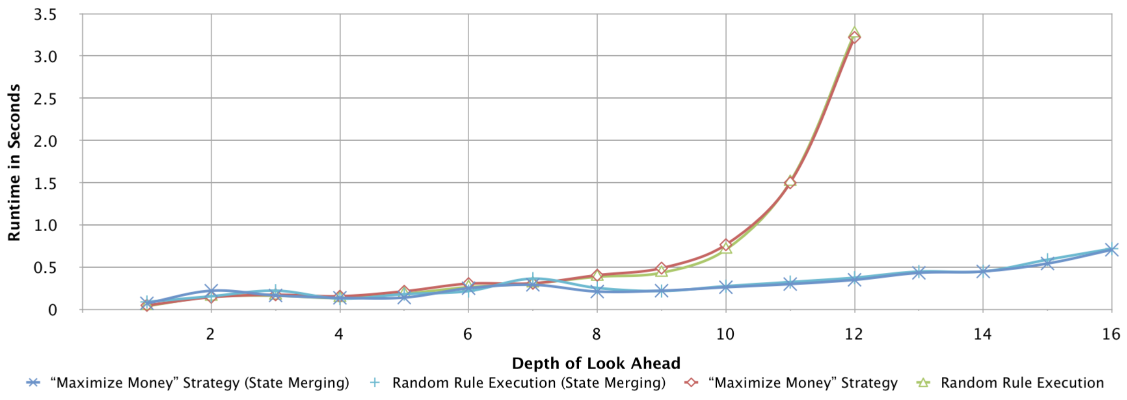


Figure 7: Runtime of random rule execution compared to a strategy-based execution for different look ahead values – with and without state merging

On a MacBook Pro (Mac OS 10.6) with a 2.4 GHz Core2Duo processor and 8 GB RAM (JVM 1.6, Eclipse 3.6.1), the computation of the look ahead illustrated in Figure 8 took 0.135 seconds, its visualization needed 1.1 seconds.[2] Even the computation of a look ahead with a depth of 12, which includes 67 states, took only 0.374 seconds. Without merging identical states, the same look ahead included 1272 states. While these unmerged states were computed in 3.22 seconds, it took the visualization code directly generated by EMF[3] and GMP[4] another 120 seconds to draw and layout the resulting graph-structure. Figures 6 and 7 show the linear

[2] For each state, ten story patterns were evaluated. All presented values are averages of ten repeated measurements.
[3] Eclipse Modeling Framework (http://eclipse.org/modeling/emf/)
[4] Graphical Modeling Project (http://eclipse.org/modeling/gmp/)

correlation between the amount of states and the overall runtime for look aheads. As can be seen, state merging effectively delays the exponential growth of states. Furthermore, Figure 7 compares the performance of a strategy maximizing the resource "money" with a random execution of available story patterns. The additional computations which are required to score and compare different alternatives when using a strategy, do not noticeably decrease the overall performance. Thus, concerning the runtime performance it can be stated that the implemented concepts of state merging and state scoring lead to acceptable execution times even for deeper look aheads.

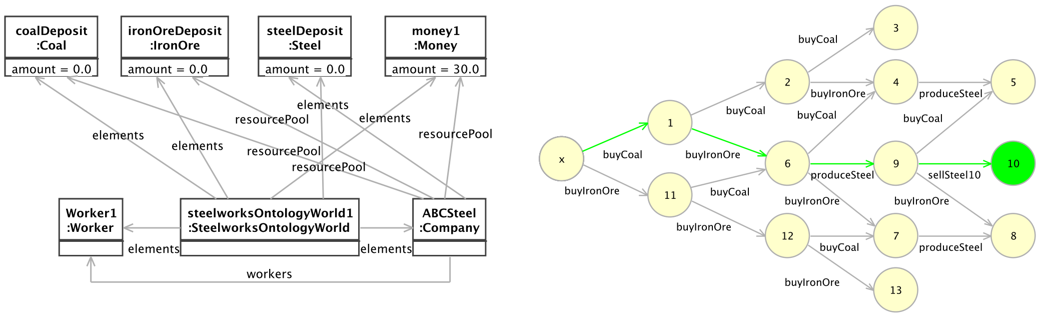


Figure 8: Starting from an initial state s_x (*left*), the look ahead can be computed (*right*)

4 Related Work

This section compares our approach to related ones, thereby focussing on approaches providing guided state space exploration or direct model checking for graph transformation systems.

In [HKMP02], Harel et al. extended their Play approach [HM03] with an analysis of which parts of the specification fit together before an actual simulation is started. Later, Harel et al. also added the ability to exclude specific situations [HKP04] during the simulation. While this allows a requirements engineer to ensure that only viable paths are offered, the possibility of expressing strategies to guide the simulation more effectively into specific situations is missing.

To *verify* whether specifications are suitable, the GUIDE approach [TS07] enables engineers to formally execute the specification. While one of them tries to arrive at an invalid state, another one executes only safe transitions. Either one has to pursue a specific strategy to arrive at a corresponding state. Consequently, while no stakeholder is involved, this approach is still suitable to explore and verify specific properties of the modeled system.

GROOVE [KR06] can explore complete state spaces, but our approach does so automatically by restricting the exploration to the most promising paths concerning a single goal (such as highest increase of a resource like money or smallest amount of transitions to reach a certain state) or a set of them within the look ahead. As a model checking tool, GROOVE can verify whether distinct properties are satisfied by the specification. GROOVE, however, does not provide a visualization for its graph transformations which is suitable to be used in stakeholder validations.

Henshin [BESW10] is another approach that is capable of creating a (partial) state space based on an initial state and a set of graph transformations. While it provides an interactive UI to explore the state space, it is not possible to express strategies to guide the exploration.

Edelkamp et al.'s approach [EJL06] reduces the state space exploration by abstracting from the original state space. Finding a path conforming to the elicited constraints in the abstracted state space ensures that such a path also exists in the original state space. Thus, heuristically, the exploration can be guided more efficiently by reducing the problem. However, since our approach does not deal with the complete state space but instead tries to elicit new inputs from stakeholders, their approach would only be feasible, if the generation of a suitable abstraction is possible at runtime to determine how the simulation should continue.

Baresi and Spoletini [BS06] approach of using *Alloy* to analyze graph transformation systems can be used to check properties of models and also to create look aheads of a specific depth. While their approach tries to answer questions that are similar to ours, their approach is not usable in an *interactive* way that can be integrated into or be a part of our visualization approach.

5 Conclusion and Future Work

In this paper, we extended our simulation approach of eliciting and validating scenarios between different, interacting stakeholders with simulation strategies and a bounded look ahead. While simulation and animation support stakeholders to understand behavioral specifications, requirements engineers had no explicit way of guiding the simulator effectively through these sessions. Now, requirements engineers can specify strategies for the simulator to ensure that the stakeholders validate viable scenarios first instead of randomly being led into dead ends due to the inherent incompleteness of the models during the requirements engineering stage. Based on our measurements, it is feasible to employ strategies for projects eliciting collaborative scenarios as they occur in workplaces, i.e. in the range of up to dozens of story patterns. For more sophisticated projects dealing with thousands and more graph transformations, the interactivity cannot be ensured.

Currently, only a limited set of functions can be used to specify how a state is scored. To express more sophisticated scoring algorithms, additional capabilities, such as a scoring function editor or a mini DSL to express the desired logic, will be implemented. Also, since the look ahead computation performs acceptable, the focus lies on increasing the efficiency of the UI.

Acknowledgements: We would like to thank the anonymous reviewers for their helpful comments. Also, we are grateful for Holger Giese's feedback on earlier versions of this paper.

Bibliography

[BESW10] E. Biermann, C. Ermel, J. Schmidt, A. Warning. Visual Modeling of Controlled EMF Model Transformation using HENSHIN. In *Proc. of the Fourth International Workshop on Graph-Based Tools (GraBaTs 2010)*. 2010.
http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/528

[BS06] L. Baresi, P. Spoletini. On the Use of Alloy to Analyze Graph Transformation Systems. In Corradini et al. (eds.), *Graph Transformations*. LNCS 4178, pp. 306–320.

Springer, 2006.
doi:10.1007/11841883_22

[EJL06] S. Edelkamp, S. Jabbar, A. Lafuente. Heuristic Search for the Analysis of Graph Transition Systems. In Corradini et al. (eds.), *Graph Transformations*. LNCS 4178, pp. 414–429. Springer, 2006.
doi:10.1007/11841883_29

[GGS09] G. Gabrysiak, H. Giese, A. Seibel. Interactive Visualization for Elicitation and Validation of Requirements with Scenario-Based Prototyping. In *Proc. of the 4th International Workshop on Requirements Engineering Visualization*. RE'09, pp. 41–45. IEEE Computer Society, Los Alamitos, CA, USA, 2009.
doi:10.1109/REV.2009.3

[GGS10] G. Gabrysiak, H. Giese, A. Seibel. Deriving Behavior of Multi-User Processes From Interactive Requirements Validation. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ASE'10, pp. 355–356. ACM, Antwerp, Belgium, September 2010.
doi:10.1145/1858996.1859073

[GHG12] G. Gabrysiak, R. Hebig, H. Giese. Simulation-Assisted Elicitation and Validation of Behavioral Specifications for Multiple Stakeholders. In *Proc. of the 21st IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. WETICE, pp. 220–225. Toulouse, France, June 25-27 2012.
doi:10.1109/WETICE.2012.17

[GHS09] H. Giese, S. Hildebrandt, A. Seibel. Improved Flexibility and Scalability by Interpreting Story Diagrams. In Magaria et al. (eds.), *Proc. of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009)*. 2009.
http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/268

[Hec06] R. Heckel. Graph Transformation in a Nutshell. *Electronic Notes in Theoretical Computer Science* 148(1):187 – 198, 2006. Proceedings of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques (FoVMT 2004).
doi:10.1016/j.entcs.2005.12.018

[HHV10] Ábel Hegedüs, Ákos Horváth, D. Varró. Towards Guided Trajectory Exploration of Graph Transformation Systems. In *Proc. of the 4th International Workshop on Petri Nets and Graph Transformation*. Volume ECEASST Vol. 40. Enschede, The Netherlands, Sept. 28 2010. Satellite Event of ICGT 2010.
http://journal.ub.tu-berlin.de/eceasst/article/view/583

[HKMP02] D. Harel, H. Kugler, R. Marelly, A. Pnueli. Smart Play-out of Behavioral Requirements. In *FMCAD '02: Proc. of the 4th International Conference on Formal Meth-*

ods in Computer-Aided Design. Pp. 378–398. Springer-Verlag, London, UK, 2002.
doi:10.1007/3-540-36126-X_23

[HKP04] D. Harel, H. Kugler, A. Pnueli. Smart Play-Out Extended: Time and Forbidden Elements. In *QSIC '04: Proceedings of the Quality Software, Fourth International Conference*. Pp. 2–10. IEEE Computer Society, Washington, DC, USA, 2004.
doi:10.1109/QSIC.2004.31

[HM03] D. Harel, R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[KNNZ00] H. J. Köhler, U. Nickel, J. Niere, A. Zündorf. Integrating UML diagrams for production control systems. In *Proc. of the 22nd International Conference on Software Engineering*. ICSE'00, pp. 241–251. ACM, New York, NY, USA, 2000.
doi:10.1145/337180.337207

[KR06] H. Kastenberg, A. Rensink. Model Checking Dynamic States in GROOVE. In *Model Checking Software*. Springer, 2006.
doi:10.1007/11691617_19

[TS07] J. Tenzer, P. Stevens. GUIDE: Games with UML for interactive design exploration. *Know.-Based Syst.* 20(7):652–670, 2007.
doi:10.1016/j.knosys.2007.05.005