Integrating Validation Techniques for Process-based Models

Andreas Speck[1], Sören Witt[1], Sven Feja[1], Sören Feja[1] and Elke Pulvermüller[2]
[1]*Christian-Albrechts-University of Kiel, Institute of Computer Science, Kiel, Germany*
[2]*University of Osnabrueck, Department of Mathematics and Computer Science, Osnabrück, Germany*

Keywords: Integration, Validation Techniques, (Business) Process Models, Behavioral Properties, Structural Properties.

Abstract: Model checking has been established as an appropriate technology to validate behavioral properties of (business) process based systems. However, further validation technologies (e. g. for structural properties) may be of interest for process models. We propose a concept to integrate validation technologies in a unique system with a common user interface.

1 INTRODUCTION

Process models in general and business process models in particular are issue of validation concepts. Primarily, these concepts are used to ensure that the process models satisfy specific quality requirements (e. g. correctness). Approaches like (van der Aalst, 1999), (van Dongen et al., 2007) or the Business Application Modeler (BAM) (Pulvermüller et al., 2010; Feja et al., 2011) rely on model checking as verification technique. Model checkers are used to verify temporal sequences in the processes. Rules to be validated by model checkers are specifications in temporal logic, which is an extension of Boolean logic with additional operators, expressing temporal sequences.

By observing typical validation situations we learned that a certain number of validation requests do not consider temporal aspects (e.g. a specific sequence of states in a process) but affect the structure of the process models. This means that temporal operators are not required to express these requirements and specifications respectively.

The paper presents and discusses solutions to handle the different kinds of verification techniques for structural and behavioral properties. In our context structural properties concern the sequencing of the model elements (including its attached elements and attributes), "while behavioral properties consider their executional sequencing" (Deutch and Milo, 2007). In addition we especially emphasize the consideration of the functional/intentional meaning of the considered model elements/content. This means that requirements of specific use cases have to be validated. In contrast, many approaches often only check predefined requirements (e. g. syntactical rules) and do not consider user-defined rules.

Generally, it may be possible to verify behavioral as well as structural properties with model checkers. As the specification language of model checkers is based on an extension of Boolean logic, model checkers can handle structures which are based on Boolean logic. However, model checkers are optimized for checking behavioral (as in dynamic) properties in process models. Another alternative is simply to use different validation systems. This lets the user to decide which technique to use. A third possibility would be an integrated system providing both Boolean logic validation and model checking.

After the look at the base and related work in section 2 we give in section 3 a more elaborated examination of structural and behavioral properties and a discussion of the different validation solutions. Section 4 describes the design and architecture of our integrated validation system, illustrated with an example.

2 RELATED WORK

The validation of structural and behavioral properties of software and software models is a key problem in software development. Model checking has been applied very early ((Emerson and Clarke, 1980) or (McMillan, 1993)) as presented in the overview in (Bérard et al., 2001).

Business processes in particular have been in the focus of model checking based approaches. Some examples may be (Köhler et al., 2002), (Anderson et al., 2005) or (Speck, 2006). Further typical approaches are (van der Aalst, 1999) and (Pulvermüller,

Speck A., Witt S., Feja S., Feja S. and Pulvermueller E..
Integrating Validation Techniques for Process-based Models.
DOI: 10.5220/0004569202460253
In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE-2013), pages 246-253
ISBN: 978-989-8565-62-4

2002) which require transforming business process models to verification models (here, Petri nets and SMV Kripke structures).

Besides model checking as only validation technique other approaches like (Köhler et al., 2002) consider alternative checking technologies for business processes and compare these. Incompatible semantics of the business process models and the verification models are identified as one reason for different validation approaches (van Dongen et al., 2007). (Fahland et al., 2009) proposes problem-specific validation solutions.

Examples of further specific validation techniques are the Petri net based approaches based on bisimulation and algebraic solutions (Morimoto, 2008) which allow to compare two models instead of validate a model against explicit specifications. The semantic expressiveness of the specification languages may be increased (e.g. with μ-calculus (Bradfield and Stirling, 2001; Kozen, 1983) or in the multi-valued logic research as in (Chechik et al., 2003)).

A motivation for our work is (Gruhn and Laue, 2007) which presents the checking of specific patterns in process models by applying a Prolog based concept. This lead us to the question if it is possible to integrate the Prolog based approach with model checking approaches in order to increase the part of (automatically) verifiable specifications. In the design of electronic systems there are approaches to combine static and dynamic testing like (Bormann et al., 2005). A general proof of the applicability of this combination in software is presented in (May, 1998). (Xu, 2008) presents an approach of combining of model checking and theorem proving. A more recent approach is the integrated checking of UML class diagrams and state transition diagrams and their structural and behavioral properties (Weitl and Nakajima, 2011). A concept of combining different validation concept is the integration of structural checking with behavioral testing. This also increases the considered cases of validation and testing and may improve the rate of detected errors in systems.

A general requirement for us is the usability of the validation techniques by applying graphical interfaces. This is the background of the checking environment as we propose it in our paper. Similarly, (Förster et al., 2007) points out the importance of a graphical representation of the requirements and proposes an UML Activities based modeling approach for constraints for business processes. Other validation approaches which provide such graphical representations are for example BPMN-Q (Awad, 2007) and BPSL Modeler (Xu et al., 2008). Though of the aforementioned approaches for graphical requirements modeling only BPMN-Q integrates different verification techniques for structural and behavioral properties. However, while behavioral properties can be checked by explicitly defined graphical rules the rules to check structural properties are only derived implicitly from the behavioral rules. On the one hand this can reduce the effort for the modeler but on the other hand the modeler has to rely on the "proper" structural rules.

3 DISCUSSION

3.1 Structural vs. Behavioral Properties

By reasoning about the structure of a process model, we do not necessarily assert properties about the (execution) behavior of a process model. However, structural properties might define prerequisites for behavioral properties. For example, syntactical properties must be valid, to allow a reasonable semantic computation. Beside syntactical requirements, several requirements from various functional domains may have to be considered. For example: The existence of a process element of type X, with an attribute $p = q$, implies the existence of an element of type Y connected to an element of type Z by a certain edge type. Such requirements are not as generic as syntactical requirements. Typical ways to express structural properties are description logic (Baader et al., 2003) and other Boolean logic based specification languages.

Behavioral properties in general are visible after applying an execution semantic for the process model. The modeled system is observed in a temporal domain. Behavioral requirements therefore address temporal relations between entities in a process model, e.g. the order of execution or reachability. For example, a certain action must take place before another action may be executed. Such requirements are generally expressed by temporal logics like LTL or CTL.

Nevertheless, behavioral properties might consider structural properties at a certain point in time, i.e. in a certain state of the process. An example might be to check for the reachability of an action, that is connected to a certain data object. The connection between the action and the data object may characterized by a certain type and direction of an edge. Of course, this is a kind of structural property, but it is temporally invariant during the activity phase of the action, it is related to. Therefore, it makes sense, to consider such a structural property within an atomic expression in a behavioral property.

However, the evaluation of behavioral properties requires the interpretation of the process model with

respect to an execution semantic. Generally, this interpretation is guided by the control flow elements of the process model. We call the result of this interpretation the *behavioral model*, while we call the process model, as it is "painted", the *structural model*. The behavioral model does not necessarily preserve structural information any more. For example, control flow operators might not have an instance in such a behavioral model, only the execution behavior defined by the control flow is preserved.

Structural and behavioral properties can express different things and can be considered as complementary. For example, the behavioral CTL requirement $AF(p)$, requires p to become true eventually on every temporal path, but it does not require p to occur on every control flow path after an AND-split in the process model. Actually, there is no reason why the structural aspect of finding this property eventually on every control flow path, should be checked implicitly.

Another example is a CTL requirement like $AG(\text{action}_0 \rightarrow \neg EF(\text{action}_1))$. It requires that from a state in which action_0 occurs no state can be reached in which action_1 is active. This rule does not require the existence of a control flow path between these two actions. Hence, the occurrence of action_1 in a completely independent (in terms of a control flow connection) part of the process, might cause this rule to evaluate to false. It depends on the actual functional background, if this behavior is adequate. An additional structural requirement, that checks for the existence of a control flow path between these actions, would avoid the false positive result. However, for both variants reasonable use cases exist.

In this contribution, we propose the integration of structural and behavioral properties and their corresponding checking techniques respectively. On the one hand in order to provide a consistent UI for the specification. On the other hand we expect, that it allows more efficient verification procedures.

3.2 Integration Approaches

We consider three alternatives for supporting the validation of the aforementioned properties of process models (structural relations/hierarchies and dynamic behavior). Key issues that have to be considered are:

- Ability to specify a wide range of requirements.
- Effort for implementing these solutions as well as extensibility, e.g. for new kinds of specification or new checking techniques.
- Degree of integration from the users perspective.
- Compatibility of the validation results.

3.2.1 Singular Validation Technique

Here a single checking technique is used to check the process models against the requirements. For a example first order logic actually allows reasoning about behavioral properties as well (Bérard et al., 2001). Or a model checker may be used to check structural properties, e.g. by checking an expression like $AG(p)$, where p is a Boolean statement, which must be true in every state of the process (AG operator).

A singular validation technique needs to be quite generic in order to allow the checking of a wide range of requirements. The more generic the technique is, the less abstraction is offered. This may raise the complexity of mapping process model and requirement specifications, resulting in high implementation effort. The mapping procedure is even responsible for providing an acceptable efficiency. Results of a singular validation technique cannot be incompatible. Nevertheless, a complex mapping towards the checking tool, may induce a complex mapping of results into an appropriate visualization in the process model.

Changing the validation technique, may have a strong impact on the mapping for all kinds of specifications, making it less extensible.

Such a generic technique and a powerful mapping may allow various levels of abstraction for requirement specification, presented to the user. The procedure of integration is (ideally) not visible to the user.

3.2.2 Multiple Dedicated Checking Systems

This is for sure the simplest solution. It is up to the user to chose the verification technique, to formulate the specification in the correct syntax and semantics and to interpret the results. In a certain case this is the most open solution, since the modeling system acts as an editor, providing to model data via interfaces to checking systems. In contrast to the first approach, this simple solution leaves the effort to integrate the models, specifications, checking technique and result presentation to the user, it actually provides no tool-based support regarding the integration.

3.2.3 Unified Validation System

In case of a more elaborated validation system like BAM it may be of interest to have an *integrated validation*. This means that two or more validation technologies are integrated into one (modeling and) validation system. Such a system allows the user to specify structural as well as behavioral properties in a similar manner. Then the validation system chooses automatically the corresponding transformation algorithm and checking tool, executes the checking and returns

the result. In the simplest case, the selection of the *right* validation technique is based on the operators which are used in the specified rule. But there is a variety of criteria that could be considered in the selection process (e. g. strictness or performance issues).

Technically, the integration of the different validation techniques can be realized with a layered structure of the validation system: modeling and specification layer, transformation layer and validation layer. We just have to introduce a further validation technology in the validation layer and we need to adapt the transformation in order to connect the further validation technology.

An advantage of this solution is that different validation technologies are provided in one tool for the specification of processes and rules. If the user does have knowledge about validation techniques he may choose the desired technique explicitly or the validation system may automatically chose the appropriate validation technology.

This third solution has the problem of extra effort to integrate the validation technologies. This requires also the realization of specific transformation concepts. For example, a transformation into the structural and behavioral model as well as the requirements transformation into the appropriate representation for the checking tool. Furthermore, it may be possible to combine different specification languages in one rule modeling language. This combination requires a special consideration in the transformation and the execution of the checking tool respectively. Besides the mentioned higher effort for this solution the extended transformation allows the definition of a common interface for the validation concerning the usage of the system and concerning the validation result. In general, an integrated validation concept looks promising. Being able to choose the appropriate checking technology would make the validation more efficient than the first solution but it bears the option to provide different validation technologies to the user.

Solution 2 is quite easy to realize from the modeling tool perspective, but does not provide any tool-based support for the integrated validation, leave the effort to use different validation techniques to the user. It requires the understanding of different modeling and specification languages. For solution 1 the effort and capabilities depend on how generic the validation technique is. Although it might increase consistency in the specification, we consider the mapping effort, which actually realizes integration, as too high. Also changing or extending the validation technique induces high effort.

The third solution can be seen as an integrated validation system which may be enhanced to further checking technologies and may provide a unique user interface. The drawback of solution 3 is the higher effort to realize the integration of the validation technologies (including the translation and transformation mechanisms).

4 DESIGN OF AN INTEGRATED VALIDATION

The realization of the concept we propose is based on the validation system BAM (Business Application Modeler). BAM has a layered design with the three layers: business process and specification (rule) modeling layer on top. An intermediate transformation layer and the verification tool layer at bottom.

4.1 Rule Modelling

BAM provides a graphical interface for modeling rules (specifications) very close to the graphical notation of the process models, called *graphical validation rules*. Basically, the meta model of these rules consist of the graphically represented set of operators of a certain specification language (e.g. Boolean Logic or Computation Tree Logic) and a subset of the meta model of the process modeling notation of models to be checked. Rules are specified by connecting patterns of process elements and operators of the specification language. The rule modeler does not have to know anything about the representation of the process model in the checking tool.

An example for such a graphical validation rule notation is the Graphical Computation Tree Logic (G-CTL), first introduced in (Feja and Fötsch, 2008). G-CTL provides a graphical representation of all operators of the textual specification language CTL, i.e. **F**uture, **G**lobally, ne**X**t or **U**ntil which are combined with path quantifiers **E**xist and **A**lways. The atomic expressions of G-CTL rules are specified by patterns of process elements. These patterns are atomic in terms of time, i.e. they can exist in a step in the discrete model of time in CTL. They are not atomic in terms of the process modeling notation. G-CTL has different appearances for different process modeling notations. E.G. for the Event-driven Process Chains (EPC), we call G-CTL correspondingly EPC-G-CTL.

Hence, the modeling component of BAM can easily be extended by further specification languages (e.g. Boolean Logic, CTL* or LTL) for different process modeling notations.

Figure 1 depicts an example for an EPC-G-CTL specification. Besides the temporal operators with path quantifiers, Boolean operators are contained as

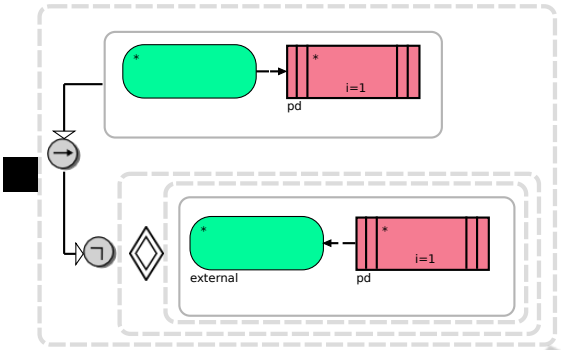

Figure 1: EPC-G-CTL rule: Personal data must not be accessed externally.

well. Dashed boxes can be understood as brackets. Solid boxes contain the patterns of process elements.

The example in figure 1 is a rule for handling data privacy, further examples are described in (Witt et al., 2012). Prosaically the rule can be read as follows: Once personal data are created by a function, they must not be read by a function with the attribute external. Reading the rule in terms of the operators means: In any state of the process (always globally) holds: If (implication) a function occurs, creating personal data (indicated by the attribute *pd*) then from this state on there must not (not operator) exist any course of action which eventually (Exists in Future) leads to a state, where a function with the attribute *external* reads the data, created before. The request for the identity of these data is specified by the identity constraint $i = 1$ in the data cluster.

Although this is a valid (EPC-G-)CTL rule it may be sufficient for most processes to express this rule without the overhead of evaluating temporal operators. This is because the order of these patterns of process elements, does not matter regarding the prohibition of using personal data externally. (The requirement that data should be created before they are used could be expressed more generally by a further behavioral rule.)

We propose the solution to use another validation technique that allows to validate against structural properties, expressed in Boolean logic. Such a graphical validation rule is depicted in figure 2. The logic is similar to the logic of the G-CTL rule in figure 1, but temporal operators have been omitted. The syntax is basically the same. In this rule the existence of a function, creating personal data, implies the nonexistence of an "external function", using these data.

4.2 Checking Techniques Integration

Above, we already introduced the layers of our integration architecture, which is outlined in figure 3.

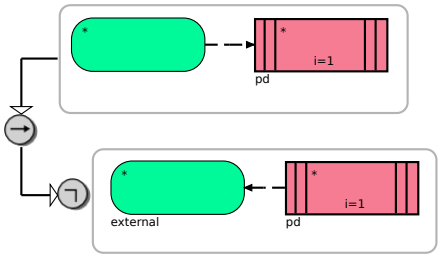

Figure 2: Graphical structural rule: collected personal data implies that it is not accessed externally.

The modeling layer basically provides three features: Modeling of processes, modeling of graphical validation rules and generic mechanisms to visualize the checking results inside the process models. The transformation layer is triggered by passing process models and rules to it. We will describe this layer more detailed in the following section. The validation layer is responsible for executing the checking techniques and returns the checking results to the transformation layer, which finally uses the visualization capabilities of the modeling layer to visualize results.

The transformation layer is the core component of the proposed integration architecture and will be described in the following. We are going to illustrate this with an example process, shown in figure 4 and the example rules in figure 1 and 2. The example process shows a cut-out of a payment process.

4.2.1 Choosing the Evaluation Path

The *basic rule processing* starts with analyzing incoming rules. Here the transformation layer decides on basis of the operators in the rule, which checking technique to use. Roughly we distinguish between structural and behavioral evaluation path. For the behavioral path the model checker Cadence SMV (McMillan and Cadence Berkeley Labs,) is currently adapted. For structural evaluation, SWI Prolog is adapted. Both paths also allow a *direct evaluation* for "simple" cases.

The basic decision between both evaluation paths is quite simple: The behavioral path will be taken, if the rule contains temporal operators as in figure 1. In case of purely Boolean operators (figure 2), the structural evaluation path is chosen.

4.2.2 Behavioral Evaluation Path

Still in the step of basic rule processing, the graphical rules are preprocessed by identifying the places in the process model, which are matched by the patterns of process elements in the rule. We call such places substates, which are atomic parts of the process model in

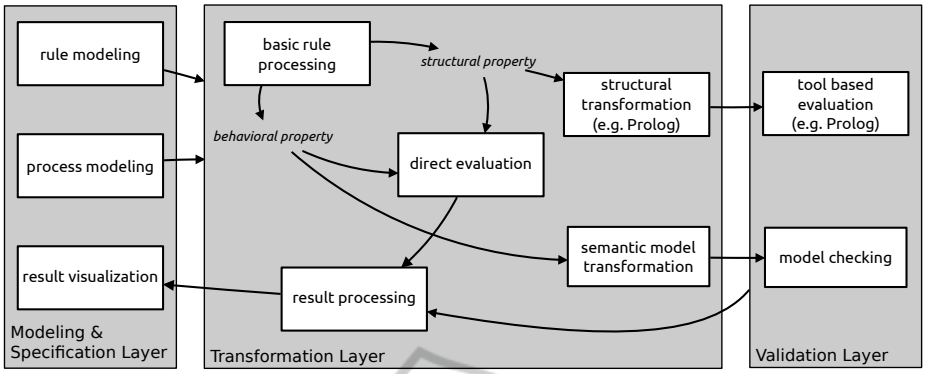

Figure 3: Outline of the checking technique integration in BAM.

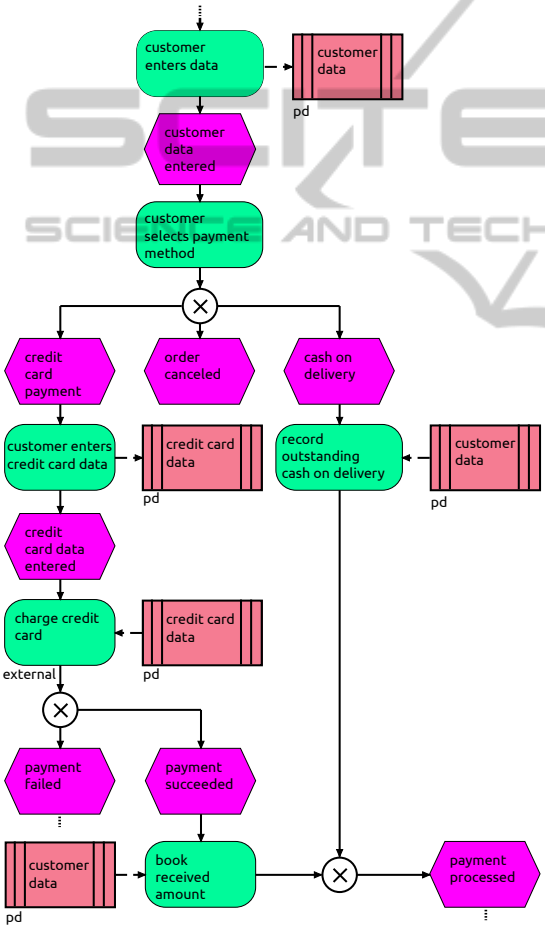

Figure 4: EPC business process model sample.

terms of time consumption. E. g. for EPC a function and all objects attached by non-control flow edges (clusters, systems etc.) form a sub-state. Sub-states are connected via the control flow and can be concurrently active, therefore, the patterns in G-CTL may not contain control flow elements.

The rule in figure 1 contains two patterns. For each pattern a (possibly empty) set of matching sub-states can be identified. Regarding the example process, the first pattern of the rule matches the sub-state around the function `customer enters data` (let this be sub-state s_a) and `customer enters credit card data` (sub-state s_b), both connected to a data cluster with the attribute *pd* via outgoing data flow. For the second pattern only the function `charge credit card` and the corresponding cluster matches (sub-state s_c).

Due to the identity constraints ($i = 1$) for the data cluster, not every assignment of matching sub-states to the patterns is consistent. Therefore, consistent assignments of substates to the patterns are determined and will be evaluated in a dedicated instance of the rule. Regarding the example rule, the clusters need to be named identically, which is only the case for s_b and s_c. This is the first instance of the rule to be evaluated. For assigning s_a to the first (upper) pattern, the second pattern is constantly set to *false*, since no matching sub-state exists. This is the second instance of the rule to be evaluated.

Next, for each rule instance will be checked, if it can be evaluated directly. Currently, direct evaluation is performed for trivial cases like the second instance, where the result can be determined without actually observing the process behavior. Because the second pattern is set to false, the EF operator evaluates to false. This is inverted to true by the negation and therefore the implication is always true, which always satisfies the AG operator. This direct evaluation is performed by simply using a parse tree of the rule.

The evaluation of the first instance is not as trivial. Here, model checking will be used. Therefore the semantic model transformation will be performed. In this step the process model is transformed into a representation, readable as input for the Cadence SMV. This representation describes the dynamic behavior of the process, i. e. all possible sequences of sub-state activity. Hence, an execution semantic needs to be applied in this step. The rule instances to be checked by

the model checker are transformed straight forward, where the patterns are replaced by tests for the activity of the assigned sub-states.

Running the model checker leads to a counter example for each rule instance, violated by the process model. A counter examples provides a sequence of the involved sub-states, causing a rule violation, which can be utilized to visualize the error. Here, the second rule is violated by the example process, since a state exists where s_b is active and from where a state can be reached in which s_c is active.

4.2.3 Structural Evaluation Path

The structural evaluation path currently also chooses between a direct evaluation and the utilization of a checking tool like Prolog. For now the criteria applied for this decision is the "complexity" of the patterns. In contrast to model checking, the pattern matching and evaluation of identity constraints can be implemented quite elegant in Prolog. Moreover, the notion of sub-states as in the behavioral evaluation has not the same relevance for structural observations. Here, the patterns may also contain control flow elements.

For simple patterns also a direct evaluation is possible, similar to the direct evaluation in the behavioral path. Here again, the rule instances are created in the same way, for the rule in figure 2, which has the same pattern and therefore the same consistent assignments. If an assignment for a pattern is found, it is set to true, otherwise to false. Hence, in case of the first instance, both patterns are set to true and the rule evaluates to false. For the second instance the first (upper) pattern is true, the second false and the rule is true.

For more sophisticated rules, a structural transformation into Prolog is implemented, which will not be described in detail. However, the graph structure of the process model is transformed into the knowledge-base of Prolog, including node and edge types. Rules are transformed into Prolog requests. Although this procedure is similar to the model checker based evaluation, it is not as expensive, since no execution semantic needs to be applied.

4.2.4 Result Processing and Visualization

Once results are returned, available information like error traces or matched patterns are used to visualize the error in the process model. Figure 5 depicts the visualization for the violation of the rule in figure 2, determined by the structural evaluation path. Currently, critical elements of the process model are highlighted.

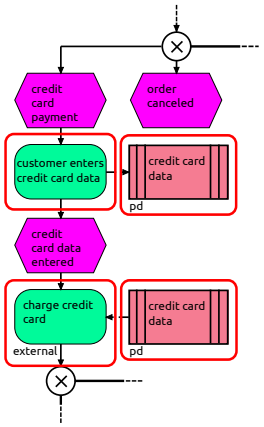

Figure 5: Error visualization in process model sample.

5 CONCLUSIONS

This paper presents concepts to integrate different validation techniques in one easy to use system, based on the Business Application Modeler. The validation techniques are integrated in or by the transformation layer between the modeling and specification layer and the validation layer. In our work we focus on the validation of business process models. The approach hides the complexity of transformation and choice of checking techniques from the users and provides a unified user interface. This enables checking of sophisticated requirements on a functional level for users, that are not necessarily familiar with the formal syntax and semantics of different elaborated checking techniques.

Current work in progress is the conjunction of structural and behavioral requirement specifications. Such a combined language could deliver an unique specification mechanism and would be more expressive at the same time. To achieve this, the notation for graphical validation rules needs to be extended and the integration mechanisms must be able to manage the evaluation of such rules efficiently. Moreover we intend to extend the selection mechanism of the *appropriate* validation technique.

A further problem, which may be addressed by the validation of business processes, may be constraint satisfaction. Constraints may be useful when trade-offs between different business process solutions have to be made. Bi-simulations would also be of interest. With bi-simulations different models may be compared. For instance this may help to validate the semantically identity of processes on different levels of abstraction.

REFERENCES

Anderson, B. B., Hansen, J. V., Lowry, P. B., and Summers, S. L. (2005). Model checking for design and assurance of e-Business processes. *Decision Support Systems*, 39(3):333–344.

Awad, A. M. H. A. (2007). BPMN-Q: A Language to Query Business Processes. In Reichert, M., Strecker, S., and Turowski, K., editors, *Proceedings of the 2nd Int'l Workshop Enterprise Modelling and Information Systems Architectures Concepts and Applications*, pages 115–128, Bonn. Gesellschaft für Informatik.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA.

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2001). *Systems and Software Verification – Model-Checking Techniques and Tools*. Springer, Berlin, Germany.

Bormann, J., Fedeli, A., Frank, R., and Winkelmann, K. (2005). Combined Static and Dynamic Verification, Version 2, Public Version March 31st, 2005, (Deliverable 3.1/1), Research Report. Technical report, project PROSYD, European IST.

Bradfield, J. and Stirling, C. (2001). Modal logics and mu-calculi: an introduction. In *Handbook of Process Algebra*, pages 293–33. Elsevier Science Publishers.

Chechik, M., Devereux, B., Easterbrook, S., and Gurfinkel, A. (2003). Multi-Valued Symbolic Model-Checking. *ACM Transactions on Software Engineering Methodology*, 12(4):371–408.

Deutch, D. and Milo, T. (2007). Querying Structural and Behavioral Properties of Business Processes. In *Proceedings of the 11th international conference on Database programming languages*, pages 169–185, Berlin / Heidelberg. Springer Verlag.

Emerson, E. A. and Clarke, E. M. (1980). Characterizing Correctness Properties of Parallel Programs Using Fixpoints. In *ICALP 1980, Automata, Languages and Programming, 7th Colloquium*, pages 169–181. Springer LNCS 85.

Fahland, D., Favre, C., Jobstmann, B., Köhler, J., Lohmann, N., Völzer, H., and Wolf, K. (2009). Instantaneous Soundness Checking of Industrial Business Process Models. In *Proceedings of the 7th International Conference on Business Process Management (BPM 2009)*, pages 278–293. Springer, LNSC 5701.

Feja, S. and Fötsch, D. (2008). Model Checking with Graphical Validation Rules. In *Proceedings of the 15th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2008)*, pages 117–125. IEEE.

Feja, S., Witt, S., and Speck, A. (2011). BAM: A Requirements Validation and Verification Framework for Business Process Models. In *11th International Conference On Quality Software*, pages 186–191, Los Alamitos, CA, USA. IEEE Computer Society.

Förster, A., Engels, G., Schattkowsky, T., and Van Der Straeten, R. (2007). Verification of Business Process Quality Constraints Based on Visual Process Patterns. In *Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE '07)*, pages 197–208.

Gruhn, V. and Laue, R. (2007). Checking Properties of Business Process Models with Logic Programming. In *5th International Workshop on Modeling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS'07)*, pages 84–93. INSTICC PRESS.

Köhler, J., Tirenni, G., and Kumaran, S. (2002). From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In *6th International Enterprise Distributed Object Computing Conference (EDOC 2002)*, pages 96–106.

Kozen, D. (1983). Results on the propositional mu-calculus. *Theoretical Computer Science*, 3(27):333–354.

May, W. (1998). *Integrated Static and Dynamic Modeling of Processes*. PhD thesis, Universität Freiburg, Germany.

McMillan, K. and Cadence Berkeley Labs. The cadence smv model checker.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers.

Morimoto, S. (2008). A Survey of Formal Verification for Business Process Modeling. In *8th International Conference on Computational Science*, pages 514–522. Springer.

Pulvermüller, E. (2002). Composition and Correctness. In *SC 2002: Workshop on Software Composition*, volume 65 of *Electronic Notes in Theoretical Computer Science (ENTCS)*. Elsevier Science Publishers.

Pulvermüller, E., Feja, S., and Speck, A. (2010). Developer-friendly verification of process-based systems. *Knowledge-based Systems*, 23(7):667–676.

Speck, A. (2006). Modelling and Verifying of e-Commerce Systems. In *Proceedings of International Workshop on Regulations Modelling and their Validation and Verification (REMO2V'06) in conjunction with CAiSE'06*, pages 857–863.

van der Aalst, W. M. P. (1999). Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650.

van Dongen, B. F., Jansen-Vullers, M. H., Verbeek, H. M. W., and van der Aalst, W. M. P. (2007). Verification of the SAP reference models using EPC reduction, state-space analysis, and invariants. *Computers in Industry*, 58(6):578–601.

Weitl, F. and Nakajima, S. (2011). Integrated model checking of static structure and dynamic behavior using temporal description logics. *Electronic Communications of the ECEASST*, 46:1–16.

Witt, S., Feja, S., Speck, A., and Prietz, C. (2012). Integrated privacy modeling and validation for business process models. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, pages 196–205, New York, NY, USA. ACM.

Xu, K., Liu, Y., and Wu, C. (2008). BPSL Modeler – Visual Notation Language for Intuitive Business Property Reasoning. *Electronic Notes in Theoretical Computer Science*, 211:211–220.

Xu, Z. (2008). Combination of Model Checking and Theorem Proving to Develop and Verify Embedded Software. *Information Technology Journal*, 7(4):623–630.