# Business Process Verification

Sven Feja, Andreas Speck
Department of Computer Science
Christian-Albrechts-University Kiel
Olshausenstrasse 40, 24098 Kiel, Germany
svfe@informatik.uni-kiel.de, Andreas.Speck@email.uni-kiel.de

Elke Pulvermüller
Department of Mathematics and Computer Science
University of Osnabrück
Albrechtstr. 28, 49076 Osnabrück, Germany
elke.pulvermueller@informatik.uni-osnabrueck.de

**Abstract:** Models of commercial systems reflect either the statical structure or the dynamic behavior of a system. The dynamic aspects are the business processes and their models.

Whereas the static relations in a system may be expressed by Boolean logic, the dynamic activities and their temporal sequences ask for a better formalism, e.g. temporal logic. Temporal logic is based on Boolean logic extended by operators expressing the temporal order of states. In general there are different technologies to verify temporal sequences. Our choice is the model checking concept.

In the paper we present examples of business process models and how these models may be checked. We introduce a model to specify the rules (rules model) and demonstrate how the results of the checks can be displayed in the business process models. These models and the rules are represented in a graphical editor. Both models are transformed into a formal language which may be processed by a verification tool - a model checker in our case. The results are then visualized in the graphical editor indicating where the model violates or keeps the rules.

## 1 Introduction

Workflows and processes are fundamental in computer-based systems. Real-time systems as well as large scale business systems, for instance, focus on process execution. All theses systems have in common that their dynamic behavior is the essential element. Even a simple purchase system has to realize a selling process.

There exist different modeling concepts for the different types of systems, e.g. SDL or the modeling language Z for time-critical systems or business process models for commercial systems.

In the paper we focus on business process model types which are typical for commercial systems: Event-Process Chains (EPCs) [Sch98] which are part of the modeling concept ARIS (**Ar**chitecture of integrated **I**nformation **S**ystems). This modeling concept has first

being brought up to model large scale ERP systems like SAP R/4. Now EPCs are used to model almost all kinds of commercial systems. Instead of deriving new (formal) models from these for verification purposes, we propose to integrate the means which are necessary for a verification into the existing modeling environment. The goal is to provide an easy-to-understand checking solution for the domain engineer who is usually not a formal methods expert.

As a real application on which we demonstrate our approach we chose the high-performance e-commerce system Intershop Enfinity (Intershop Communications AG). This e-commerce system is used in large scale systems of retailers (e.g. Otto GmbH & Co KG or Quelle GmbH), in the automotive branch (e.g. Volkswagen AG, MAN AG, BMW AG) or in e-procurement systems (e.g. run by the German Federal Ministry of the Interior or governments of other countries or large companies). Taking e-commerce systems as an example domain has the advantage, that a large number of users are familiar with their basic functionalities, and from a business view perspective e-commerce systems may be considered as an extension of ERP systems with similar complexity.

The remainder of the introduction provides an overview over the typical steps and models in the e-commerce requirements modeling. Section 2 outlines related work in the domain of requirements verification. Section 3 describes our Temporal Logics Visualization Framework with its elements (in particular its visual temporal logic language and its visual error representation) followed by a demonstrating example in section 4 and the conclusion.

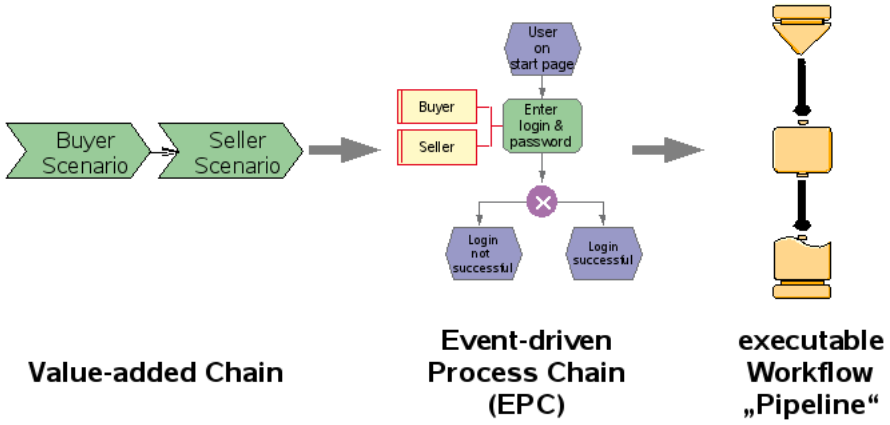


Figure 1: Model Types in e-Commerce Systems (ARIS4Enfinity).

### 1.1 ARIS Modeling Approach for e-Commerce Systems

The ARIS approach supports in general a wide range of applications. Therefore, in this paper the generic ARIS concept is limited to a specific instance which is more suitable for e-commerce systems: ARIS4Enfinity.

Figure 1 depicts models in the ARIS4Enfinity modeling approach. In the development of an Enfinity-based e-commerce system the requirements definition starts with a very abstract overview of the basic scenarios for the functionalities of the system. The model type used is mainly the value added chain which may capture rough temporal dependencies. The value added chain models are used as an early description since they are easy to understand for the customers as well as the developers. However, due to the value added chain models are kept simple and abstract there is usually little need for an automated verification.

The EPCs are used to model the business process requirements in detail (c.f. model elements description in section 1.2). The EPC models are ideal for the communication between the domain experts (economists) and the computer scientists, since they are still understood by both groups.

The EPC models serve as base for the design of the implementation. In the case of Intershop Enfinity, executable workflow models (called *Pipelines*) represent the design and are executed by the system's application server.

If the domain experts want to check the business processes of an e-commerce system, this is generally done on the level of EPC models. Therefore, business rules, regulations and system specific requirements which have to be implemented by the system are to be both, expressed and verified on this business process (EPC) level. If the EPC models do not represent the requirements correctly then the resulting system will hardly meet the needs. Therefore it is essential to verify the requirement description represented by the EPC models.

### 1.2 EPC Model Elements

As mentioned above the EPC model is part of the modeling concept ARIS as well as the specific profile for the development of Enfinity systems *ARIS4Enfinity* [Bre02]. The basic elements of an EPC model are shown in figure 1 (in the middle): The control flow is symbolized by a sequence of events (magenta/violet hexagons) and functions (green rectangles with rounded edges) which are connected by arrows representing the control flow. Branches in the control flow are defined by the Boolean logic operators: AND, OR and XOR. In the figure an XOR is depicted (the circle with a cross in the middle). AND requires that all paths of the branch are active. OR indicates that at least one path is used. XOR allows that one and only one path is chosen. We ignore further elements like organizational elements (depicted in figure 1 for *Buyer* or *Seller*).

## 2 Related Work

In our work we propose the integration of means to verify requirements directly into the development and requirements models. This way, the gap between domain knowledge and formal methods expertise may be reduced. All parts are on the same abstraction

level. Related work may be found in the field of requirements specification. A comparison of requirements elicitation and specification/definition methodologies can be found in [PO98],[EK04]. According to [EK04], the use of formal and automatic methods is "the least ambiguous requirements representation allowing for automatic verification techniques". A push-button formal technique to verify the fulfillment of automata-based specifications is model checking [CGP01]. It uses (temporal) logics as specification language and checks these (temporal) requirements against a model. As opposed to other formal approaches (e.g. theorem provers) it is more restricted in what can be verified leading, on the other hand, to the advantage of a developer-friendly extensive automatization.

The usage of model checking requires the input of formal specification formulas/rules and finite state machine models. Therefore, any input needs to be transformed to such a formal format. The core idea of our approach is similar to the idea of Model-Driven Software Development [VS06]. High-level platform independent models are successively transformed to lower-level platform dependent models. While there exist other approaches to realize such transformations from higher abstraction levels to formal models for the purpose of verification [DP02], [SPJF02] we, in particular, propose to raise the abstraction level of the rules and error representation in addition. The rules the requirements models have to fulfill are expressed on the same (higher, visual) abstraction level as the business process models. The path to raise the abstraction level of models, rules and error representation may also be found in [CDH$^+$00] and [HD01], for instance. However, in this work the abstraction level is still on the programming language level while we aim at even higher business process model levels. Beyond verification, the domain of model-driven testing may be considered as related to our work (e.g. [BDG$^+$07]).

Other interesting approaches to raise the abstraction level of the rules may be found in mapping natural language to formal descriptions as well as in visualizing requirements and rule models. For instance, [CDHR02] provides temporal logic patterns to ease the mapping of natural language to temporal formulas. The visualization of requirements is well known in hardware related development domains and gets increasing attention in the software development as well. For instance, a visualization of requirements in UML models may be found in [KGLC06, Kon06]. The components used are SPIDER [KC05], Hydra [MC01] and Theseus [GCKK06a]. The main principle is to derive a UML model from requirements which are specified in natural language. However, UML is not generally accepted by business process engineers. Therefore, the usage of business process models like EPC or BPMN [OMG06] and visual requirements specification for these is needed.

Besides a user-friendly representation of the models together with their corresponding verification tasks it is necessary to process such extended models. As opposed to our top-down approach related work may be found in [Pul09], for instance. There, formal models are extended in a bottom-up manner to increase the expressiveness of low-level formal models and, thus, to decrease the gap between developer models and verification models.

# 3 Business Process Modeling

In our terms, modeling of requirements means the usage of a (temporal) logic to specify requirements graphically. This visualization allows to formulate the requirements for processes on the same level of abstraction as the models. Given specification properties which are typically difficult to understand and available in a textual format, we have to deal with the challenge to provide corresponding graphical models for them. We solve this task by means of the Temporal Logics Visualization Framework (TLVF). Section 3.1 gives a short overview of the Framework. More details of the TLVF may be found in [FF08].

Section 3.2 provides our approach of interpreting validation errors. Additionally, the development of adaptation advices for some major types of specifications is explained. Finally, section 3.2 explains the visualization of validation errors and adaptation advice in the original process model.

## 3.1 Temporal Logics Visualization Framework – TLVF

The purpose of TLVF is to deliver all required means for the visualization of temporal logics in union with a process model or workflow. As shown in figure 2 the framework is divided in three layers. The first aggregates the logics (the bottom layer). The second defines the graphical symbols for each operator of a logic and the third layer provides the process model, the graphical rules definition and the required transformation tasks.

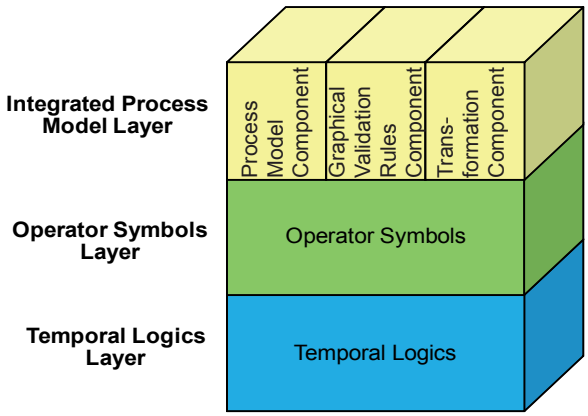


Figure 2: Temporal Logics Visualization Framework.

With these layers is it possible to state graphical rules of any logics (e. g. CTL and LTL) in union with desired process models (e. g. EPC or BPM). Moreover, the transformation of the process model and the graphical rules to an appropriate format for different model checkers is provided.

A graphical logic is defined by its *operator symbols* and a so called *placeholder*. The operator symbols are the corresponding graphical representation of the textual operators (e. g. quantifiers and Boolean operators). For a rule definition with these symbols the connection to a process model is needed. Therefore, the placeholder can be filled with an appropriate process model element. The general definition of **G**raphical **CTL** (G-CTL) is depicted in figure 3.

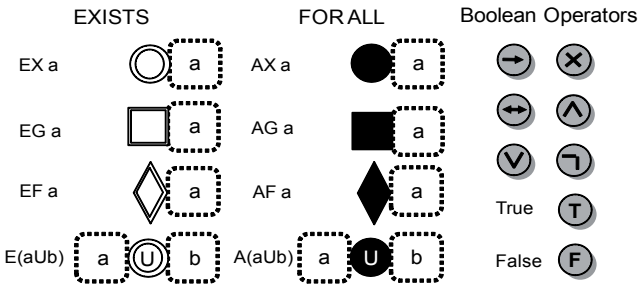


Figure 3: Symbols of G-CTL.

### 3.2 Validation of Business Process Models

The business process models (EPC models) are checked against specifications models (G-CTL). If the checking result is positive then there is no need for any further activity. When an error is detected we must display this error in a way the users can easily understand. Although the checking technology we apply – model checking – is very powerful, in case of an error a model checker simply presents a counter example [CGMZ95] (and in case that more than one specification is checked the specification violated is presented as well).

The information that a specific rule is not satisfied determines the process paths which need to be examined. One of these paths is the counter example a current model checker can deliver. However, as mentioned above, a model checker typically only delivers one counter example. Our approach uses this counter example to visualize it in the process model, as described in the following subsections.

The visualization of violated rules in a process model can be achieved on different ways. One would be the visualization of all witness scenarios in the terms of the original model. This was done for UML models in [KGLC06]. A second way is the presentation of the counter example in terms of the original model as in [GCKK06b] where it is achieved by generating a sequence diagram of the problem scenario.

In contrast to [KGLC06] and [GCKK06b] our approach presents the results of the validation in the original process model. Additionally, it is possible to offer direct advices for error correction in case of a validation error. This is achieved by an automatic interpretation and visualization of the validation error. The following sections 3.2.1 and 3.2.2

explain how the interpretation and visualization of validation errors can be accomplished.

### 3.2.1 Interpretation of Validation Errors

The interpretation is based on the validation errors which in our case are delivered by a model checker. In case of a validation error a model checker gives a counter example which describes a state of error of the model.

To provide adaptation advices for process models in case of validation errors the interpretation of the rule causing the error is needed. A complete interpretation of validation errors would require to regard all possible specification rules. An approach we use is to focus on often used specifications. [DAC98] has proposed so called specification patterns. These patterns are derived from the analysis of property specifications [1]. A full list of the property specification may be found in [DAC09].

The three important specification patterns we identified for the validation of business process models are: the **Response**, **Universality** and **Absence** pattern. The survey of [DAC98] shows that 80% of the regarded specification properties are stated according to these three patterns. For this reason, we have decided to describe our approach of visualizing violations delivered by a model checker with these three types of patterns. The patterns are defined by [DAC98] as follows (in falling occurrence):

- **Response** A state/event P must always be followed by a state/event Q within a scope.
- **Universality** A given state/event occurs throughout a scope.
- **Absence** A given state/event does not occur within a scope.

These types of rules can be expressed in CTL as follows:

1. **Response** `AG(P --> AF (S) )`
2. **Universality** `AG(P)`
3. **Absence** `AG(NOT P)`

In this paper we only use these simple CTL formulas for interpretation. In future, we will extend our interpretations to more complex formulas as defined in [DAC98] and [DAC09]. The interpretation of this subset of rules can be done in terms of process models.

Now, the interpretation of this subset of rules is possible.

**`AG ( P --> AF (S) )`** The process element `S` has to occur in a process step starting with `P`.

[1] Furthermore, they have been tested in a survey with over 500 property specifications. The properties are collected from a wide variety of application areas. Some areas which are important for this paper are communication protocols, GUIs, control systems or distributed object systems.

**AG (P)** The process element `P` has to occur in every process step.

**AG (NOT P)** The occurrence of process element `P` is forbidden in every process step.

The advice is given in terms of the original process model. For example, if the rule `AG ( Authorization is required --> AF (Approval request) --> AF (Approval decision evaluation) )` is not fulfilled by a model the advice would be:

*The event Authorization is required must be followed by the function Approval request must be followed by the function Approval decision evaluation in every subsequent process path.*

This advice is given in a textbox as shown in figure 8 for the example process of section 4. This textbox is directly connected to the frame of the erroneous path. This enables the process modeler to focus on the validation problem instead of browsing the whole process models.

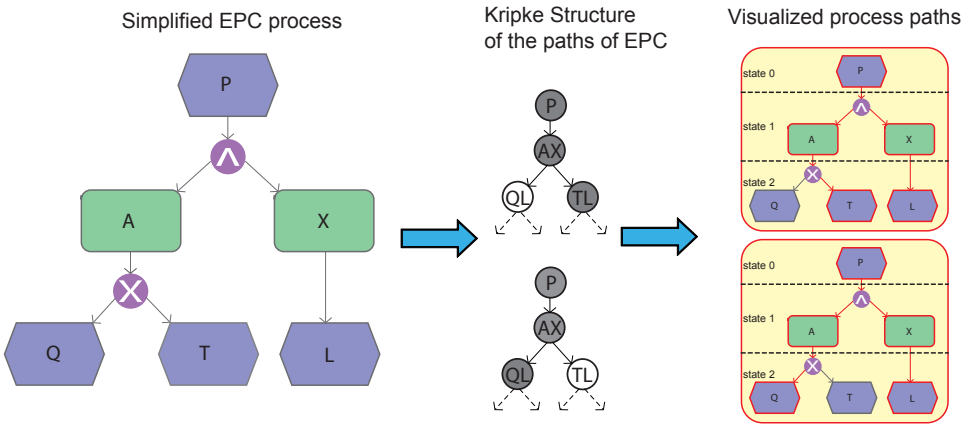


Figure 4: Visualization of paths.

### 3.2.2 Visualization of Validation Errors

The main task of the visualization of validation errors is to highlight every erroneous path in the original process model. In addition, if it is possible an advice for a highlighted erroneous path is given. An advice is possible if the regarded specification property matches one of the three specification patterns (Response, Universality and Absence). Figure 4 presents the visualization approach [2] as whole. The transformation of a simplified EPC process to a Kripke Structure is depicted on the left and middle of figure 4. Two possible paths of the EPC are shown as shaded elements in the Kripke Structure.

[2]The notation of the EPC is the one used by our developed editor.

The visualization of the different paths in the editor is shown on the right of figure 4. The frames of the process elements and arrows of a specific path are recolored in red. The advices are given in text boxes when selecting a erroneous path. In addition, for a better understanding of the states of the systems the process models are split by dashed lines. With this it is easier to locate the process elements belonging to a state. Furthermore, this supports the understanding and adjustment of the process model.

## 4 Example Process

Figure 5 depicts a typical, but rather small example of a (sub-) business process (named *Order finalization process*) as defined in the analysis phase of a project. The model describes the approval functionality in an e-procurement system modeled as EPC. This model is one small sub-process out of a set of some tens to several hundreds of such sub-processes (the number depends on the complexity of the system), which represent the complete business process description.

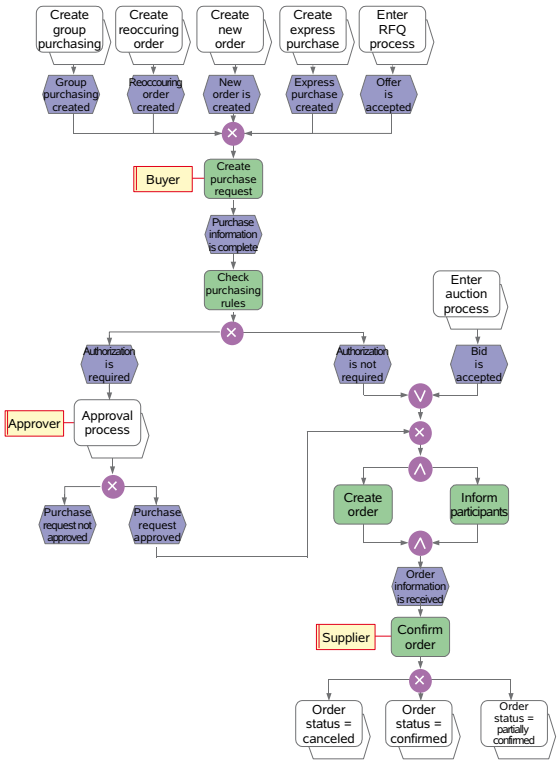


Figure 5: Example: Approval Procedure in the *Order finalization process* of an e-Procurement System.

When a tool – like TLVF (*Temporal Logics Visualization Framework*, c.f. section 3.1) – is used each of these sub-models is developed in a screen page of its own. Other tools similar to TLVF (but without integrated rule representation, though) realize the connections in-between these sub-models as hyperlinks.

In the e-commerce system domain there exist business rules (as in most other domains) to which the requirements models have to keep to. In most cases these rules are based on experience or legal regulations. Some examples for such rules are:

1. An order is always to be completed by the payment and the order confirmation. This seems to be simple. However, when a large number of sub-models are developed there is a certain risk, that there might occasionally exist a path which bypasses the payment. Maybe the payment procedure is bypassed for testing purposes.
2. If the condition that an *Authorization is required* exists then the next step is the *Approval Process*.
3. If the condition that an *Authorization is required* exists then specific functions in the sub-process *Approval Process* must exist.

The first example is a simple check if all paths contain the payment function. This problem may also be detected by a manual check. However, an automated check may save time and is less error-prone. Below in this paper we focus on the verification of the second and third example since they point to some typical problems.

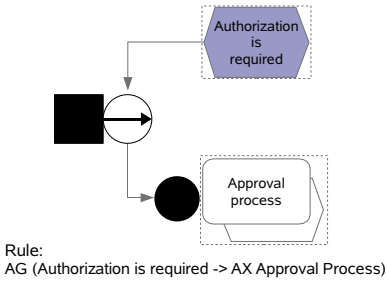


Figure 6: Temporal Rule 3: Graphical G-CTL Representation.

### 4.1 Checking a Process

One problem is that the completeness of the requirements model has to be ensured. In our example we want to verify that when *Authorization is required* the function *Confirm authorization* (implicitly followed by an approval process) is executed next. This is a typical business rule given in a requirements document.

The graphical version of this term as proposed in G-CTL is depicted in the figure 6. The CTL formula is shown at the bottom of the figure.

In case the verification fails (i.e. that the *Approval process* does not follow directly the event *Authorization is required*), it is obvious that the sub-process *Approval process* is missing after the event. The verifier then may recommend to insert *Approval process*. This recommendation is based on domain specific knowledge (or the rule itself, respectively).

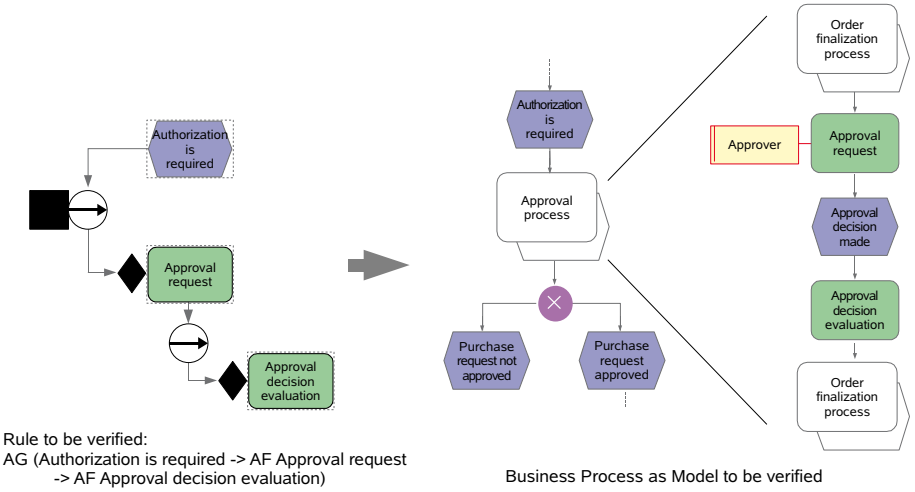


Figure 7: Checking Functions in the sub-process *Approval process*.

## 4.2 Checking Details in a Sub-Process

Another problem to represent and deal with rules is that only parts (as sub-processes) of a large business process are usually visualized on one window by graphical tools. The hyperlinks indicate that there is a connection to another part of the process which is not shown in detail (hidden). With this technique the models may be nested and connections to neighboring parts of the processes are indicated.

Figure 7 depicts the rule to be verified as well as the (sub-)process models (as hyperlink and inflated).

This concept allows human users to keep the overview. However, for checking purposes it is necessary also to be aware of the details. The automated checking concept proposed in this paper supports the complete verification. The checker will check the complete process regardless how the sub-processes are distributed on the different windows. Hence the details of a hidden subprocess may also be considered and checked. We can check that the function *Approval request* and later the function *Approval decision evaluation* are part of *Approval process* which is initiated by the event *Authorization is required*.

### 4.3 Detection and Presentation of Errors

In case the G-CTL is fulfilled there is no need for extra information. The model visualization will not be changed. When an error occurs, however, the model checking tool presents a counter example. This counter example helps to identify the error.

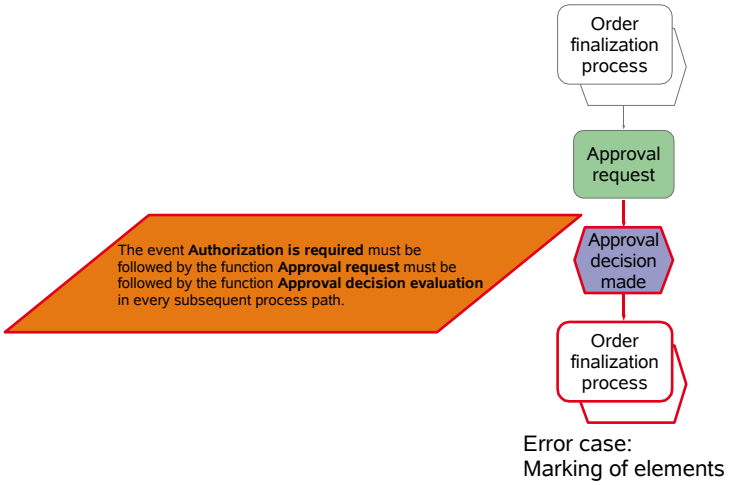


Figure 8: Error: *Approval decision evaluation* is missing.

The TLVF now interprets the counter example and displays the location in the path where the error occurs. In many cases this error description is suitable to correct the error.

In our case (depicted in figure 8) the function *Approval decision evaluation* is missing in the sequence. The text in the parallelogram describes the violated rule. The location of the error is marked with red lines surrounding the shapes of the misplaced functions and events as well as the control flow connecting these functions and events. This marking helps the user to identify the error and its location. The markings are realized be interpretation of the results of the model checking run.

## 5 Conclusion and Future Work

The Temporal Logics Visualization Framework (TLVF) enables a visual modeling and validation of temporal requirements. For business process development (e.g. e-commerce systems) it is essential to bridge the gap of the domain expert (and the domain rules like business rules to be considered by the requirements models) and the low-level verification viewpoint. Our work contributes in this task by providing a graphical logic language based

on CTL (G-CTL) as well as means to visualize validation errors. In particular, both parts are integrated in the EPC modeling language which is frequently used for business process modeling.

Besides the pure verification of the EPC business process requirements models of the results of the checking are displayed graphically in the Temporal Logics Visualization Framework (TLVF). Moreover, the checking system can not only mark the error, it may also recommend solutions. Simple solutions result from the temporal logic applied. More complex solutions, e.g. complete missing sequences depend on the application domain and have to be stored in repositories. Both features improve the applicability of such analysis concepts thereby helping to minimize the errors in the requirements documents.

Further work has to be carried out to support different viewpoints on the models. The domain expert should be able to select different abstraction levels and views on the business process model to reason about requirements.

Another improvement may be to apply our CoV (Component Verifier) model checker [Pul06] instead of the (currently used) model checker SMV. The CoV model checker is a symbolic model checker prototype particularly aiming at reducing the gap between low-level formal models and higher-level component-oriented software systems neglecting further consideration of performance optimization.

## References


[BDG+07] Paul Baker, Zhen Ru Dai, Jens Grabowski, Øystein Haugen, Ina Schieferdecker, and Clay Williams. *Model-Driven Testing: Using the UML Testing Profile*. Springer, Berlin, 1 edition, 2007.

[Bre02] Michael Breitling. Business Consulting, Service Packages & Benefits. Technical report, Intershop Customer Services, Jena, 2002.

[CDH+00] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Păsăreanu, Robby, and Hongjun Zheng. Bandera: extracting finite-state models from Java source code. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 439–448, New York, NY, USA, 2000. ACM.

[CDHR02] James C. Corbett, Matthew B. Dwyer, John Hatcliff, and Robby. Expressing Checkable Properties of Dynamic Systems: The Bandera Specification Language. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(1):34–56, 2002.

[CGMZ95] Edmund M. Clarke, Orna Grumberg, Kenneth L. McMillan, and Xudong Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *DAC '95: Proceedings of the 32nd ACM/IEEE conference on Design automation*, pages 427–432, New York, NY, USA, 1995. ACM.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts; London, England, 3 edition, 2001.

[DAC98] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the Second Workshop on Formal Methods in Software Practice*, pages 7–15. ACM Press, 1998.

[DAC09] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. A System of Specification Patterns. http://santos.cis.ksu.edu/spec-patterns/, February 2009.

[DP02] Daniel C. DuVarney and Iyer S. Purushothaman. C Wolf - A Toolset for Extracting Models from C Programs. In Doron A. Peled and Moshe Y. Vardi, editors, *Internation Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, volume 2529 of *Lecture Notes in Compter Science*, pages 260–275. Springer Verlag, 2002.

[EK04] M. Jose Escalona and Nora Koch. REQUIREMENTS ENGINEERING FOR WEB APPLICATIONS – A COMPARATIVE STUDY. *Journal of Web Engineering*, 2(3):192–212, 2004.

[FF08] Sven Feja and Daniel Fötsch. Model Checking with Graphical Validation Rules. In *15th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2008), Belfast, NI, GB*, pages 117–125. IEEE Computer Society, April 2008.

[GCKK06a] Heather Goldsby, Betty H. C. Cheng, Sascha Konrad, and Stephane Kamdoum. A Visualization Framework for the Modeling and Formal Analysis of High Assurance Systems. In *Proceedings of the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, Genova, Italy, October 2006.

[GCKK06b] Heather Goldsby, Betty H.C. Cheng, Sascha Konrad, and Stephane Kamdoum. Enabling a Roundtrip Engineering Process for the Modeling and Analysis of Embedded Systems. In *Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2006)*, Genova, Italy, October 2006.

[HD01] John Hatcliff and Matthew B. Dwyer. Using the Bandera Tool Set to Model-Check Properties of Concurrent Java Software. In Kim G. Larsen and Mogens Nielsen, editors, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR)*, number 2154 in Lecture Notes in Computer Science, pages 39–58. Springer, Denmark 2001.

[KC05] Sascha Konrad and Betty H. C. Cheng. Facilitating the Construction of Specification Pattern-based Properties. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 329–338, Washington, DC, USA, 2005. IEEE Computer Society.

[KGLC06] Sascha Konrad, Heather Goldsby, Karli Lopez, and Betty H. C. Cheng. Visualizing Requirements in UML Models. In *REV '06: Proceedings of the 1st international workshop on Requirements Engineering Visualization*, page 1, Washington, DC, USA, 2006. IEEE Computer Society.

[Kon06] Sascha Konrad. *Model-driven Development and Analysis of High Assurance Systems*. PhD thesis, Michigan State University, East Lansing, MI, October 2006.

[MC01] William E. McUmber and Betty H. C. Cheng. A general framework for formalizing UML with formal languages. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 433–442, Washington, DC, USA, 2001. IEEE Computer Society.

[OMG06] OMG. Business Process Modeling Notation (BPMN) Specification. Technical report, Object Management Group (OMG), Februar 2006. http://www.omg.org/docs/dtc/06-02-01.pdf.

[PO98] Paul W. Parry and Mehmet B. Özcan. The Application of Visualisation to Requirements Engineering, 1998.

[Pul06] Elke Pulvermüller. *Verifikation von Komponenten-basierten Systemen auf Basis eines erweiterten temporalen Verifikationsverfahrens.* PhD thesis, Friedrich-Schiller-Universität Jena, January 2006.

[Pul09] Elke Pulvermüller. Reducing the Gap between Verification Models and Software Development Models. In *Proceedings of the 8th International Conference on New Software Methodologies, Tools, and Techniques (SoMeT_09)*. IOS Press, September 2009. To appear.

[Sch98] August-Wilhem Scheer. *ARIS - Modellierungsmethoden, Metamodelle, Awendungen.* Springer, Berlin, 1998.

[SPJF02] Andreas Speck, Elke Pulvermüller, Michael Jerger, and Bogdan Franczyk. Component Composition Validation. *International Journal of Applied Mathematics and Computer Science*, 12(4):581 – 589, December 2002.

[VS06] Markus Völter and Thomas Stahl. *Model-Driven Software Development : Technology, Engineering, Management*. John Wiley & Sons, June 2006.