Proposal of a Hierarchical Approach to Formal Verification of BPMN Models Using Alvis and XTT2 Methods*

Krzysztof Kluza, Grzegorz J. Nalepa, Marcin Szpyrka, Antoni Ligęza

AGH University of Science and Technology,
al. Mickiewicza 30, 30-059 Krakow, Poland
{kluza,gjn,mszpyrka,ligeza}@agh.edu.pl

Abstract BPMN is a visual notation for modeling business processes. Although there are many tools supporting it, they rarely provide formal verification of models. We propose a new approach to formal verification of BPMN models using the Alvis modeling language and the XTT2 knowledge representation. The structure of the BPMN model can be analyzed using translation to Alvis. Alvis models can be verified with dedicated tools, and their properties can be linked to the properties of the original BPMN model. On the other hand, selected BPMN elements can be verified using the XTT2 decision tables. Several BPMN elements can be translated to XTT2 and checked using the HeaRT rule engine with the HalVA verification and analysis tool. The paper constitutes an overview of the methods and concepts and presents preliminary results of our research.

1 Introduction

Business Process Model and Notation (BPMN) has recently emerged as a leading visual notation for modeling Business Processes. A BPMN model defines how the organization works by describing the ways in which operations are carried out to accomplish its intended goals. The progress in using the BPMN notation and the increasing complexity of the modeled processes make new advanced methods and tools needed.

BPMN provides a large collection of notation elements and allows for modeling various workflow structures, such as conditional operations, loops, event-triggered actions, splits and joins of sequence flow, etc. Moreover, it supports the hierarchical approach to design; thus, the process can be modeled on several abstraction levels.

The complexity of BPMN makes the formal verification of models a tough task. Although there are many tools supporting BPMN modeling, most of them do not provide any kind of formal model verification. In this paper, a new hybrid approach to formal verification of BPMN models is presented. It uses Alvis [1] and Extended Tabular Trees version 2 (XTT2) [2] methods. The considered approach is partially based on our previous research [3,4]. It extends and separates the verification process into two layers: the structure (or flow) layer and single components (mainly tasks) of the BPMN model.

*The paper is supported by the *BIMLOQ* Project funded from 2010–2012 resources for science as a research project.

This hierarchical separation provides verification of distinct properties on different abstraction levels. For the global (process structure) verification, the translation to Alvis modeling language is considered. The structure of the BPMN model can be analyzed thanks to its similarity to Alvis model, which is suitable for information systems modeling with subsystems working in parallel. For the local (model elements) verification, verification of single BPMN elements is considered. Such BPMN elements are mapped to the XTT2 knowledge representation, which can be verified using the HeaRT rule engine [5] as well as the HalVA verification and analysis tool [6].

This paper constitutes an overview of the methods and concepts, and presents preliminary results of the research aiming at formal verification of selected BPMN models using Alvis and XTT2 methods. We have limited the presentation to describing a simple yet illustrative case study of a student's project evaluation process.

The rest of the paper is organized as follows. Section 2 presents the BPMN notation and the selected case study. In Section 3 several works related to our research are presented. The BPMN model structure verification concept is introduced in Section 4, while the BPMN elements verification concept is given in Section 5. The evaluation of our approach is presented in Section 6. A short summary is given in the final section.

2 Business Process Modeling Notation

A Business Process can be defined as a collection of related tasks that produce a specific service or product for a particular customer. Business Process Model and Notation (BPMN) is a leading visual notation for modeling Business Processes. It uses a set of predefined graphical elements to depict a process and how it is performed. The current version of BPMN defines three models to cover various aspects of processes:

1. *Process Model* – describes the ways in which operations are carried out to accomplish the intended objectives of an organization. The process can be modeled on different abstraction levels: *public* (collaborative Business 2 Business Processes) or *private* (internal Business Processes).
2. *Choreography Model* – defines expected behavior between interacting business participants in the process.
3. *Collaboration Model* – can include Processes and/or Choreographies, and provides a Conversation view (which specifies the logical relation of message exchanges).

In our research, the internal Business Process Model is considered. There are four basic categories of elements used to model such processes: flow objects (*activities*, *gateways*, and *events*), connecting objects (*sequence flows*, *message flows*, and *associations*), swimlanes, and artifacts.

For the purpose of our research, only a subset of BPMN elements (flow objects and sequence flows) is considered. A task is a kind of activity, and a model defines the ways in which individual tasks are carried out. Gateways determine forking and merging of the sequence flow between tasks in a process, depending on some conditions. Events denote something that happens in the process. The icon in the event circle depicts the event type, e.g. envelope for *message event*, clock for *time event* (see Fig. 1).

Let us analyze an exemplary BPMN model of a simple student's project evaluation process. Fig. 1 depicts the evaluation process of a student's project for an Internet technologies course. However, the process can be used for any kind of a project depending on the rules which are applied to the particular tasks [7].

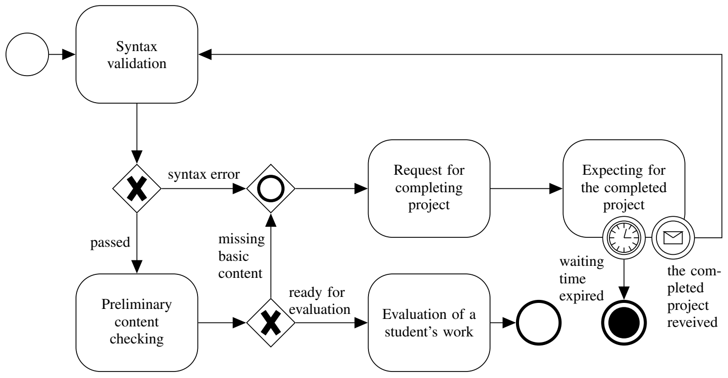

Syntax validation
syntax error
passed
Request for completing project
Expecting for the completed project
missing basic content
waiting time expired
the completed project reveived
Preliminary content checking
ready for evaluation
Evaluation of a student's work

Figure 1. An example of the student's project evaluation process

In the considered example, the process is applied to the website project evaluation. At the beginning, the syntax is automatically checked. Every website code in XHTML needs to be well-formed in terms of the XML standard, and valid wrt the XHTML DTD.

If the project syntax is correct, preliminary content checking is performed. Then, if the project contains expected elementary XHTML tags, it can be evaluated and a grade can be given according to the rules defined by the teacher. On the other hand, if the project contains any syntax error or lacks some basic required content, it is requested to be completed. After receiving the completed project, the whole process starts from the syntax checking again. However, if the completed project is not received on time, the process is terminated (thus, the author of the project does not get a credit).

The example will be used for presentation of our research concerning a hierarchical approach to formal verification. We discuss the existing related works beforehand.

3 Related works

Most of the recent approaches to analysis of the BPMN models consider a restricted subset of BPMN elements in the model. They focus on checking of selected properties of the BPMN model through its transformation to a formal language.

In [8] Raedts et al. presented an approach transforming BPMN models to Petri nets, and these to the mCRL2 algebraic language. This allows for verification of the model using the mCRL2 toolset. Because the discovered problem have to be manually identified in the BPMN model, this can slow the result interpretation process.

Dijkman et al. in [9] proposed a similar approach. They presented a formal specification of the BPMN to Petri nets mapping, and thanks to this, they identified a number of deficiencies in the BPMN specification. The implementation of the approach transforms a BPMN model to a PNML file, which can be used in ProM tool in order to check the model for absence of dead tasks and absence of incomplete process executions. One of the limitations of this approach is not supporting of OR-join gateways.

Similar research conducted by Ou-Yang and Lin [10] proposed a Petri-net-based approach to evaluate the feasibility of a BPMN model. This approach enables to reveal deadlocks and infinite loops. It consists in manually translating of the BPMN model to the Modified BPEL4WS representation, and then to Colored Petri-net XML (CPNXML). The resulted CPNXML representation can be verified using CPN Tools. The major limitations of this research are the limited assessment criteria, and lack of support of the multiple merge and split conditions in BPMN.

Another research direction concerns the translation of BPMN models to Yet Another Workflow Language (YAWL) [11], a modeling language for Business Processes. The BPMN2YAWL tool for such transformation was presented in [12]. Such model can be further checked using a YAWL-based verification tool. The recent research by Wynn et al. [13] presented the verification of YAWL models with advanced constructs, such as cancellations or OR-joins. The paper describes the mapping of a model to an extended Petri net to determine the model correctness, i.e. the following properties are verified: soundness, weak soundness, irreducible cancellation regions, and immutable OR-joins. Although in this research the process is modeled in YAWL, according to the authors, it can be applicable to BPMN as well. However, all of the YAWL approaches consider only BPMN to YAWL transformation. Thus, the errors revealed in the YAWL model can not be easily tracked in the BPMN model.

One of the recent paper in the field of BPMN model verification by Lam [14] proposed a transformation of the BPMN model to New Symbolic Model Verifier (NuSMV) language in order to do a model-checking analysis. The strength of this approach is that it has mathematical foundations and addresses the correctness issue of the transformation. However, this approach assumes a specification of Computation Tree Logic (CTL) formulas, which stipulate the required properties of the model to be checked. Therefore, it is not possible to check automatically a BPMN model of the process which does not have any properties specified using CTL.

The main drawback of these solutions is that it is difficult to map the resulting model back to the BPMN one. Although the tools reveal some errors in the model after translation, it is hard to find the corresponding place in the BPMN model and fix them.

4 BPMN model structure verification

In this section, we present the concept of BPMN model structure (global) verification. For such a verification, the model structure is translated to the Alvis modeling language.

4.1 Alvis modeling language

Alvis [1] combines the advantages of formal methods and practical modeling languages. The main differences between Alvis and more classical formal methods, especially process algebras, are: a user-friendly syntax and a visual modeling language (communication diagrams) for defining communication among agents. The main differences between Alvis and industry programming languages is a possibility of formal verification of Alvis models e.g. using model checking techniques.

The key concept of Alvis is an *agent*, which denotes any distinguished part of the system with a defined identity persisting in time. An Alvis model is a system of agents that usually run concurrently, communicate one with another, compete for shared resources etc. The dependencies among agents are described with three model layers: graphical, code and system one.

The *code layer* defines the behavior of individual agents. Each agent is described with a piece of source code. Agents can be either *active* or *passive*. *Active agents* perform some activities and each of them can be treated as a thread of control in a concurrent or distributed system. *Passive agents* do not perform any individual activity, but provide a mechanism for the mutual exclusion and data synchronization.

The *graphical layer* (communication diagram) defines connections (communication channels) among agents. A communication diagram is a hierarchical graph with nodes representing agents or parts of the model from the lower level. The diagrams allow for combining sets of agents into modules, represented as *hierarchical agents*. Active and hierarchical agents are drawn as rounded boxes while passive ones as rectangles. An agent can communicate with other agents through *ports*, drawn as circles placed at the edges of the corresponding agents. Communication channels are depicted as lines (or broken lines) with arrowheads showing the direction of communication.

From the users point of view, the *system layer* is predefined and only graphical and code layers have to be designed. The system layer is strictly connected with the system architecture and the chosen operating system. Alvis provides a few different system layers. The most universal one is denoted by α^0 and makes Alvis similar to other formal languages. The layer is based on the following assumptions: 1) each active agent has access to its own processor and performs its statements as soon as possible; 2) the scheduler function is called after each statement automatically; 3) in case of conflicts, agents priorities are taken under consideration (if two or more agents with the same highest priority compete for the same resources, the system works indeterministically).

4.2 BPMN to Alvis transformation

To verify the properties of the structure of the whole BPMN model, the model has to be transformed into Alvis model. The transformation procedure starts with preparing the *initial set of agents* which correspond to activities in the BPMN model. In the second stage, the initial set of agents is optimized. An Alvis agent can work like a buffer which collects a signal/value and then sends it to the next agent.

For each identified agent, we define its interface (ports) by considering the set of *surrounding edges* for a given BPMN activity i.e. sequence flows that go to or from the activity; however, if a sequence flow goes from the activity to a gateway, we consider the sequence flow going from the gateway. Each surrounding edge is transformed into a port of the corresponding agent. Moreover, for each port an identifier (a name) has to be added. To complete the agent definition, its behavior should be defined with additional code in the Haskell functional programming language.

Other agents in the Alvis model can be defined in very similar way. Moreover, in the considered example, a student is treated as a part of the system environment in the Alvis model. Thus, a project (its submission or resubmission), a grade and a timeout are sent via border ports, which have to be further specified. To represent the possibility of a timeout, the agent definition should contain the Alvis statement with a time out branch. After receiving an error signal, the agent waits particular time for a revised project, and after this time a time_out signal is generated and the agent finishes its activity.

Although in Alvis a decision table activity can be represented as Haskell function, such an approach is beyond the scope of this paper. The approach considering full Alvis representation of the presented BPMN model was proposed in [4]. In the approach presented in this paper, Alvis is used only as a tool for global verification. For the purpose of local verification, the XTT2 method suits much better, and contrary to Alvis can provide a precise verification of single BPMN elements, such as gateways or tasks.

The last stage of the transformation procedure is to define communication channels in the Alvis model graphical layer, which in most cases consist in connecting pairs of ports. A more complex case is the transformation of the OR gateway, which requires to connect two pairs of ports. The complete communication diagram is shown in Fig. 2.

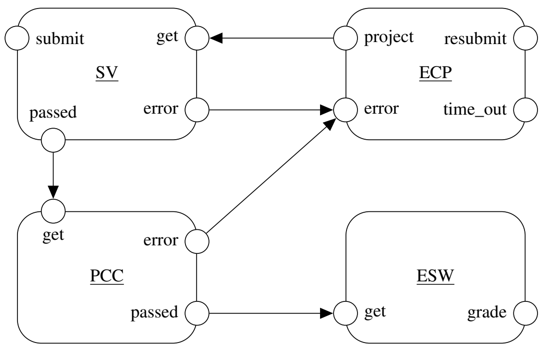


Figure 2. Alvis model – communication diagram

4.3 Verification of the model structure

The transformation of a BPMN model into an Alvis one is only a half-way to the formal verification of the model. Next, the Alvis model is transformed into a *labelled transition system* (LTS), used for formal verification. An LTS graph is an ordered graph with nodes denoting states of the considered system and edges denoting transitions among states. A state of a model is represented as a sequence of agents states. A state of an agent is four-tuple that consists of: agent mode (e.g. running, waiting), its program counter (point out the current step/statement), context information list (contains additional information e.g. the name of called procedure) and a tuple with parameters values.

There are two possible approaches to the formal verification of an LTS graph. If the graph is stored in the form of Haskell list, it is possible to add additional functions that inspect the list e.g. to find states with specified properties. On the other hand, such an LTS graph can be encoded using the *Binary Coded Graphs* (BCG) format and verified with the CADP toolbox.CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking. The verified properties can be divided into two groups usually called *safeness* and *liveness* ones. The former link with states properties while the latter link with an LTS graph paths' properties.

5 BPMN elements verification

Apart from the BPMN model structure analysis, checking several properties of single BPMN elements (local verification) is needed. Thus, our approach allows for verification of selected BPMN elements, which are mapped to the XTT2 knowledge representation. Thanks to the formal representation of XTT2, it is possible to verify several properties of these elements using the HeaRT rule engine [5] with the HalVA tool [6].

5.1 XTT2 rule representation

EXtended Tabular Trees v2 (XTT2) is a knowledge representation that incorporates an attributive table format. In this approach, similar rules are grouped in separated tables, and the system is split into a network of such tables representing the inference flow [15]. The XTT2 structure and rules can be modeled visually using the HQEd (HeKatE Qt Editor) rule editor. This table-based representation can be automatically transformed into HeKatE Meta Representation (HMR) which is suitable for direct execution by the HeKatE RunTime (HeaRT), a dedicated inference engine. HeaRT also provides a verification module – HeKatE Verification and Analysis (HalVA) [16]. The module implements a debugging mechanism that allows tracking system trajectory and logical verification of models. It is important to notice that formal verification is possible thanks to the formalized description in the ALSV(FD) logic of the XTT2 rules.

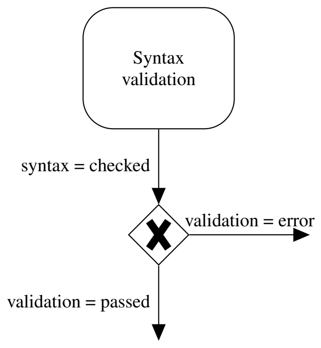
5.2 Gateways verification

Several problems related to selected BPMN elements may be considered. In the case of gateways, it should be checked if all the possible conditions are taken into account during the design. The proposed approach is as follows. A gateway BPMN element is translated to a table XTT2 knowledge representation – in this case it is represented as a single table. Diagram elements are translated to the XTT2 form according to appropriate logic functions. Thus, a BPMN element and its sequence flows are transformed to an XTT2 table filled with proper rules. Similar approach to the analysis of the BPMN elements and corresponding logic functions can be found in [3].

An exemplary decision table corresponding to the XOR gateway from the case study is presented in Fig. 3. The table consist of four columns. The first two, marked with **(?)**, contain condition attributes, and the second two, marked with **(->)**, contain the decision attributes. Each row contains a single rule that specifies the requirements for the flow.

The syntax of the resulting table can be validated in HQEd and then verified using HeaRT with HalVA. In the presented example, assuming that the *validation* attribute can take one of three values: *error*, *passed* or *warning*, the table can be verified against its completeness. It can be observed that the state in which the *validation* attribute takes the *warning* value is not included.

It is important to note that even if all model elements are validated, the whole model structure is still not grasped. Therefore, the verification of the model structure, presented in Section 4, is needed.



(?) syntax	(?) validation	(->) error flow	(->) passed flow
checked	error	1	0
checked	passed	0	1

Figure 3. BPMN gateway verification

5.3 Tasks verification

Since BPMN does not specify the control logic of particular tasks, currently it has to be implemented manually. In the proposed approach it can be specified either using rules in the form of the XTT2 table or network, or as a HeaRT callback.

After specification of the task logic using the XTT2 decision tables, there is a possibility of their formal verification. Currently, HeaRT with the HalVA module allows for verification and analysis of the XTT2 table, i.e.: checking the inconsistency of a single rule, inconsistency of a pair of rules, incompleteness (lack of the ability to react for every admissible input values), subsumption of conditions and subsumption of a pair of rules, as well as identity and equivalence of rules [6].

An exemplary XTT2 decision table for the *Evaluation of a student work* task is shown in Table 1. The table can be used for evaluation a project, according to the specified rules. The output of the table is a grade for a project.

(?) implemented functionality	**(?) quality**	**(->) grade**
= basic	= low	:= satisfactory (D)
= any	= low	:= satisfactory (D)
= basic	= high	:= good (C)
= advanced	= fair	:= very good (B)
= advanced	= high	:= excellent (A)

Table 1. Decision table for student's project evaluation

In the presented table, it can be observed that there is no rule which can determine the *grade* when *implemented functionality* is *basic* and the *quality* of the project is *fair*. Thus, the verification would give the information about uncovered states (incompleteness), as well as it would inform that the second rule subsumes the first one. This is important when the system has to work correctly for any admissible input data and produce deterministic, consistent solutions.

6 Evaluation

The verification method presented in this paper is a new hybrid approach to the BPMN model verification and constitute a preliminary attempt to BPMN model execution.

From the structure point of view, the Alvis model resembles the original BPMN one. After a verification of the Alvis model, it is easy to link the model properties to the properties of original BPMN model. Contrary to the solutions presented in [9,10], our approach supports the OR-join gateway and the multiple merge and split.

From the single element point of view, the approach allows for using rule verification methods. Although this requires to specify gateway conditions using the ALSV(FD) logic and define the logic of tasks using the XTT2 representation, this can be a consistent method, complementary to the Business Processes. Moreover, the transformation from BPMN to XTT2 can be used for execution purposes in the future.

Therefore, the presented approach differs from the earlier attempts in addressing hierarchical verification of BPMN models. It allows for verification of both:

1. model structure (or flow) and
2. single elements (gateways and tasks) of the BPMN model.

In both presented cases, the BPMN elements are taken into account. However, thanks to the separation of layers, the approach provides the verification of distinct properties on different abstraction levels.

In the case of the BPMN model structure, the properties to verify can be divided into two groups: *safeness* and *liveness*. The former is related to states properties e.g. a project with correct content cannot be treated as a defective one. The latter concerns properties of LTS graph paths e.g. if the time out signal has not been generated and a project with correct content has been provide, the system must provide a suitable grade.

When it comes to the BPMN elements, there are many properties which can be verified, such as: lack of *redundancy*, *consistency*, *minimal representation*, or *completeness*.

Although the approach concerns only a small subset of BPMN, extending of this subset is expected in the future. Dedicated tools enabling automatic translation of the BPMN model to Alvis and XTT2 representations are planned to be implemented. Moreover, the formal definition of transformation rules will be developed.

7 Conclusion

The paper presents preliminary results of the research concerning verification of BPMN models. The original contribution is the proposal of a hybrid and hierarchical approach to formal verification of selected BPMN models. We propose an approach which uses the Alvis modeling language for the global verification of the model structure and the XTT2 knowledge representation for the local verification i.e. verification of single BPMN elements in the model. The presentation of the approach has been limited to the presentation of a simple, yet illustrative, case study of a student's project evaluation process. The considered example contains only a few activities, gateways, and events, However, it is possible to use the presented approach for more complex models.

References

1. Szpyrka, M., Matyasik, P., Mrówka, R.: Alvis – modelling language for concurrent systems. In Bouvry, P., Gonzalez-Velez, H., Kołodziej, J., eds.: Intelligent Decision Systems in Large-Scale Distributed Environments. Studies in Computational Intelligence. Springer-Verlag (2011) (in press).
2. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science **20**(1) (2010) 35–53
3. Kluza, K., Maślanka, T., Nalepa, G.J., Ligęza, A.: Representing BPMN diagrams with XTT2-based business rules proposal. In Brazier, F.M., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C., eds.: Intelligent Distributed Computing V. Studies in Computational Intelligence. Springer-Verlag (2011) in press.
4. Szpyrka, M., Nalepa, G.J., Ligęza, A., Kluza, K.: Proposal of formal verification of selected bpmn models with alvis modeling language. In Brazier, F.M., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C., eds.: Intelligent Distributed Computing V. Studies in Computational Intelligence. Springer-Verlag (2011) in press.
5. Nalepa, G.J.: Architecture of the HeaRT hybrid rule engine. In Rutkowski, L., [et al.], eds.: Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II. Volume 6114 of Lecture Notes in Artificial Intelligence., Springer (2010) 598–605
6. Nalepa, G., Bobek, S., Ligęza, A., Kaczor, K.: Halva - rule analysis framework for xtt2 rules. In Bassiliades, N., Governatori, G., Paschke, A., eds.: Rule-Based Reasoning, Programming, and Applications. Volume 6826 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2011) 337–344
7. Nalepa, G.J., Kluza, K., Ernst, S.: Modeling and analysis of business processes with business rules. In Beckmann, J., ed.: Business Process Modeling: Software Engineering, Analysis and Applications. Business Issues, Competition and Entrepreneurship. Nova Publishers (2011)
8. Raedts, I., Petković, M., Usenko, Y.S., van der Werf, J.M., Groote, J.F., Somers, L.: Transformation of BPMN models for Behaviour Analysis. In Augusto, J.C., Barjis, J., Nitsche, U.U., eds.: MSVVEIS, INSTICC press (2007) 126–137
9. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. preprint 7115. Technical report, Queensland University of Technology, Brisbane, Australia (2007)
10. Ou-Yang, C., Lin, Y.D.: BPMN-based business process model feasibility analysis: a petri net approach. International Journal of Production Research **46**(14) (2008) 3763–3781
11. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet another workflow language. Information Systems **30**(4) (2005) 245–275
12. Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L.: Transforming BPMN diagrams into YAWL Nets. In Dumas, M., Reichert, M., Shan, M.C., eds.: Business Process Management. Volume 5240 of Lecture Notes in Computer Science. Springer (2008) 386–389
13. Wynn, M., Verbeek, H., Aalst, W.v.d., Hofstede, A.t., Edmond, D.: Business process verification – finally a reality! Business Process Management Journal **1**(15) (2009) 74–92
14. Lam, V.S.W.: Formal analysis of BPMN models: a NuSMV-based approach. International Journal of Software Engineering and Knowledge Engineering **20**(7) (2010) 987–1023
15. Nalepa, G.J., Ligęza, A., Kaczor, K., Furmańska, W.T.: HeKatE rule runtime and design framework. In Giurca, A., Nalepa, G.J., Wagner, G., eds.: Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009) Cottbus, Germany, September 21, 2009, Cottbus, Germany (2009) 21–30
16. Ligęza, A., Nalepa, G.J.: Rules verification and validation. In Giurca, A., Gasevic, D., Taveter, K., eds.: Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. IGI Global, Hershey, New York (2009) 273–301