Developer-friendly verification of process-based systems

Elke Pulvermueller a,*, Sven Feja b, Andreas Speck b

a Department of Mathematics and Computer Science, University of Osnabrück, Albrechtstraße 28, 49069 Osnabrück, Germany
b Department of Computer Science, Christian-Albrechts-University of Kiel, 24098 Kiel, Germany

ARTICLE INFO

Article history:
Received 26 August 2009
Received in revised form 15 March 2010
Accepted 15 March 2010
Available online 21 March 2010

Keywords:
Business process verification
Graphical specification
Graphical result presentation
Extended temporal logic
User-friendliness
Model checking

ABSTRACT

System quality is a key issue in modern systems development. Tool support is essential for checking the system quality efficiently. This is particularly true with respect to the dynamic interactions of the processes within a system. A first generation of checkers – model checkers – provide a basic technology for the verification of process-based systems.

Conventional model checkers bear two drawbacks concerning mainly their user-friendliness which impede their broad application. First, model checkers in general do not support the graphical representation of rules (specifications). Although a model may be described with a graphical notation, the specification which has to be checked against the model is generally still text-based. This makes the usage of the checker difficult for process modeling experts. Second, the expressiveness concerning the verification model semantics to be checked is limited to states which are connected by transitions. However, many system development models (e.g. the business process model we use as example) embrace more element types. These are unsupported by the conventional model checkers resulting in a loss of verification precision.

The checking system we present in this paper integrates both novelties: the graphical notation for a user-friendly specification and an extended specification language together with a corresponding verifier which supports the checking of many different types of elements (although the paper presents the approach with only two types). The integration is realized by an XML-based transformation system which links the graphical editor to the checking tool.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

System quality is a key issue in the software development. The goal of software engineering is to build correct models and systems without errors. Checking tools and concepts to apply these tools have been developed. In this paper we introduce an integrated approach supporting the user-friendly modeling as well as the checking.

In general there are two dimensions in software systems to be examined: the static system structures and the dynamic processes.

The static structures are, for instance, the hierarchies and dependencies between functions or classes as they may be expressed with UML class-diagrams. These static dependencies may be best formalized in Boolean logic formulas. The dependencies are require or exclude. Moreover, there are alternative and option dependencies. We may evaluate the Boolean equations and verify the correctness of them. Verification tools like constraint solving systems support this approach [29].

The dynamic processes and workflows describe the system states and their temporal changes. There are numerous models to express the behavior of systems: the two UML interaction diagrams (sequence and collaboration diagram or communication diagram, respectively) may be used for a detailed modeling near to the code level. More abstract diagrams are the UML activity diagrams, BPMN models (Business Process Modeling Notation) or ARIS EPCs (cf. Section 2) which all may be used to describe business processes.

All these system development models are rich in notation and semantics to support the modeling close to the users’ needs. The goal of this paper is to decrease the gap between verification models (together with their languages) and those semantic-rich system development models. This gap impedes the useful application of formal verification techniques by system development experts. Current verification models are restricted and simple for the purpose of a feasible verification. The price for this simplicity is a loss of verification precision and a modeling gap in the world of the system developer. We propose to extend the verification model and also the corresponding specification language to enable a more precise verification where required. The extensions are controlled to avoid an unreasonable increase in verification complexity.
Besides more precision to express additional, required temporal properties (based on existing system development models) we combine the specification language with an easy-to-use notation. Such a notation has to be embedded into the graphical modeling world of the system developer. Currently, besides the semantic gap a notational gap exists. If a system development expert wants to verify temporal behavior using a model checking approach usually s/he has to write a text-based formal specification (e.g. the CTL temporal logic formulas). We aim at integrating the specification and the development models. The developer should be able to express correctness rules directly in the development model and with similar means instead of in a completely different language.

Both the extended specification language (together with its corresponding verifier) and the graphical notation in combination lead to a new checking system for process-based systems.

The application domain of this paper consists of business processes described with EPC models. EPCs are frequently used in the development of commercial (software) systems. These models may be considered as abstract. However, the business process models are the direct base to derive code models. A correct business process model must be considered as essential for the correctness of the system to be developed. Moreover, the EPC models we use have already at least one thing in common with other development models used in other domains: they embrace different types of model elements (and not just states) which have to be distinguished – this model element distinction is a major issue of this paper.

1.1. Paper outlook

The notation for modeling business processes and its application are subject of the following Section 2. Moreover, our graphical support to express specifications (the G-CTL notation) is also part of this section.

Section 3 presents the extensions of the specification language in order to achieve a higher precision or expressiveness, respectively (the ECTL1 extended temporal logic language). This is followed by the integration of the G-CTL graphical notation and the ECTL1 specification language (resulting in EG-CTL) in order to exploit the benefits of their combination – a more powerful and user-friendly checking system.

An example of the graphical presentation of a larger business (sub-) process and rules (specifications) for this model is given in Section 4. The second part of Section 4 describes the architecture of our checking system for process-based systems.

2. Business process models and specifications

In the paper we focus on business process model types which are typical for commercial systems: Event-driven Process Chains (EPCs) [30] which are part of the modeling concept ARIS (Architecture of Integrated Information Systems). This modeling concept has first been brought up to model large scale ERP systems like SAP R/4. Now EPCs are used to model almost all kinds of commercial systems.

The EPC models are the main means used to describe business processes. These may be the processes within commercial systems such as ERP systems (enterprise resource planning systems) or e-commerce systems. For example the largest commercial standard e-commerce system Intershop Enfinity provides a specific ARIS profile with a main focus on EPC models: ARIS4Enfinity [6]. EPCs may also be used to model administrative procedures of government institutions e.g. taxation or residence regulations according to European rules.

Our approach will be demonstrated by means of business process examples modeled with EPCs. These examples are typical for the e-commerce domain [6].

2.1. Event-driven process chain model elements

An EPC is an ordered graph of events and functions. The connectors between functions and events allow alternative and parallel execution of processes. A detailed description may be found in [33].

The main elements in the EPC models are (cf. Fig. 1):

- Functions are considered as active elements in EPCs. They describe functionality such as tasks or activities. Functions represent transformations from one state to another, follow-up state. If different follow-up states can occur, the selection of the respective follow-up state can be modeled explicitly by logical connectors (as described below). Functions may be refined into another EPC (hierarchical functions). In the EPC model rounded rectangles represent functions.

- Events are passive elements which describe the conditions or circumstances which result from functions or are triggering the execution of functions. An event is represented by a hexagon.

- The control flow connects events, functions or logical connectors creating a chronological sequence and depicts the logical interdependencies between them. Control flows are represented by arrows.

- Logical connectors express the logical relationships between elements in the control flow (events and functions). The relationships correspond to the logical operations AND, OR and XOR. Fig. 1 depicts the graphical representation of an XOR relationship. The notation elements for AND or OR are similar and with the corresponding Boolean symbol within the circle (cf. Table 1, left column). An XOR in a control flow defines a branching point or branch, respectively. There a decision is required which follow-up state (or path, respectively) is to be taken exclusively. The counterpart of a branch is a merge which means that different branches are merged into one. Branches as well as forks use the same symbol. An AND may represent the fork or join in the control flow. A fork activates the outgoing control flows in parallel. The join synchronizes incoming control flows. OR is the weakest relation. An opening OR connector activates one or more control flows and deactivates the rest of them. The counterpart of this is the closing OR connector which activates the control flow when at least one of the incoming control flows is activate.

Besides these EPC model elements there are several others. A further remarkable element is the organizational unit and its assignment which describes the connection between an organizational unit (a person or an organization responsible for a specific function) and the function it is responsible for.
2.2. Graphical specification

EPC graphs are visual representations of a process model. However, the languages to express temporal logic statements are text-based. The Temporal Logics Visualization Framework (TLVF) provides a graphical notation to express the temporal specifications [14]. By means of TLVF, a system developer is able to develop and verify within the same modeling world.

TLVF contains all required means for the visualization of temporal logics in connection with and integrated into a process model or workflow. As depicted in Fig. 2 the framework is divided in three layers:

- The lowest layer (base layer) supports different temporal logic languages. Besides CTL and especially ECTL (as proposed in this paper, cf. Section 3.2) other temporal logics such as LTL or CTL* may be used. A mapping between the graphical representation and the respective temporal logic is required.

- The second layer defines the graphical symbols for each operator of a logic language. The graphical symbols for CTL are the G-CTL symbols shown in Fig. 3.

- The third layer provides the process model, the graphical rules definition and the required transformation tasks. These are the tools for the presentation of the process model and the graphical specification as well as the transformation system. The latter is responsible for the transformation between the graphical representation and the checker (input/output).

This layered framework allows stating graphical rules for any logic language (e.g. CTL or LTL) integrated and connected with the desired process model (e.g. EPC). The model and the rules are within one and the same model instead of the usual strict separation. Moreover, the transformation of the process model and the graphical rules to an appropriate format for different model checkers is supported.

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 the language elements of Graphical CTL (G-CTL) is shown in Fig. 3. G-CTL operators are based on CTL operators. In CTL there are two types of operators which are combined pairwise: Path quantors always (A) and exists (E) which indicate the occurrence within a path. The temporal operators determine the temporal order. The most important temporal operators are: in the future (F), globally (G), next (X) and until (U). Examples for pairwise combinations are: AG — always globally or EX — exists next.

Fig. 4 presents a concrete example of a G-CTL specification. It states that the function Check all current offers has to be true until the event Offer accepted becomes true. Fig. 4 demonstrates the application of the generic formula format A(aldb) of Fig. 3. The elements with dashed lines in the generic formula pattern represent the placeholders. They are replaced by concrete element instances (e.g. Offer accepted event).

3. Expressiveness of the specification language

Formal methods are a powerful tool to ensure a system's compliance with the requirements and expected quality properties. The goal of this paper is to overcome the obstacles hindering the broad application of formal methods. Formal languages play a key role in achieving this goal as they provide the interface be-
between the human developer and the formal, often misanthropic world. Above, we presented G-CTL which transforms the syntax of a formal temporal logic language into a more user-friendly representation. Besides a graphical representation, we propose, furthermore, an improvement of the expressiveness of the formal language.

3.1. The need for more expressiveness

The specification language of known model checkers (e.g. SMV) is based on finite state automata and, therefore, does only support states and their transitions. This may lead to problems if our specification has further model elements. For instance, instead of just states we have functions and events in EPCs. We experience a modeling gap between the models such as business process model (like EPC models) and the low-level models used during a formal verification.

In case of a verification request the general approach to tackle this gap is as follows: the models of commercial software systems are transformed to a finite state model. This formal model is called Kripke structure (or Kripke model). A Kripke structure is a finite state automaton in the form of a 4-tuple consisting of a finite set of states, a set of initial states (a subset of the set of all states), a transition relation and a labeling function. The latter marks states with certain properties or propositions, respectively, which hold in the respective states.

If we just transform all the functions and events of an EPC model into states of a Kripke structure with a surjective mapping important semantic information is lost. After the transformation, we cannot distinguish between functions and events anymore – both are states in the verification model.

Fig. 5 provides an example demonstrating the problem. This EPC model represents the price alert sub-process. It starts with another sub-process which describes the determination of the price threshold and is continued by the payment and order fulfillment.

In the price alert sub-process we check all current offers as long as there is none within a given price limit (i.e. as long as the price threshold is not reached). We assume the business system developer wants to check the model part within the red rectangle against the G-CTL specification from Fig. 4. Using the described transformation approach leaves us unsatisfied. After the transformation each event as well as each function is represented as a state in a finite state automaton. What we want to know is if it is true that the function Check all current offers is true along the model paths until Offer accepted becomes true (only referring to the model excerpt within the red rectangle). However, we only want to know if the function Check all current offers is true without any "interrupting" other functions along the workflow paths. In particular, we do not care about the event Offer not accepted along the workflow paths. If functions and events are both mapped to states the result of this check will be that the temporal requirement is false due to the event Offer not accepted. The model checking algorithms cannot identify that Offer not accepted is an event. Therefore, we need a model checking algorithm which selectively ignores specific model element types.

The transformation applied in Fig. 5 is straightforward since each model element is mapped to a state. Each model element which might be a subject of verification has to be a state in the traditional model checking technique. Assuming we would map the function Initiate price alert to a transition we could not check a specification referring to this function using model checking.

Table 1 completes the straightforward mapping of EPCs to finite state automata providing the different except business process variants. Although this paper has not the intention to present a sophisticated or novel transformation from EPC business process models to Kripke structures the provided mapping might support the understanding of the following sections. As described in Section 2.1 EPC models support three kinds of branching connectors: AND, OR and XOR. The details of the straightforward transformation of these branches are as given in Table 1. Each EPC in the left column of Table 1 is mapped to the corresponding automaton in the right column. The AND branch corresponds to a fork where all outgoing control flows are processed in parallel. There is no specific condition to be fulfilled by the outgoing flows. The flows are activated in parallel. In a join the ingoing control flows are synchronized. This synchronization may be realized in the corresponding automaton by means of conditions which may be attached to the follow-up state (e.g. to state Order_is_invoice). The OR and XOR connectors use conditions to control the outgoing flows.
contrast to the OR connector the XOR connector allows only one outgoing flow to be activated. The merge of XOR and closing OR connector activate their outgoing flow when one incoming flow gets active (in case of the XOR merge only one incoming flow may become active).

As we aim at a model checking based verification the mapping has to be to a finite state automaton or Kripke structure, respectively. Mapping the EPC to a Petri net, for instance, would require a second mapping from the Petri net to the finite state automaton. Another approach would be to apply another formal checking technique, e.g. Petri net based simulation and checking techniques. However, even then, the problem outlined in this paper would exist. Using Petri nets, for instance, we could map functions to transitions and events to places. The mapping of organizational units and the other model elements in EPCs, however, will, again, lead to a surjective mapping and thus a loss of information when transformed.

Although the problem is general and although also our solution could be transferred to other kinds of models (e.g. BPMN models, Petri nets) in principle, this paper only focuses on EPC models and a verification based on model checking.

3.2 Towards more expressiveness with the extended temporal logic language ECTL1

To enable a more fine-grained specification and a more sophisticated verification it is essential that the distinction between the different model elements (in our example, events and functions) is preserved in the verification model. The middle column of Fig. 6 symbolically maps events to rectangles (named transition states) and functions to circles (named states). In other words, the verification model (the Kripke structure) now supports two different kinds of model elements instead of just states. We have, therefore, extended the Kripke structure. It is called ELKS1 [28].

Besides such an additional distinction for different model elements we need an extension to the formal language to exploit the additional expressiveness captured in our verification model. We have developed the temporal logic language ECTL1. Compared to traditional CTL so-called specializers are supported in addition. They allow the developer to distinguish between requirements referring to states (e.g. functions) and those referring to transition states (e.g. events) during the verification step.

The specializers and their usage are shown in the example of Fig. 6 (right column, bottom). In ECTL1 a formula for our requirement of above (i.e. “function Check all current offers (p) is true along the model paths within the red rectangle until Offer accepted (q) becomes true”) has the form: \( A[S[p] \cup T[q]] \).

The specializer \( S[p] \) restricts all checking for \( p \) to states (in our EPC example: functions). The specializer \( T[q] \) only considers transition states (in our EPC example: events) ignoring all states at the same time. These additional specializers allow formulating temporal logic requirements over a model which distinguishes different kinds of states.

In general, model checkers produce a set of states. Each state within this result set may be used as a starting state to fulfill the given temporal statement. Internally, the model checker algorithms work recursively with such sets [9]. By means of the introduced specializers it is possible to produce a view on the result set which contains only states or only transition states. For instance, the specializer \( S \) filters and thus limits the result set produced for the ECTL1 temporal logic formula \( S[AX T[p]] \) to only states (ignoring transition states in the final result set).

The interested reader is referred to [28] for more details on the definition and use of ELKS1, ECTL1 together with the satisfaction function and the implementation of the correspondingly adapted model checking algorithms.

An alternative approach to the presented one may be to use the straightforward mapping of the EPC to an automaton (as in the
example of Fig. 5). All model elements which are an issue in the checking procedure have to be transformed to states. In the general model checking approach transitions cannot be labeled. To distinguish now between the different types of model elements we mark the states accordingly. States representing events are labeled with the property IS_EVENT while states representing EPC functions are assigned the property IS_FUNCTION.

Using this model we try to find an appropriate formula for the example requirement of Fig. 6 and experience severe problems with this task. The straightforward mapping to \( A[(p \land \text{IS\_FUNCTION}) \lor (q \land \text{IS\_EVENT})] \) is semantically different to what the developer wants to express. A model checker would return false for the excerpt in the red rectangle of Fig. 5. In both states, Offer_not_accepted and Offer_accepted, the first part of the formula \((p \land \text{IS\_FUNCTION})\) does not hold. Though the second part of the formula \((q \land \text{IS\_EVENT})\) is fulfilled in one of the two states it does not hold in the second \((q\) does not hold in the state Offer_not_accepted). Due to the Always-operator \(A\) the overall formula does not hold in the considered excerpt. Using other Boolean combinations does not lead to the expected result either (e.g. the formula \(A[(p \land \text{IS\_FUNCTION}) \lor (q \lor \text{IS\_EVENT})]\)). The cause of this problem is due to the fact that both model elements, events and functions, are checked for the second part of the formula \((q \land \text{IS\_EVENT})\). Using the specializers, however, the functions are completely ignored when performing the check of the second part of the formula. The model checking algorithms have to be adapted to perform the requested semantics which we presented at the beginning of this section.

In our previous work (c.f. [31] and [28]) we used this unsatisfying alternative labeling approach and examined the problem.
3.3. G-CTL and ECTL1

Using ECTL1 a business system developer may express the verification requirements in a more precise way and closer to the business system developer’s view. This additional expressiveness reduces the modeling gap as described in the beginning. However, it also increases the modeling complexity to a certain extent: If a developer wants to express the requirements in a more sophisticated way (e.g. using ECTL1) s/he has to learn and apply the syntax and semantics for the respective language.

Where a detailed verification is not necessary and where simplicity has priority the new specializers may be skipped. Otherwise, the new specializers of ECTL1 should be reflected in the user-friendly graphical modeling notation G-CTL. Fig. 7 presents EG-CTL, the extended G-CTL which embraces the specializers in addition. They form rectangles with round edges and surround the properties. This symbolizes that they operate like filters or wrappers as described above.

4. Realization and architecture

This section presents a slightly larger example of a realistic business process model and its checking in order to demonstrate the procedure of applying our checking concept. Based on this we introduce the system architecture of our checking system for process-based systems.

4.1. Checking a business process model

As example we choose an e-procurement system. This is a specific type of e-commerce system used to automate the procurement of mostly large organizations or companies which reduces the cost for procurement considerably. A quite well-known example may be the large e-procurement system of the German federal government.

Fig. 8 shows a realistic business process EPC model – one of a large number of sub-processes as part of the overall procurement process. Starting from different order types (created in other sub-processes) a purchase request is initiated which is followed by a fork in an authorization path exclusive or the purchase without authorization. Finally, the concrete order is created and processed by different other sub-processes.

This sub-process may still be considered as a small one compared to other sub-processes (which are too large for presenting them in a paper). This gives an impression why checking tools are useful in business process systems. A tool automatically checks large process models and combinations of sub-processes. This is a very difficult task for a human being. Therefore, such a tool support is highly effective in a quality assurance procedure.

The example rules to be checked are specified in Fig. 9. We focus on the approval branch. In the example, we want to verify that whenever the event Authorization is required occurs it is immediately followed by the function Approval request which then is immediately followed by the function Approval decision evaluation. In particular, in our example scenario there may be any event between these two functions. Therefore, from a verification point of view the concrete event (or the question if there is an event at all) is not of interest.

Fig. 9. EG-CTL specification examples.

Fig. 10. Visualization of the violated specification (the error path is marked red/bold). (For interpretation of references to color, the reader is referred to the web version of this article.)
While a model checker will return with false reporting an error for the left rule (the graphical presentation of this error is depicted in Fig. 10) the rule on the right column of Fig. 9 is correct (i.e., the model checker returns with true as expected). The rule statement using the specializers allows a more specific checking. The rule on the left of Fig. 9 cannot express what the developer intended to check as it does not distinguish between events and functions.

Fig. 10 presents the visualization of errors. While most model checkers just present a counter example (in a textual mode [9]) the TLVF (cf. Section 2.2) provides the possibility to visualize the erroneous path graphically. Again, this error representation is directly integrated into the development model (here, the EPC model) reducing the development gap. The path containing the error is highlighted in red. This eases the task of finding the cause of the error. The developer does not have to switch between different model worlds.

4.2. Checking system architecture and usage

The two core elements of our checking system have already been introduced: the graphical editor and the model checker with an extended specification language. Both are integrated by a transformation system which is part of the Transformation Component in the third layer of TLVF (cf. Section 2.2). The transformation system is detailed in Fig. 11. The system is derived from our generic transformation system XTC (XML Transformation Coordinator [16,17]). XTC is a hierarchical transformation system applying XSLT.

The graphical editor is based on Eclipse. The editor embraces two parts—an EPC and EPC-G-CTL editor. Both editors are generated using the Eclipse frameworks GMF [2] and EMF [1]. While, the editors may be used independent to each other they are based on a shared domain model. That means, an editor is able to show only its elements and connections and ignores everything else. For instance, the model editor does not present the specification elements.

The verification is done by our new model checker called CoV (Component Verifier) which aims at supporting higher-level verification with adapted CTL model checking algorithms [28]. CoV follows the traditional CTL model checking algorithms but extends them to process an extended temporal logic (i.e., ECTL1 with its specializers) and to produce conditional results (conditional verification).

The overall checking procedure is as follows:

1. The system developer creates the EPC model together with the graphical rules in the same graphical editor (using similar notation).
2. The EPC model and the EG-CTL specification are both exported to XML format.
3. The XTC system transforms both parts to the CoV input (i.e., to the ELKS1 model and the ECTL1 specification).
4. CoV checks the model.
5. In case of an error, the checker indicates the erroneous path in the model. If an error is detected by the underlying model checker the XTC system transforms the checking result to an XML input for the Eclipse visualization. The Eclipse visualization presents a highlighted path which indicates the error (as depicted in Fig. 10).

5. Related work

The verification of software models has been a key issue in the model checking research for a long time (as in [12] or [25]). An introduction and overview about many of these approaches is given in [4].

Based on early approaches to formalize business process models like [22] ideas how to apply formal methods for business processes emerged. Examples for early approaches employing model checking on business processes are [23,3] or [31] and [32]. [23] evaluates different checking technologies for their application to business processes. The role of business rules as specifications is discussed in [32].

Further research on the verification of business process systems is the evaluation of different technologies in the area of model checking and other formal techniques in order to solve specific problems. Such problem-specific solutions have been elaborated by [13], for instance. Similarly, [19] investigate the checking of process patterns by applying a PROLOG based concept.

Other approaches for the verification of business process systems are based on Petri nets (e.g. [10] using BPMN). With Petri nets the business processes are mapped to the Petri net elements similar to our Kripke structure mapping. In the Petri net based approaches the verification is often based on bi-simulation and algebraic solutions (e.g. [26]). In the background model checker technology may be used for the process verification. Our approach relies directly on the push-button model checking technology and temporal logic requirement specifications.

Most approaches applying formal methods to business process models for the purpose of checking rely on pure model transformations. Such transformations result in a loss of information and, therefore, verification precision mainly for two reasons. First, the incompatible semantics of the business process models and the verification models cause several problems resulting in different alternative approaches to tackle them [35]. Second, additional information is lost during the transformation due to the above detailed surjective mapping. [34,27] are only two of those approaches transforming business processes models to verification models (here, Petri nets and SMV Kripke structures). As opposed to that we propose to enrich the verification models and increase the precision of the specification as well as of the checking result.

Another research direction is the realization of checking environments as we propose in our paper. [15] stresses the importance of a graphical representation of the model. However, they derive the notation from UML activity diagrams and the result of the LTL-based checking is presented in a textual manner.

In the domain of formal methods approaches may be found which concentrate on an increase of semantic expressiveness of the specification languages (e.g. the μ-calculus [5] and [24] or in the multi-valued logic research as in [8]). Extensions to the temporal logic for LTL have been proposed in [7] or [20], for instance. In these approaches a link to software models or business process models is missing. In contrast to [7,20,18] and [21] we are able to explicitly distinguish and mix specializers (and thus views) for different model elements. The general idea of specializers for different model elements has not been elaborated yet otherwise. A user-friendly integration into an overall checking environment for system development models is beyond the scope of the mentioned approaches.

6. Summary and outlook

Current model checking technology allows checking models of dynamic processes. However, conventional model checkers are not user-friendly enough. Our checking system integrates two novelties in order to improve the usability of checkers: A graphical notation for a user-friendly specification and an extended specification language together with a corresponding verifier – CoV (Component Verifier).

The graphical notation of the specifications (EG-CTL, the extended graphical CTL) is intentionally close to the notation of the initial system development model (in our case ARIS EPC business process models). The extended specification language enables to express the specifications more precisely if necessary. It supports the checking considering the many different types of notation elements. As most current system development modeling languages (e.g. ARIS EPC models) embrace a large number of notation elements the gap to the verification model is reduced in our approach. The integration of the graphical editor and the checker is realized using our XML-based transformation system XTC (XML Transformation Coordinator) [16].

We are currently working on further extensions of the specification language to increase the user-friendliness furthermore while keeping the verification task feasible. The employment of error patterns [11] might be one way. Future work may be also found in the exploitation of the hierarchical nature of current system development models. Compositional verification approaches may be integrated.

References

[15] Alexander Förster, Gregor Engels, Tim Schatkowski, Ragnhild Van Der Straeten, Verification of business process quality constraints based on visual
process patterns, in: Proceedings of the First Joint IEEE/IFIP Symposium on
[16] Daniel Fötsch, Elke Pulvermüller, A concept and implementation of higher-
[17] Daniel Fötsch, Andreas Speck, XTC – The XML transformation coordinator for
XML document transformation technologies, in: DE'06: Proceedings of the
17th International Workshop on Database and Expert Systems Applications,
[18] Dimitra Giannakopoulou, Jeff Magee, Fluent model checking for event-based
systems, Proceedings of the 9th European Software Engineering Conference
( ESEC) held jointly with 10th ACM SIGSOFT International Symposium on
[19] Volker Gruhn, Ralf Laue, Checking properties of business process models with
logic programming, 5th International Workshop on Modeling, Simulation,
Verification and Validation of Enterprise Information Systems (MSVVEIS'07),
[20] Bengt Jonsson, Ahmed Hussain Khan, Joachim Parrow, Implementing a model
checking algorithm by adapting existing automated tools, in: Proceedings of
the International Workshop on Automatic Verification Methods for Finite State
[21] Ekkart Kindler, Tobias Vesper, ESTL: a temporal logic for events and states, in:
Proceedings of the 19th International Conference on Application and Theory of
[22] Christoph Klauck, Heinz-Jürgen Müller, Formal business process engineering
based on graph grammars, International Journal on Production Economics 50
[23] Jana Köhler, Giuliano Tirelli, Santhosh Kumaran, From business process
model to consistent implementation: a case for formal verification methods,
in: 6th International Enterprise Distributed Object Computing Conference
[24] Dexter Kozen, Results on the propositional mu-calculus, Theoretical Computer
1993.
[26] Shoichi Morimoto, A survey of formal verification for business process
modeling, in: ICCS 2008, 8th International Conference, LNCS, 5102, Springer,
[27] Elke Pulvermüller, Composition and correctness, in: SC 2002: Workshop on
Software Composition, Electronic Notes in Theoretical Computer Science
[28] Elke Pulvermüller, Reducing the gap between verification models and software
development models, The 8th International Conference on Software
297–313.
[29] Wolfgang Runte, Modelling and solving configuration problems on business
processes using a multi-level constraint satisfaction approach, in: The Young
Researchers Workshop on Modeling and Management of Business Processes
[30] August-Wilhem Scheer, ARIS - Modellierungsmethoden, Metamodells,
[31] Andreas Speck, Variabilitäten in dynamischen Abläufen, Habilitation,
Friedrich-Schiller-University, Jena, Germany, 2004.
[32] Andreas Speck, 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, 2006,
pp. 857–863.
[33] Anni Tsai, Jiacun Wang, William Tepfenhart, Daniela Rosea, EPC Workflow
model to WIFA model conversion, in: 2006 IEEE International Conference on
[34] Will M.P. van der Aalst, Formalization and verification of event-driven process
M.P. van der Aalst, Verification of the SAP reference models using EPC
reduction, state-space, analysis and invariants, Computers in Industry 58 (6)