Proposal of Formal Verification of Selected BPMN Models with Alvis Modeling Language*

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

Abstract. BPMN is a leading visual notation for modeling business processes. Although there is many tools that allows for modeling using BPMN, they mostly do not support formal verification of models. The Alvis language was developed for modeling and verification of embedded systems. However, it is suitable for the modeling of any information systems with parallel subsystems. The goal of this paper is to describe the concept of using Alvis for a formal verification of selected BPMN models. In the paper a translation from BPMN to Alvis model is proposed. The translation is discussed and evaluated using a simple yet illustrative example.

1 Introduction

Modeling of business process can be considered on two levels. The first one concerns the visual specification with a modeling notation e.g. BPMN. The second one is related to the *correctness* of the process model. This general feature is of great importance in case of safety critical systems. The analysis of the formal aspects of the model opens up possibility of optimization of the process model.

The goal of the paper is to describe the possibility of using Alvis for a formal verification of BPMN models. Alvis [6] has been originally defined the modeling and verification of embedded systems, but it is also suitable for the modeling of any information systems with subsystems working in parallel. One of the main advantages of this approach is the similarity between BPMN and Alvis models. The Alvis model resembles the original BPMN one from the graph structure point of view.

Marcin Szpyrka · Grzegorz J. Nalepa · Antoni Ligęza · Krzysztof Kluza
AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: {mszpyrka,gjn,ligeza,kluza}@agh.edu.pl

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

F.M.T. Brazier et al. (Eds.): Intelligent Distributed Computing V, SCI 382, pp. 249–255.
springerlink.com

Thus, after a verification of the Alvis model, it is easy to link the model properties to the properties of original BPMN model.

The paper is organized as follows. Section 2 provides a short presentation of BPMN and Alvis modeling languages. It presents a BPMN model of the student's project evaluation process. Selected Alvis features that are essential from the considered problem point of view are described as well. The transformation method from BPMN to Alvis is described in Section 3. We have limited the presentation to describing a simple case study. However, it is possible to use this approach with more complex models. A short summary is given in the final section.

2 BPMN and Alvis Modeling Languages

Business process [7] can be defined as a collection of related, structured tasks that produce a specific service or product (serve a particular goal) for a particular customer. Business Process Model and Notation (BPMN) [5], is a visual notation for modeling business processes. The notation uses a set of predefined graphical elements to depict a business process and how it is performed. For the purpose of this research, only a subset of BPMN elements is considered i.e. elements used to model orchestration business processes, such as flow objects (*events*, *activities*, and *gateways*) and connecting objects (*sequence flows*). The model defines the ways in which individual tasks are carried out. Gateways are to determine forking and merging of the sequence flow between tasks depending on some conditions. Events denotes something that happens in the process. The icon within the event circle denotes the event type, e.g. envelope for *message event*, clock for *time event*.

Let us analyze a BPMN use case describing a student project evaluation process. The diagram shown in Fig. 1 depicts the evaluation process of a student's project for the Internet technologies course. The process is applied to the website evaluation. At the beginning, the syntax is automatically checked. Every website code in XHTML needs to be a *well-formed* XML and *valid* w.r.t. XHTML DTD.

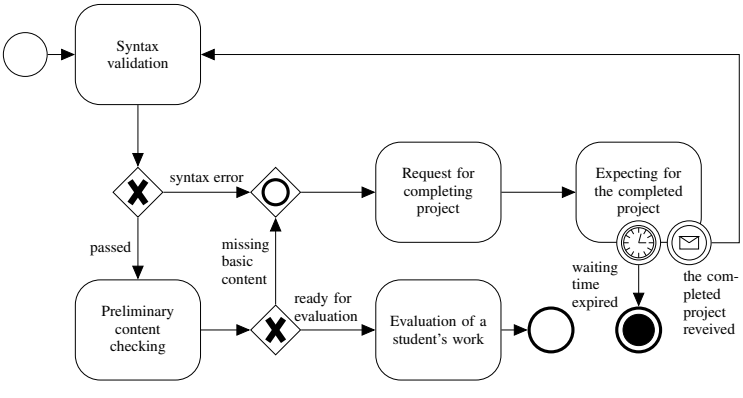

Fig. 1 An example of the student's project evaluation process

If the syntax of the project file is correct, preliminary content checking is performed. Then, if the project contains expected elementary tags (e.g. at least several headings, an image and a table), it can be evaluated and a grade can be given according to some specified evaluation rules [3]. 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 key concept of Alvis [6] is an *agent* that denotes any distinguished part of the system under consideration 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. To describe all dependences among agents Alvis uses three model layers: graphical, code and system one. The *code layer* is used to define the behavior of individual agents. Each agent is describe with a piece of source code implemented in Alvis Code Language (AlvisCL) [6]. From the code layer point of view, agents are divided into *active* and *passive* ones. *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) is used to define interconnections (communication channels) among agents. A communication diagram is a hierarchical graph whose nodes may represent both kinds of agents (*active* or *passive*) and parts of the model from the lower level. From users point of view, the *system layer* is predefined and only graphical and code layers must be designed. Alvis provides a few different system layers. The most universal one is denote by α^0 and makes Alvis similar to other formal languages. The layer is based on the following assumptions: each active agent has access to its own processor and performs its statements as soon as possible; the scheduler function is called after each statement automatically; in case of conflicts, agents priorities are taken under consideration.

3 BPMN to Alvis Transformation

The transformation procedure starts with preparing the *initial set of agents*. Initially, each activity is treated as a potential active agent in the corresponding Alvis model. Because the names of agents in Alvis are programming identifiers that must start with an upper-case letter, the initial set of agents is as follows: {*SV, PCC, RCP, ESW, ECP*}, where *SV* stands for *Syntax validation*, etc. In the second stage, the initial set of agents is optimized. Consider the *Request for completing project* activity: From the data flow point of view, the corresponding Alvis agent works like a buffer that collects a signal/value and then sends it to the next agent.

The agent realizes three steps (entering the loop, *in* and *out* statements) in every cycle of its activity. These steps do not provide any essential information from the verification point of view, but influence the state space size. Without losing any possibility of the Alvis model verification, we can join the *RCP* and *ECP* agents into one (*ECP* name has been chosen for the agent that represents these two activities).

For each agent identified in the previous stage we have to define its interface i.e. the set of ports. To do this, we consider the set of *surrounding edges* for a given BPMN activity. *Surrounding edges* are these ones that go to or from the activity, but if an edge goes from the activity to a gateway, instead of the edge we consider edge going from the gateway. Each surrounding edge is transformed into a port of the corresponding agent. Moreover, we have to add names (identifiers) for each port. The result of this step for the *Syntax validation* activity is shown in Fig. 2. The edge drawn with dashed line is not a surrounding edge for the considered activity.

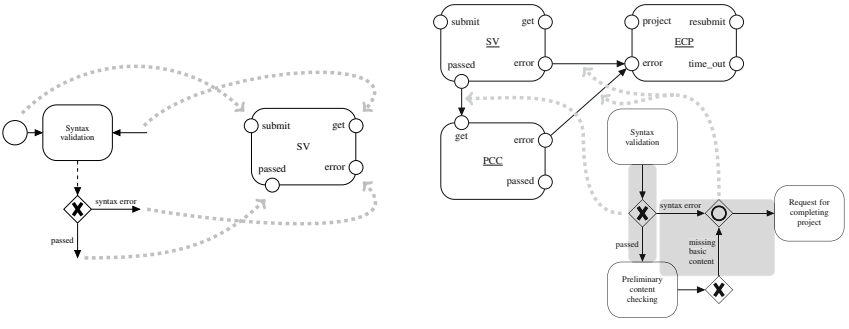

Fig. 2 Transformation of the *Syntax validation* activity to the *SV* agent (left), Generation of communication channels (right)

To complete the *SV* agent definition it is necessary to define its behavior:

```
data Project =  None | Defective | CorrectSyntax
             | CorrectContent String String deriving(Eq,Show);
data Grade = Grade Char deriving(Show);}
-- ...
agent SV {
  pr :: Project = None;
  in submit pr;
  loop {
    if(pr == Defective) { out error; }
    else { out passed pr; }
    pr = None;
    in get pr; }
```

It should be underlined that the BPMN XOR gateway is represented here by the *if else* statement. The *SV* agent: waits for a project to evaluate; collects it via the *submit* port; uses the *pr* parameter to store the project; if the project is *defective*, sends a signal via the *error* port, otherwise sends the project via the *passed* port; waits for revised project (if necessary).

Other agents in the Alvis model are defined in very similar way. Let us focus only on the most interesting features of the code layer. In the model, a student is treated as a part of the system environment. Thus, a project (submission or resubmission), a grade and a timeout are sent via border ports. The ports are specified as follows:

```
in submit   Project [0]   durable;         out time_out [] [];
in resubmit Project [1..] signal durable; out grade Grade [];
```

It means that a value of the *Project* data type can be sent once (at the beginning) via the *submit* port and may be sent (it's not necessary) via the *resubmit* port every one day. In both cases, if a *Project* value is provided by the environment, it waits for serving. A signal (without any particular value) can be sent any time via the *time_out* port and a value of the *Grade* data type may be sent via the *grade* port.

The last stage of the transformation procedure is to define communication channels in the Alvis model graphical layer. In most cases it is easy to point out pairs of ports that should be connected. In the considered example, the most interesting is the transformation of the OR gateway (see Fig. 2). It is necessary to connect two ports (*SV.error* and *PCC.error*) with the *ECP.error* port.

The transformation of a BPMN model into an Alvis one is only a half-way to the formal verification of the BPMN model. Next, the Alvis model is transformed into a *Labelled Transition System* (LTS) that is used for a formal verification. An LTS graph is an ordered graph with nodes representing states of the considered system and edges representing transitions among states. A state of a model is represented as a sequence of agents states. A state of an agent is four-tuple: agent mode (e.g. running, waiting), its program counter (point out the current step/statement), context information list (additional information)and a tuple with parameters values (see [6]). The initial state for the considered system is:

```
SV:  (running,1,[],-)
PCC: (running,1,[],-)
ECP: (running,1,[],-)
ESW: (running,1,[],(-,-))
*:   submit/0, resubmit/1
```

It means that all agents are *running* and are about to execute their first steps. The last line means that a value will provided from the environment to the *submit* port immediately, and in 1 day a value via the *resubmit* port may be provided.

An LTS graph is verified with the CADP toolbox [1]. CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking. Let us focus on the considered model safeness properties. The LTS graph contains 1438 nodes. Many of them are results of the Alvis language specificity. For example, Alvis distinguish the states before and after entering a loop that do not correspond to different *states* of the BPMN model. From the BPMN model point of view, an analysis of the corresponding Alvis model deadlocks is very important. Deadlocks are state without any transition going from them. Moreover, states that represent a situation when the environment generates signals but the Alvis model does not respond to them are also treated as deadlocks. The LTS graph for the model contains only one state without a transition going from it:

```
SV:  (W,7,[in(get)],-)
PCC: (W,2,[in(get)],-)
ECP: (F,0,[],-)
ESW: (W,1,[in(get)],(-,-))
```

It represents the situation after the *ECP* agent time out. Moreover, the LTS graph contains 4 other deadlocks that represent situations after grading a project. From the LTS graph analysis, we can also conclude that our system always provide a grade for a project with correct content, and that for any received project, the system provide a grade or a request for a revised project.

4 Conclusion and Future Work

Practical analysis of business process models is needed in case of many systems, e.g. control or embedded systems. The analysis of the formal aspects of the model allows for optimization of the model. To perform it, translations of the process model to some formal specification can be considered. Alvis language was developed for modeling and verification of embedded systems. It is suitable for the modeling of any information systems with parallel subsystems. The goal of this paper is to describe the concept of using Alvis for a formal verification of selected BPMN models.

In the paper, a proposal of the translation from BPMN to Alvis model is discussed and evaluated using an example. The transformation of a BPMN model into an Alvis model allows for formal verification. The Alvis model is transformed into a Labelled Transition System that is used for a formal verification. There are two possible approaches to a formal verification of an LTS graph. It can be also encoded using the Binary Coded Graphs format and verified with the CADP toolbox.

As future work, a heterogeneous verification approach is considered. In this case the XTT2 rule language [4] is proposed to model selected activities. XTT2 rules (and tables) can be formally analyzed using the so-called verification HalVA framework [2]. In this approach table-level verification would be performed with HalVA and the global verification would be provided translation to Alvis model.

References

1. Garavel, H., Mateescu, R., Lang, F., Serwe, W.: CADP 2006: A toolbox for the construction and analysis of distributed processes. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 158–163. Springer, Heidelberg (2007)
2. Nalepa, G.J., Bobek, S., Ligęza, A., Kaczor, K.: HalVA – rule analysis framework for XTT2 rules. In: Bassiliades, N., Governatori, G., Pasckhe, A. (eds.) RuleML2011 - International Symposium on Rules, Lecture Notes in Computer Science, Springer, Heidelberg (accepted for publication 2011)
3. 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 Science Publishers (to be published, 2011)
4. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science 20(1), 35–53 (2010)

5. OMG: Business Process Model and Notation (BPMN): Ftf beta 1 for version 2.0 specification. Tech. Rep. dtc/2009-08-14, Object Management Group (2009)
6. Szpyrka, M., Matyasik, P., Mrówka, R.: Alvis – modelling language for concurrent systems. In: Bouvry, P., González-Vélez, H., Kołodziej, J. (eds.) Intelligent Decision Systems in Large-Scale Distributed Environments. SCI, vol. 362, pp. 315–341. Springer, Heidelberg (2011)
7. White, S.A., Miers, D.: BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc., Lighthouse Point (2008)