Business Application Modeler: A Process Model Validation and Verification Tool

Sören Witt, Sven Feja, Andreas Speck and Christian Hadler
Christian-Albrechts-University of Kiel, 24118 Kiel, Germany, (swilsvfelaspelcha)@informatik.uni-kiel.de

***Abstract*—(Business) Process models are common artifacts in requirements engineering. The models can be enriched with plenty of (detailed) information and their at least semi formal character even enables model driven approaches or direct execution in workflow engines.**

Validity of process models is crucial. Manual checking is expensive and error-prone, especially for requirements that regard the content level (e.g. compliance). To enable automated checking, an adequate method for formal specification is necessary.

We present the Business Application Modeler (BAM), which is a modeling and Validation & Verification tool that integrates modeling of processes and formal graphical validation rules. These rules can be automatically applied to process models. In particular, the modeler is supported by visualizations of checking results directly in the process models. Next to highlighting mechanisms this support includes recommendations for the correction of errors.

I. Introduction

Most kinds of manual tests are comparatively time-consuming, cost-intensive, and a high probability is given that not all errors will be detected. Moreover, such manual testing does not scale to complex BPMs with many subsystems or hierarchies. A more promising approach is to check as much requirements as possible automatically with formal methods. However, for the application of formal techniques the intended audience has to be considered. Hence, adequate mechanism for the specification of formal requirements as well as for the visualization of checking results have to be provided.

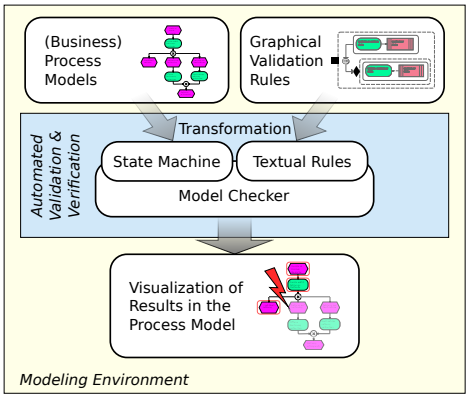

Fig. 1. Structure of BAM and the general validation process.

BAM [1] allows the graphical specification of explicit, formal requirements on the basis of (business) process models. The graphically specified requirements are used to automatically validate the models. Thereby, BAM enables the usage of automatable and thus repeatable techniques to check various kinds of graphical process models. Figure 1 depicts BAM's core areas. First, the *modeling environment* to model processes, graphical validation rules and visualize checking results. Second, the *Validation & Verification* which enables the automated checking whether the processes models comply to the rules. In this contribution we extended BAM with mechanisms for the visualization of results beyond the highlighting of violated parts of a process like in [1] or [2].

II. Business Application Modeler

Basically, BAM is a plug-in for the IDE Eclipse. The IDE offers various basic features including a very extensible plug-in mechanism. With respect to the modeling of processes, BAM comes with a basic model editor. It enables the user to define meta models for custom or existing process model notations on the basis of a generic meta meta model. For the demonstration of BAM, we utilize the *Event-driven Process Chain* (EPC) notation, which allows to enrich process models with detailed information. Basically, arbitrary properties for any model element can be specified with attributes.

A. The G-CTL notation for Graphical Validation Rules

BAM provides a rule editor that allows the graphical modeling of validation rules. The notation for these rules is the Graphical Computational Tree Logic (G-CTL) [3]. Currently, we focus on rules regarding the temporal behavior, defined in process models, although first steps with respect to structural properties have been made [4].

G-CTL allows to embed patterns of process elements (in solid rounded rectangles) into a graphical logic expression, which is a straight forward representation of the textual Computation Tree Logic (CTL). An example for a G-CTL rule is depicted in figure 2. The rule contains two patterns (in solid rectangles) and requires: If a state of the process matches the upper pattern, a state matching the lower pattern must eventually be reached. In [5] we present a detailed explanation of this and further G-CTL rules. The patterns of process elements define facts or circumstances like it is done in process models. Therefore, the rules are defined on the same level of abstraction as the process model. This makes G-CTL more comprehensible to the stakeholders of a development project than a textual CTL rule, which suffers from the indirect representation of the facts to test for.

Furthermore, G-CTL rules are specifiable in a generic manner, enabling the reusability for multiple process models.

978-1-4799-3032-6/14/$31.00 © 2014 IEEE

RE 2014, Karlskrona, Sweden

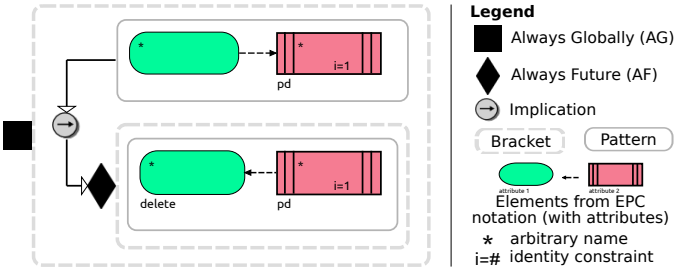

Fig. 2. A G-CTL example rule, requiring the deletion of personal data [5].

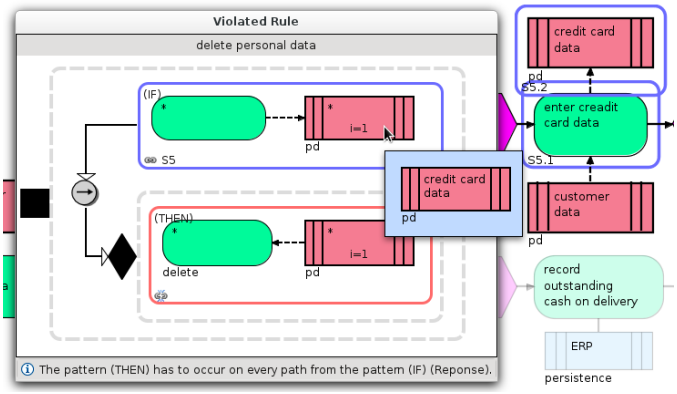

Fig. 3. Visual feedback utilizing the G-CTL rule. Elements matched by the upper pattern are highlighted. A recommendation is presented under the rule.

Instead of concrete element names wildcards ("*", c.f. fig. 2) can be used and identity constraints (e. g. "$i = 1$", c.f. fig. 2) facilitate to specify if elements across different patterns refer to the same or different elements in the process. [6] provides a formal description of the pattern matching mechanism.

B. Automatic Validation and Visualization of Results

Basically, BAM allows to adapt multiple checking techniques via a plug-in interface. Currently, we prefer the formal method model checking, since it provides immediate support for CTL. The general procedure of using a model checker as checking plug-in in BAM is shown in figure 1. The process model to be checked is transformed into a textually defined state machine and the G-CTL rules are transformed into appropriate textual CTL according to [6]. After calling the model checker, BAM examines the results in order to provide visual feedback as well as recommendations to the modeler.

In case of a violated rule the model checker delivers an error trace, i. e. the sequence of states the process has gone through. Additionally, BAM evaluates if and how the patterns in the rule are matched and recognizes characteristic types of rules. This information enables the following (combinable) methods to visualize errors and support the modeler:

1) Parts of the process that did not appear in the error trace are displayed transparently, which emphasizes the defective way through the process.

2) The modeler can move stepwise through the states of the error trace. All elements that were active in the particular state are highlighted (including elements on parallel paths).

3) To help understanding how a rule affects a process, the violated rule can be displayed beside the process model (c.f. fig. 3). BAM indicates if a pattern of the rule could be found in the process model and if so, the matched elements are marked in the process, and are shown by placing the mouse over the element in the pattern (box below the mouse cursor in fig. 3).

4) BAM recognizes certain types of rules according to [7] and gives recommendations how to solve the problem–yet without any interpretation of the CTL.

C. *Process Model Views* – MultiView

Additionally, a concept called *MultiView* is implemented in BAM. *MultiViews* allow the assignment of specific responsibilities in the process modeling workflow and provide mechanism to handle the increasing complexity of process models. The mechanism are based on configurable filters, which determine whether and how specific model elements are depicted and whether they are editable. Examples are referenced in [1].

III. Conclusion

BAM allows the specification of formal, graphical validation rules (G-CTL) on the level of process models. Rules can be specified in a reusable manner, allowing to automatically check the process model for these properties (requirements). The checking results are mapped into visual feedback and recommendations. BAM has been successfully applied to the processes and rules presented in [8] and [5] and to several more complex internal test cases and intra-corporate process models.

Our further research will focus on the enhancement of G-CTL towards domain specific purposes. Moreover, we intend to build reusable rule sets for various domains. Another research direction is the tighter integration of *MultiViews* with the validation mechanism.

References

[1] S. Feja, S. Witt, and A. Speck, "BAM: A Requirements Validation and Verification Framework for Business Process Models," in *11th Int. Conf. on Quality Software*. IEEE Computer Society, 2011, pp. 186–191.

[2] A. M. H. A. Awad and M. Weske, "Visualization of Compliance Violation in Business Process Models," in *Business Process Management Workshops 2009*. Springer, 2010, pp. 182–193.

[3] S. Feja and D. Fötsch, "Model Checking with Graphical Validation Rules," in *15th IEEE International Conference on the Engineering of Computer-Based Systems*. IEEE Computer Society, 2008, pp. 117–125.

[4] A. Speck, S. Witt, S. Feja, S. Feja, and E. Pulvermüller, "Integrating Validation Techniques for Process-based Models," in *Proceedings of the Eighth International Conference on Evaluation of Novel Approaches to Software Engineering*. SciTePress, 2013, pp. 246–253.

[5] S. Witt, S. Feja, A. Speck, and C. Prietz, "Integrated privacy modeling and validation for business process models," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. ACM, 2012, pp. 196–205.

[6] S. Witt, S. Feja, and A. Speck, "Applying Pattern-based Graphical Validation Rules to Business Process Models," in *IEEE International Conference on Software Testing, Verification, and Validation Workshops*. IEEE Computer Society, 2014, pp. 274–283.

[7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specifications for Finite-State Verification," in *Proceedings of the 21st Int. Conf. on Software Engineering*. ACM, 1999, pp. 411–420.

[8] T. Stuht, A. Speck, S. Feja, S. Witt, and E. Pulvermüller, "Rule Determination and Process Verification using Business Capabilities," in *Working Conf. on the Pract. of Enterprise Modelling*. Springer, 2012, pp. 46–60.