Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1427–1436

DOI: 10.15439/2015F44
ACSIS, Vol. 5

Efficiency of formal verification of ArchiMate business processes with NuSMV model checker

Piotr Szwed
AGH University of Science and Technology
E-mail: pszwed@agh.edu.pl

***Abstract*—We investigate an application of model checking techniques to automated verification of business processes expressed in ArchiMate language. As a verification tool the state of the art symbolic model checker NuSMV is used. The proposed approach consists in fully automated translation of behavioral elements embedded in ArchiMate models into a corresponding representation in NuSMV language and then verifying its properties specified in CTL. Since our goal is to build an interactive verification tool, we focus on time efficiency of the verification process. We report results of tests performed on artificial process models of various complexity, as well as on a real business process example. The results show, that the described approach can be applied successfully, however, verification of complex business process specifications may face the problem of state space explosion. In such a case, to make the verification feasible, various reductions and simplifications can be applied.**

***Index Terms*—ArchiMate, business process verification, model checking, NuSMV**

I. Introduction

THE GOAL of business process verification is to check if processes intended to be used or already implemented within an organization exhibit desired behavioral properties. The analyzed models may represent combinations of manual tasks performed by employees with operations supported by IT tools, as well as fully automated specifications run by process execution engines. In both cases process verification is appreciated by business organizations, as it can detect potential errors, design flaws and ambiguities.

Although graphical process modeling tools offer support for local syntax checking, e.g. correct use of links between elements of the diagram, some structural errors remain undetected, especially those resulting from incorrect use of synchronization mechanisms [1]. Partial analysis of model behavior can be performed by simulation techniques, however, only application of formal methods can give unequivocal answer that the verified system meets formally specified requirements.

Obviously, the landscape of business process modeling notations is dominated by such languages, as BPMN [2] or EPC [3], [4]. In this work, however, we decided to focus on verification of process models defined in ArchiMate, a contemporary, open and independent language intended for description of enterprise architectures [5]. Although ArchiMate comprises a variety of elements intended to model important aspects of enterprise architectures, constructs allowing to model behavior can be found only in the *Business* layer. They include events, processes (also understood as activities), interactions, collaborations and several types of junctions.

Formal system verification can by done either by deductive reasoning or model checking [6]. Deductive reasoning consists in formulating theorems specifying desired system properties and proving or falsifying them using manual or automated techniques. However, deductive reasoning methods gives very little information on causes, if the verified property does not hold.

Model checking allows to verify a concurrent system modeled as a finite state transition graph against a set of specifications expressed in a propositional temporal logic. It employs efficient internal representations and quick search procedures to determine automatically, whether the specifications are satisfied along the computational paths. Moreover, if a specification is not met, the verification procedure delivers a counterexample that can be used to analyze the source of the error. The main problem faced by model checking is state space explosion [7]. At the very beginning only small examples could have been processed. A significant progress in this technique was achieved with application of ordered binary decision diagrams (OBBD) [8] allowing to model systems consisting of millions of states and transitions.

Although formal tools reached the state of the art, they are not commonly used in the engineering practice. According to Huuck [9] three factors decide on successful application of formal tools: they should be simple to use, the time spent on model preparation and verification should be comparable with other user activities, and, finally, a tool should provide a real value, i.e. deliver information that was previously not available.

We were motivated by an idea of developing a software tool that fully automatically translates behavioral elements of a business model expressed in ArchiMate language to a corresponding finite-state graph required by the model checker. Then, after running the verification and detecting errors, valuable information about specifications not met and counterexamples can be returned to the process designer.

The concept of verification system is presented in Fig. 1. The business model is defined within Archi [10], a well known ArchiMate modeling tool.

As a verification platform the state of the art symbolic model checker NuSMV [11] is used. NuSMV allows to enter a model comprising a number of communicating finite state machines (FSM) and automatically checks its properties specified as Computational Tree Logic (CTL) or Linear Temporal Logic (LTL) formulas. We have developed an Archi plugin that

978-83-60810-66-8/$25.00

extracts a subgraph of ArchiMate behavioral elements and transforms it into NuSMV model descriptions. Specifications of desired properties are defined by a process designer, however, a part of them is generated automatically by an analysis of the process structure.

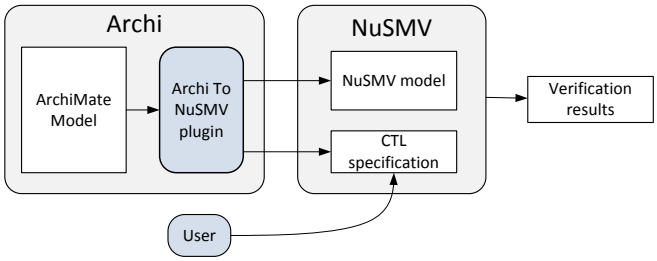

Fig. 1. The concept of the verification system

This paper is a continuation of our previous work [12], where an initial concept of a model checking verification system dedicated to ArchiMate models was described. Since our goal is to build an interactive verification tool, the main concern in this work is the *time efficiency* of the verification process. In order to assess it, we performed tests a for set of artificial process models of various complexity, as well as we verified a real business process example. In all cases we collected information related to the size of state space and processing time.

The paper is organized as follows: next Section II discusses various approaches to the verification of business models. Section III presents basic concepts of ArchiMate language. It is followed by Section IV, which describes details of the NuSMV model generation procedure. Section V reports results of time efficiency tests. Section VI provides concluding remarks.

II. Related works

Application of formal methods to verification of business processes was surveyed by Morimoto [1]. Author distinguished three prevalent approaches: based on automata, Petri nets and process algebras. The first approach consists in translating the process description into a set of communicating automata (state machines) and performing model checking with such tools, as SPIN [13] or UPPAAL [14]. In analysis of Petri net models, basically simulation techniques are used, especially in the case of more expressive colored Petri nets.

Model checking has an established position in verification of business processes. It was applied in [15] to BPMN models extended with temporal and resource constraints. In [16] verification of of e-business processes was achieved by translation to CSP language and checking refinement between two specifications. In [17] authors implemented a system that translated BPEL specification into NuSMV language, what allowed them to check properties defined as CTL formulas. Three types of correctness properties were analyzed: invariants, properties of final states and temporal relations between activities. The first two can be classified as *safeness*, the last as the *liveness* property. Similarly, in work by Fu et al. [18] CTL was applied to the verification of e-services and workflows with both bounded and unbounded numbers of process instances. The work [19] discusses verification of data-centric business processes. The correctness problem was expressed in the LTL-FO, an extension to the Linear Temporal Logic, in which propositions were replaced by First Order statements about data objects.

Wynn et al. [20] discuss verification tools developed for models in YAWL language [21], with a special focus on OR-joins and cancellation constructs. The verified process properties are predefined. e.g. a soundness property is a combination of three conditions: a process should always complete, after the completion all its subprocesses should be inactive and every its part should be executable. Verification algorithms for YAWL are closely related to those of Petri nets, i.e. analyzing reachability graphs and in hard cases applying Petri nets reduction techniques.

In our previous works [22], [23] we proposed a method for verification of ArchiMate behavioral specifications based on deductive reasoning. The described approach consisted in transforming ArchiMate model into a set of LTL formulas, then extending it with formulas defining desired system properties and formally proving them using semantic tableaux method.

Although verification of business processes have been investigated for at least 15 years, surprisingly, there is very little information provided about efficiency of applied techniques. This in particular concerns the formal verification with the model checking approach. However, quantified results for a quite complex process specified in YAWL are given in the work [20] entitled remarkably "Business Process Verification - Finally a Reality!". Without reductions the soundness verification of the process stages took about a few dozen seconds; applying reductions decreased the verification time of the whole process below ten seconds.

NuSMV [11] is a state of the art model checker that has been succesfully used for various verification tasks including formal protocol analysis [24], verification of requirements specification [25] or planning tasks [26]. The package uses a special language (named also NuSMV) to define the verified model as a set of linked finite state machines, as well as its specification in form of temporal logic formulas. The model submitted to the verification tool must be manually coded in NuSMV language or generated from another language amenable to state transition system, e.g a state charts [27] or reachability graphs of Petri nets [28].

III. ArchiMate

ArchiMate [5] is a contemporary, open and independent language intended for description of enterprise architectures. The definition of ArchiMate has been accompanied by an assumption, that in order to build an expressive business model, it is necessary to use the relationships between completely different areas, starting from business motivation to business processes, services and infrastructure.

The language comprises five main modeling layers shortly characterized below.

- *Business* layer includes business processes and objects, functions, events, roles and services.
- *Application* layer contains components, interfaces, application services and data objects.
- *Technology* layer gathers such elements as artifacts, nodes, software, devices, communication channels and networks.
- *Motivation* layer allow to express business drivers, goals, requirements and principles.
- *Implementation&Migration* layer contains such elements, as work package, deliverable and gap.

ArchiMate allows to present the architecture in the form of views, which, depending on the needs, can include only items from one layer or can show vertical relations between elements belonging to different layers, e.g.: a relationship between a business process and a function of the component software.

ArchiMate provides a small set of constructs that can be used to model behavior. It includes *Business Processes*, *Functions*, *Interactions*, *Events* and various connectors (*Junctions*), which can be attributed with a logical operator specifying, how inputs should be combined or output produced. According to language specification casual or temporal relationships between behavioral elements are expressed with use of *triggering* relation. On the other hand, ArchiMate models frequently use *composition* and *aggregation* relations, e.g. to show that a process is built from smaller behavioral elements (subprocesses or functions).

In should be also noted that *Business Activity* present in ArchiMate 1.0 specification was removed in version 2.0. Instead, an atomic process should be used.

Although the set of behavioral elements seems to be very limited when compared with BPMN [2], after adopting a certain modeling convention its expressiveness can be similar [29]. An advantage of the language is that in allows to comprise in a single model a broad context of business processes including roles, services, processed business objects and elements of lower layers responsible for implementation and deployment.

Another process modeling notation that can be almost directly mapped on ArchiMate constructs is *Event-driven Process Chain* (EPC) [3], [4]. All behavioral elements of both languages are exactly the same: events, functions (or processes in ArchiMate) and various joins and splits (XOR, OR and AND).

IV. Model generation

This section discusses language patterns that can be used to model ArchiMate elements in NuSMV, as well as details of the translation procedure.

A. ArchiMate model

The internal structure of an ArchiMate model constitutes a graph of nodes linked by directed edges. Both nodes and edges are attributed with information indicating the type of element or relation. While generating NuSMV code describing behavioral aspects of ArchiMate model, we focus on components of the *Business layer*: processes (interactions, functions), events and various junctions.

It should be noted that ArchiMate behavioral constructs have no precisely defined semantics. In fact, translation from ArchiMate specification to NuSMV assigns a semantics, which, although arbitrarily selected, follows a certain intuition, e.g. how to interpret an activity or an event.

Definition 1 (ArchiMate model)**.** ArchiMate model AM is a tuple $\langle V, E, C, R, v, e\rangle$, where

- V is a set of vertices,
- $E \subset V \times V$ is a set of edges,
- C is a set of ArchiMate element types,
- R is a set of relations,
- $vt\colon V \to C$ is a function that assigns element types to graph vertices
- $et\colon E \to R$ assigns relation types to edges.

As we focus on business layer elements that are used to specify behavior, it is assumed that $C =$\{*Process*, *Function*, *Interaction*, *Event*,*Junction*, *AndJunction*, *OrJunction*, *Other*\} and $R =$\{*triggering*, *association*, *composition*, *other*\}.

We will discuss the procedure of NuSMV model generation on a small process example presented in Fig. 2. The whole process is activated upon occurrence of the event *Start*. Then the subprocess *P1* is launched, which is followed by *P2*. If *P2* terminates correctly, a decision is made whether additional subproces *P3* should be executed or the control flow leads to event *End* directly. However, execution of *P2* can be interrupted by the event *Interrupt*, which redirects back to the process *P1*.

B. NuSMV model

The basic structural unit in NuSMV language is a *module* understood as a set of variables and statements that assign initial values to variables and define a transition relation. Depending on the module definition, we may distinguish input variables corresponding to stimuli, internal state variables and output variables (actions).

Definition of a module introduces a new type that can be instantiated. Hence, it is possible to declare a variable of a module type and bind it during declaration resembling a constructor call to a number of input variables. Subsequent variables definitions may reference outputs of other modules instances as their inputs. This allows to define a system of communicating state machines of desired complexity, which propagates input stimuli to its components causing subsequent state changes and generation of output signals. Typically, the model integration is achieved within the special *main* module, however, it can be distributed among lower level modules, which are referenced from *main*.

Fig. 3 shows the structure of NuSMV model corresponding to the process in Fig. 2. Although the structure of the presented process is clearly sequential, it is realized by a number of concurrent state machines (modules) linked by their output

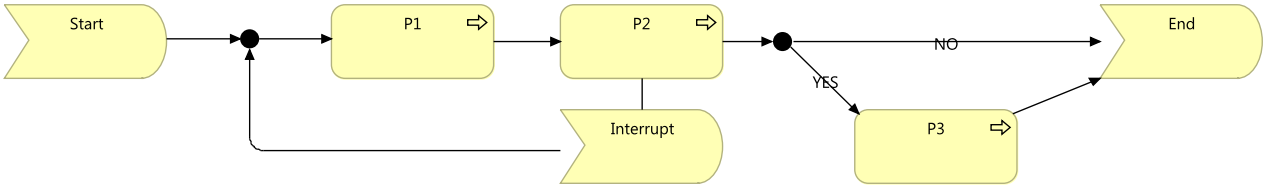

Fig. 2. Sample ArchiMate process specification

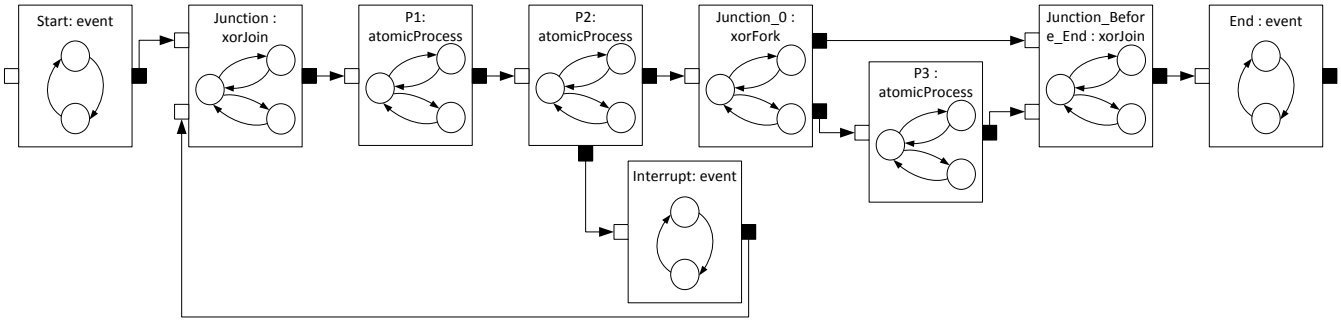

Fig. 3. Linked NuSMV modules used to model the process in Fig. 2

and input variables. The reusable modules correspond to the language constructs: processes, events, forks, joins, etc.

After conducting an analysis of components used to describe ArchiMate processes the following basic modules were identified and implemented:

- $atomicProcess_n$: n-ary atomic process has exactly one input, one primary output and n additional outputs, which can be activated if one of n exceptions occurs. The exception should be modeled in ArchiMate as an event linked with the process by the association relation.
- $event$: has only one input and one output (a boolean flag). Multiple recipients may use this flag as trigger.
- $andFork$: used to model AndJunction in Archmate. The module construction is analogous to event.
- $andJoin_n$: n-ary andJoin produces output signal, if all n inputs are set to TRUE.
- $xorFork_n$: n-ary xorFork have one input and n outputs. Upon module activation, only one among possible outputs will be triggered.
- $xorJoin_n$: n-ary xorJoin has n inputs and sets the output flag if any of them is set. Moreover it tracks the number of inputs, e.g. if two from n inputs are activated, the output flag will be set twice.

Fig. 4 shows the state diagram of the module `atomicProcess1`. The number "1" appearing in the module name indicates the number of additional outputs, which can be set as a result of exception occurrence. The process is activated by the input signal *trigger*. Upon the signal arrival it changes the state from *idle* to *started*. Then a choice can be made between the states *finished* and *interrrupted1*. Synchronously, the corresponding output variable is set: either *outflag* or *exccptflag1* to *TRUE*. The output variable, whichever is set, will be cleared during the transition to *idle* state.

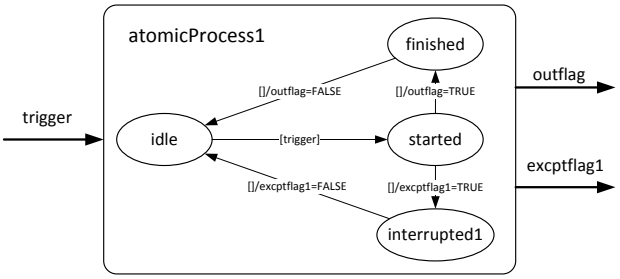

Fig. 4. State machine modeling an atomic process

The NuSMV code for the module is given in Fig. 5. It should be mentioned, that in the case of a process having n exceptional outputs, we generate module `atomicProcess_n` with states *interrupted1*,. . . , *interrruptedn* and n output flags *exceptflag1*,. . . , *exceptflagn*.

Basically, all modules corresponding to ArhiMate language elements were implemented as state machines, which receive input(s), change their internal states and produce outputs. However, due to performance issues we provided also alternative synchronous implementations, which immediately compute output values based on inputs. An example of synchronous process implementation is given in Fig. 6. Its definition uses an invariant that restricts acceptable combinations of variables instead of a transition relation. Basically, synchronous implementations allow to reduce interleaving, what makes the internal model representation in NuSMV smaller and speeds up the verification process. However, synchronous implementations are rather intended to be used

```
MODULE atomicProcess1(trigger)
VAR
   state : {idle,started,finished,interrupted1};
   outflag : boolean;
   excptflag1 : boolean;
ASSIGN
   init(state) := idle;
   next(state) :=
      case
         state = idle & trigger: {started};
         state = started : {finished,interrupted1};
         state = finished & !outflag : idle;
         state = interrupted1 & !excptflag1 : idle;
         TRUE : state;
      esac;
   init(outflag) := FALSE;
   next(outflag) :=
      case
         state = finished : TRUE;
         state = idle : FALSE;
         TRUE : outflag;
      esac;
   init(excptflag1) := FALSE;
   next(excptflag1) :=
      case
         state = interrupted1 : TRUE;
         state = idle : FALSE;
         TRUE : excptflag1;
      esac;
SPEC
   AG (trigger = TRUE -> AF (outflag = TRUE | excptflag1 = TRUE))
```

Fig. 5. NuSMV code of the module `atomicProcess1` (the number 1 indicates number of exceptional outputs)

for such elements as business events, forks or joins, than processes, for which activation and termination should be modeled as two distinguishable events separated by time.

```
MODULE atomicProcessSynchro1(trigger)
VAR
   outflag : boolean;
   excptflag1 : boolean;
INVAR
   (!trigger & !outflag & !excptflag1) |
   (trigger & outflag & !excptflag1) |
   (trigger & !outflag & excptflag1)
```

Fig. 6. Synchronous process implementation, a simplification of the state machine in Fig. 4

It should be mentioned that in the generated code is compatible with with the input language of nuXmv [30], the NuSMV succesor released at the end of 2014. In particular, the `process` keyword is not used and the choice between synchronous and asynchronous execution of model elements is entirely done within the generated model.

C. Generation procedure

The generation procedure consists of the following stages:

1) *Refactoring.* With relation to the numbers of inputs and outputs, it is expected that elements fall into one of two classes: $1 : m$ (one input and m outputs) or $n : 1$ (m inputs and one output). Hence elements with the arity $n : m$ are replaced by two two elements: the first is an appropriate xorJoin or andJoin of arity $n : 1$. The second is an atomic process, event or fork of arity $1 : n$.
2) *Assigning representation.* For each ArchiMate element an appropriate NuSMV module type is selected and configured based on element type and numbers of inputs/outputs. Only required modules are generated, e.g. if the specification uses only processes with one and three exceptional outputs, only modules defining `atomicProcess1()` and `atomicProcess3()` will be generated.
3) *Main module generation.* This step comprises declaration of variables and linking them. For roots (modules without inputs) appropriate initial variables and transitions are added as well.
4) *Generation of specification.* The implemented procedure analyses the graph of elements and generates CTL specifications. See Section IV-D.

The generated NuSMV code for the *main* module is presented in Fig. 7. It can be noticed that variables definition are unordered and the code contains forward references, e.g. the output variable `Junction.output` is referenced before `P1` definition. The event *Stop* has two inputs. As the result of model refactoring an *OrJunction* (variable `Junction_Before_End`) was introduced into the model. For the event *Start* constituting a root element, the boolean variable `Start_trigger` with corresponding transition was added. The initial value of `Start_trigger` is FALSE, and the next value is chosen from {FALSE,TRUE}.

```
MODULE main
VAR
   P1 : atomicProcess(Junction.output);
   P2 : atomicProcess1(P1.outflag);
   P3 : atomicProcess(Junction_0.outflag2);
   Start : event(Start_trigger);
   End : event(Junction_Before_End.output);
   Junction : xorJoin(Interrupt.outflag,Start.outflag);
   Junction_0 : xorFork(P2.outflag);
   Interrupt : event(P2.excptflag1);
   Junction_Before_End : xorJoin(P3.outflag,Junction_0.outflag1);
   Start_trigger : boolean;
ASSIGN
   init(Start_trigger) := FALSE;
   next(Start_trigger) := {FALSE,TRUE};
```

Fig. 7. Generated NuSMV main module code for the process in Fig. 2

D. Generation of specification

As a specification language we use CTL, which allows to formulate properties applying to a tree of computations (paths) starting from a given state. As the tree defines a set of imaginable futures, CTL is called the branching time logic. CTL formulas are combinations of two types of operators *path quantifiers* and *linear-time operators*.

The path quantifiers are:

- $A\,p$ – p holds for every path in a tree and
- $E\,p$ – there exists a path in a tree, for which p holds

Temporal operators include:

- $F\,p$ – p holds true sometime in the future,

- $G\,p$ – p holds true globally in the future,
- $X\,p$ – p holds true next time and
- $p\,U\,q$ - p holds true until q holds true.

Usually a specification formally describing requirements is entered by a user. However, we tried to derive some *liveness* requirements based on control flows within ArchiMate model (see Definition 1).

The implemented procedure generating a set of specifications comprises four steps:

1) Build a set of paths $\Pi = \{\pi_i\}$ within the ArchiMate model,
2) Restrict elements in π_i to events only (elements from the set Evt)
3) Build a partial mapping $R\colon Evt \to 2^{Evt}$
4) Generate the specification for each pair $(e_i, R(e_i))$ in R

In the first step (1) a depth-first search starting from *roots* (ArchiMate elements having no predecessors) is performed. It returns a set of paths $\Pi = \{\pi_i\}$ comprising ArchiMate elements linked by control flow relation. For a path $\pi_i = (e_{ib}, \ldots, e_{ie})$, its last element e_{ie} is either a final element in the model (without successors) or a branching element (already present in π_i). The set of obtained paths reflects only topological relations within the process model. The procedure does not attempt to interpret the model according to any behavioral semantics. This is left to the verification tool.

For the example presented in Fig. 2 the set of paths Π comprises three elements:
π_1 = (`Start, Junction, P1, P2, Junction_0, Junction_Before_End, End`)
π_2 = (`Start, Junction, P1, P2, Junction_0, P3, Junction_Before_End, End`)
π_3 = (`Start, Junction, P1, P2, Interrupt, Junction`)

Any requirements specification must reference terms, in which the model is expressed. We decided to focus on elements of *Event* type, which in business process definitions are typically used to mark important process states (e.g. initial, final and intermediate events).

In the step (2) the paths from Π are restricted to ArchiMate elements being events. For the discussed example the restricted set of paths Π^r={(`Start, End`), (`Start, Interrupt`)}.

In the next step (3) a partial mapping $R\colon Evt \to 2^{Evt}$ is built. The mapping R assigns all (potentially) reachable events to first events appearing in paths from Π. Continuing the example from Fig. 2, the mapping R contains only one pair: (`Start`, {`End`,`Interrupt`}).

Finally, in the step (4) for each event $e \in \operatorname{dom} R$, a pair $(e, R(e))$ is converted into a set of specifications taking the form of (1), where $\mathcal{G} = \{AG, EG\}$, $\mathcal{F} = \{AF, EF\}$ and $\mathcal{O} = \{\bigvee, \bigwedge\}$.

$$\mathcal{G}((f \to \mathcal{F}(\underset{l_i \in R(f)}{\mathcal{O}} l_i))) \tag{1}$$

Fig. 8 gives specifications generated for the process from Fig. 2. The specification `AG (Start.outflag -> EF (Interrupt.outflag | End.outflag)).` is equivalent to the statement: *for every path, starting with Start event, it is possible to reach a state, where End or Interrrupt events occur*. This requirement is obviously true for the discussed example. On the other hand specifications, where conjunction of reachable events occurs are false. An example false specification is `AG( Start.outflag -> AF(Interrupt.outflag & End.outflag))`, what was justified by a counterexample trace comprising 20 elements produced by NuSMV. The types of generated specification are controlled by program parameters. In particular, generation of specifications using conjunctions of reachable events can be switched off.

```
SPEC
    AG( Start.outflag
    -> AF(Interrupt.outflag & End.outflag))
SPEC
    AG( Start.outflag
    -> AF( Interrupt.outflag | End.outflag))
SPEC
    AG( Start.outflag
    -> EF( Interrupt.outflag & End.outflag))
SPEC
    AG( Start.outflag
    -> EF( Interrupt.outflag | End.outflag))
SPEC
    EG( Start.outflag
    -> AF( Interrupt.outflag & End.outflag))
SPEC
    EG( Start.outflag
    -> AF( Interrupt.outflag | End.outflag))
SPEC
    EG( Start.outflag
    -> EF( Interrupt.outflag & End.outflag))
SPEC
    EG( Start.outflag
    -> EF( Interrupt.outflag | End.outflag))
```

Fig. 8. Generated CTL specifications for the process in Fig. 2

V. Experiments

The goal of the conducted experiments was to assess the efficiency of the workflow shown in Fig. 1, in which:

1) An ArchiMate specification is prepared with Archi tool.
2) With "one-click" a corresponding NuSMV model is generated.
3) A set of specifications for the obtained model is checked by calling NuSMV.

In particular we were focused on the time efficiency of the third step, as it seemed to be crucial for the presented approach. During the experiments NuSMV was launched as an external process in the interactive mode, then commands to load models, report numbers of variables and check automatically generated specifications were submitted. The NuSMV output was grabbed and information on models processed, as well as the execution times were collected.

A. Artificial test cases

Tests reported in this section aimed at assessing the relationship between a process complexity and the time required to check CTL specifications with NuSMV. Obviously, process

specifications can form very diverse structures, however, we assumed that the key feature characterizing the process complexity is the numbers of branches and loops in the control flow. Hence, the basic process pattern, which was analyzed, comprised several branches and loops placed between two events *Start* and *Stop*. Fig. 9 shows an example, in which four subprocesses are arranged to form two branches and two loops. We have prepared a number of test cases being variations of this pattern and the results were summarized in Table I.

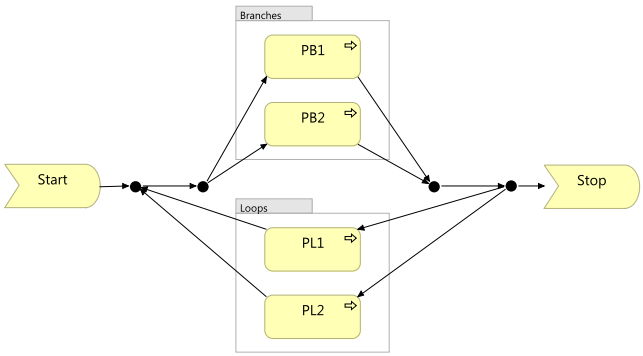

Fig. 9. Test case 4p2b2l: a process consisting of four subprocesses forming two branches and two loops

Each case name in Table I is encoded as a string: $n\,\mathrm{p}\,m\{\mathrm{b}|\mathrm{ab}\}\,k\,\mathrm{l}$, where n–number of processes, m–number of branches and k–number of loops. Symbol 'b' appearing in the case name means that the XOR branches (*xorJoins*) were used, wheras 'ab' corresponds to parallel branches placed between two *AndJoins*.

The column marked as CC gives the value of cyclomatic complexity, a commonly used measure that can be interpreted as the number of independent paths in the specification [31]. It is calculated according to following formula: cc = e-n+2p, where where e is the number of edges, n is the number of nodes and p is the number of connected components. The formula is applicable to sequential processes. For processes containing parallel activities the number of independent paths can be smaller, as activities starting and finishing synchronously actually build up a single control flow paths. Corrected values of cyclomatic complexity are added in parentheses.

Subsequent columns give numbers of state variables, total numbers of states and numbers of reachable states. *Dia* is a system diameter, i.e. the longest path from the initial state.

Columns *T* and *TPS* give total execution times and the execution time divided by the number of specifications (For each case 8 specifications were automatically generated and tested.)

NuSMV uses internally OBDD as a representation of a state transition system. It is well known, that the size of OBDD depends heavily on the variable ordering [8], [32], [33]. Selecting an optimal ordering is a NP-hard task, however several heuristics can be used to improve the ordering and in consequence reduce the model size, as well as the processing time. The corresponding option offered by NuSMV is called: *dynamic variable ordering*. The column *TD* gives the total processing time for dynamic variable ordering (the sift algorithm described in [34] was applied), whereas *TDPS* shows *TD* divided by the number of specifications.

The test cases are arranged into three groups. The first comprises process specifications with various numbers of branches, however without any loops. As it can be observed, in spite of growth of problem size, the numbers of reachable states and processing times remain relatively small.

The second group contains hard cases, i.e. those with several OR branches and loops. This makes both the numbers of reachable states very high and for the most complex case 24p12b12 (24 processes, 12 branches and 12 loops) the time spent to check one specification ranges to 80 seconds.

The third group includes cases with parallel branches that are synchronized at an *AndJoin*. This makes the space of reachable states much smaller, as well as keeps execution times relatively small (up to 560ms for the case 24p12ab12l). It can be noticed that for the cases 7p2ab5l, 8p3ab5l, 9p4ab5l and 10p5ab5l the numbers of reachable states are equal. This is a natural consequence of the model structure. Regardless of the number of parallel subprocesses, they all start and finish in the same states synchronized by the *AndJoin*.

Table I shows that applying dynamic variable ordering can be beneficial for complex models, e.g. for 16p8b8l, 20p10b10l and 24p12b12l it allowed to reduce the processing time up to 90%. For smaller models it was inefficient.

B. Business process example

In this section we present results of performance tests for a medium size business process from the banking domain. The process evaluates a credit order sent over Internet and issues an approval or a rejection decision. Its description was prepared with Archi based on the EPC model published in [35] (on the page 44).

The process definition comprises five views presented in Fig. 10, Fig. 11 and Fig. 12. Following the modeling approach typical for the EPC, multiple events defining process states or conditions are used. Moreover, the events mark boundaries of subprocesses represented by the views. They bind them into a coherent model comprising branches or loops, whose endpoints are indicated by the referenced events. Due to limited space the presented models are restricted to behavioral elements, i.e. they do not comprise information on engaged roles, involved systems and processed data, which can be specified in ArchiMate.

The process is divided into three stages. The first, presented in Fig. 10, aims at collecting customer data for the received credit order. The subprocess handles two cases: when a customer is new or already present in the database of the banking system, what may influence the credit decision.

The next stage shown in Fig. 11 ends with three conditions: the credit order is accepted (green credit decision), rejected (red credit decision) or inconclusive (gray credit decision).

The final stage (Fig. 12) comprises activities, whose goals are to prepare the credit offer (in the case of green credit

TABLE I
RESULTS OF PERFORMANCE TESTS.

Name	CC	Vars	Total states	Total states	Reachable	Dia	T [ms]	TPS [ms]	TD [ms]	$TDPS$ [ms]
2p2b0l	2	15	$2^{12} \cdot 3^2 \cdot 4^1$	147456	910	14	24.47	3.06	40.14	5.02
3p3b0l	3	19	$2^{15} \cdot 3^3 \cdot 5^1$	4423680	1830	14	38.18	4.77	86.03	10.75
4p4b0l	4	23	$2^{18} \cdot 3^4 \cdot 6^1$	$1.27402 \cdot 10^8$	3118	14	50.37	6.30	110.30	13.79
5p5b0l	5	27	$2^{21} \cdot 3^5 \cdot 7^1$	$3.56726 \cdot 10^9$	4810	14	57.27	7.16	205.65	25.71
8p8b0l	8	39	$2^{30} \cdot 3^8 \cdot 10^1$	$7.04482 \cdot 10^{13}$	12670	14	135.37	16.92	355.86	44.48
12p12b0l	12	55	$2^{42} \cdot 3^{12} \cdot 14^1$	$3.27222 \cdot 10^{19}$	30990	14	357.63	44.70	663.00	82.87
4p2b2l	4	27	$2^{21} \cdot 3^4 \cdot 4^1 \cdot 5^1$	$3.39739 \cdot 10^9$	91926	25	1032.42	129.05	596.09	74.51
6p3b3l	6	35	$2^{27} \cdot 3^6 \cdot 5^1 \cdot 6^1$	$2.93534 \cdot 10^{12}$	342294	25	1025.12	128.14	1356.48	169.56
8p4b4l	8	43	$2^{33} \cdot 3^8 \cdot 6^1 \cdot 7^1$	$2.36706 \cdot 10^{15}$	941906	25	3808.37	476.05	2895.52	361.94
10p5b5l	10	51	$2^{39} \cdot 3^{10} \cdot 7^1 \cdot 8^1$	$1.8179 \cdot 10^{18}$	2153330	25	11216.15	1402.02	7116.50	889.56
16p8b8l	16	75	$2^{57} \cdot 3^{16} \cdot 10^1 \cdot 11^1$	$6.82405 \cdot 10^{26}$	$1.36819 \cdot 10^7$	25	74301.78	9287.72	7552.02	944.00
20p10b10l	20	91	$2^{69} \cdot 3^{20} \cdot 12^1 \cdot 13^1$	$3.21085 \cdot 10^{32}$	$3.44569 \cdot 10^7$	25	240468.40	30058.55	22965.01	2870.63
24p12b12l	24	107	$2^{81} \cdot 3^{24} \cdot 14^1 \cdot 15^1$	$1.43403 \cdot 10^{38}$	$7.47279 \cdot 10^7$	25	640779.35	80097.42	70495.57	8811.95
7p2ab5l	7(6)	37	$2^{29} \cdot 3^7 \cdot 8^1$	$9.39309 \cdot 10^{12}$	109038	25	978.63	122.33	1428.27	178.53
8p3ab5l	8(6)	39	$2^{30} \cdot 3^8 \cdot 8^1$	$5.63586 \cdot 10^{13}$	109038	25	1011.67	126.46	1195.66	149.46
9p4ab5l	9(6)	41	$2^{31} \cdot 3^9 \cdot 8^1$	$3.38151 \cdot 10^{14}$	109038	25	1340.40	167.55	1362.44	170.30
10p5ab5l	10(6)	43	$2^{32} \cdot 3^10 \cdot 8^1$	$2.0289 \cdot 10^{15}$	109038	25	1059.06	132.38	1811.73	226.47
16p8ab8l	16(9)	61	$2^{44} \cdot 3^{16} \cdot 11^1$	$8.33015 \cdot 10^{21}$	265530	25	6205.62	775.70	4423.14	552.89
20p10ab10l	20(11)	73	$2^{52} \cdot 3^{20} \cdot 13^1$	$2.0414 \cdot 10^{26}$	418078	25	3257.10	407.14	7084.61	885.58
24p12ab12l	24(13)	85	$2^{60} \cdot 3^{24} \cdot 15^1$	$4.88429 \cdot 10^{30}$	614578	25	4484.91	560.61	12463.37	1557.92

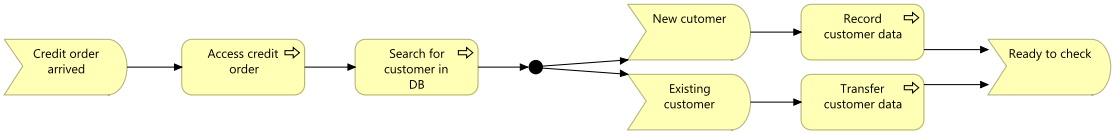

Fig. 10. Collect data

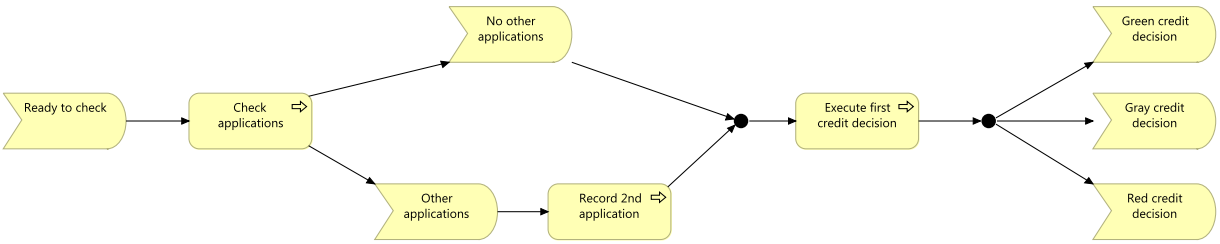

Fig. 11. First stage of the credit decision

decision), process credit order rejection for the red credit decision or reevaluate the submitted documents, if the decision was inconclusive (gray). The third case loops back the control flow to the results of the second stage.

Based on this specification the NuSMV model was generated. During the refactoring phase several joins were added before events: *Ready to check*, *Green credit decision*, *Gray credit decision* and *Red credit decision* events. An automatically generated specification checked during the experiment was the following:

```
AG( Credit_order_arrived.outflag ->
AF( Credit_offer_sent.outflag |
Contentual_problems.outflag |
Credit_order_rejected.outflag )).
```

It expresses the requirement that each credit order ends with a conclusive decision (an offer is sent or the order is rejected) or there are some problems related to the collected documents (conentual problems) that need further processing.

The cyclomatic complexity of the whole model was equal $cc = 55-42+2 \cdot 1 = 15(14)$. The value in the parentheses takes into account parallel tasks: *Create and send correspondence* and *Archive documents*.

Performing model checking for interleaving semantics of events, forks and joins failed. Without dynamic variable or-

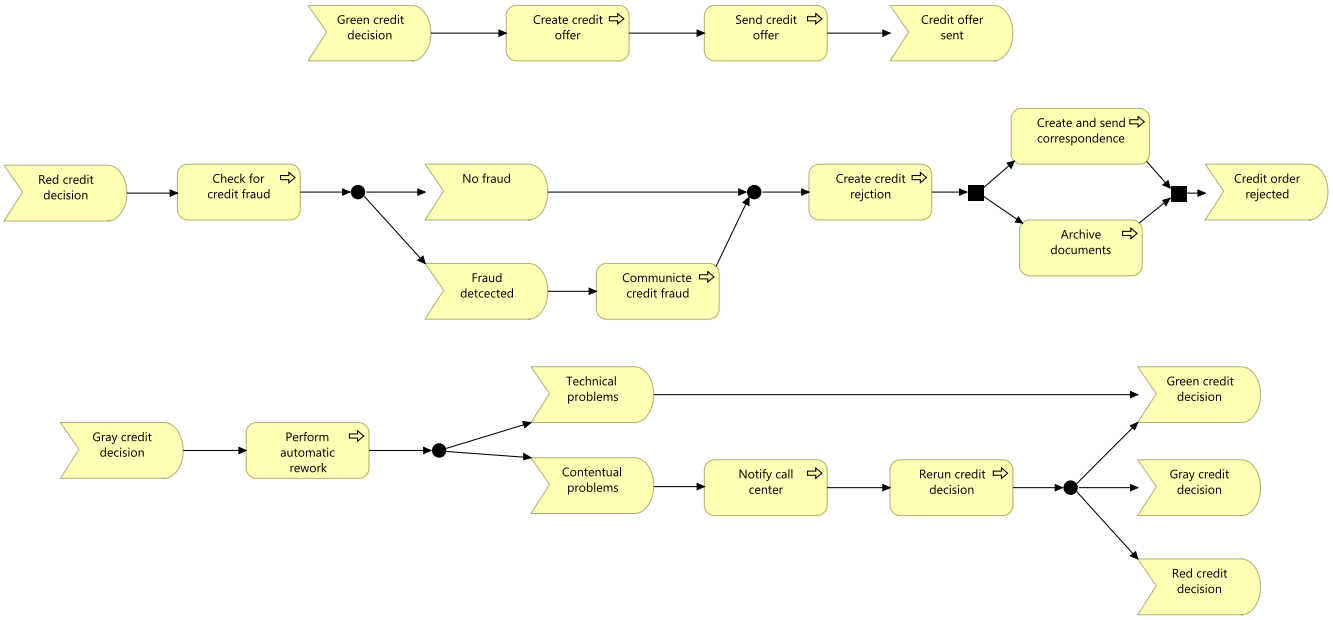

Fig. 12. Final stages of the credit decision

dering NuSMV consumed about 6GB memory. After applying dynamic ordering, the memory footprint was smaller (less then 500kB). However, in both cases the verification processes did not terminate in an acceptable time. This result was somehow disappointing, because based on the cyclomating complexity value, we were expecting that the verification is feasible.

For the synchronous semantics of events, binary forks and joins the model verification succeeded. The total number of states was equal $2^{53} \cdot 3^{17} \cdot 5^2 \approx 2.90798 \cdot 10^{25}$. The number of reachable states was equal $3.05508 \cdot 10^9$ (i.e. greater by 2 orders of magnitude than the most complex processes in Table I). Without dynamic variable ordering the procesisng time was really long: 768522.57 ms = 12.5 min. However, applying dynamic variable reordering (sift method) reduced it to 41706.78 ms. We consider this result acceptable.

VI. Conclusions

This paper investigates the problem of automatic verification of behavioral specification embedded within ArchiMate models. We propose an approach consisting in fully automatic translation of ArchiMate specification into a model in NuSMV language and then verifying it with the NuSMV model checker. Requirements specification in form of CTL formulas can be entered by user, but the implemented tool is capable of generating specifications based on analysis of control flows.

The main concern of our work was the time efficiency of the verification process. We tested it on a set of of artificial business process specifications, as well as on a real business process example.

The results show, that the described approach can be applied in an interactive verification tool, however, due to state space explosion problem, verification of complex business process specifications can still be a challenge for symbolic model checkers. Hence, dedicated model generation techniques focusing on keeping models compact, e.g. simplifying the models, avoiding interleaving and generating partial models, should be employed.

Although the presented considerations are related to processes defined ArchiMate language, the results of tests are applicable to process models defined in other languages including BPMN and EPC.

References

[1] S. Morimoto, “A survey of formal verification for business process modeling,” in *Proceedings of the 8th international conference on Computational Science, Part II*, ser. ICCS ’08. Berlin, Heidelberg: Springer-Verlag, 2008. doi: 10.1007/978-3-540-69387-1_58. ISBN 978-3-540-69386-4 pp. 514–522.

[2] OMG, “Business Process Model and Notation (BPMN) version 2.0,” OMG, Tech. Rep., January 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[3] A. Scheer, *Aris - Business Process Modeling*, ser. ARIS - Business Process Modeling. Springer, 1999, no. v. 2. ISBN 9783540644385

[4] A.-W. Scheer and M. Nüttgens, “ARIS architecture and reference models for business process management,” in *Business Process Management*. Springer, 2000, pp. 376–389.

[5] The Open Group, *Open Group Standard. Archimate 2.1 Specificattion*. Van Haren Publishing, Zaltbommel, 2013. ISBN 978 94 018 0003 7

[6] E. M. Clarke and J. M. Wing, “Formal methods: State of the art and future directions,” *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[7] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, “Model checking and the state explosion problem,” in *Tools for Practical Software Verification*. Springer, 2012, pp. 1–30.

[8] R. E. Bryant, “Symbolic boolean manipulation with ordered binary-decision diagrams,” *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 293–318, 1992.

[9] R. Huuck, "Formal verification, engineering and business value," in Proceedings First International Workshop on *Formal Techniques for Safety-Critical Systems,* Kyoto, Japan, November 12, 2012, ser. Electronic Proceedings in Theoretical Computer Science, P. C. Olveczky and C. Artho, Eds., vol. 105. Open Publishing Association, 2012. doi: 10.4204/EPTCS.105.1 pp. 1–4.

[10] P. Beauvoir, "Archi, archimate modelling tool," 2015, [Online; accessed March 2015]. [Online]. Available: http://www.archimatetool.com/

[11] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*. Springer, 2002, pp. 359–364.

[12] P. Szwed, "Verification of ArchiMate behavioral elements by model checking," in *Computer Information Systems and Industrial Management*, ser. Lecture Notes in Computer Science, K. Saeed and W. Homenda, Eds. Springer International Publishing, 2015, vol. 9339, pp. 132–144. ISBN 978-3-319-24368-9. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-24369-6_11

[13] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[14] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!" in *Computer Aided Verification*. Springer, 2007, pp. 121–125.

[15] K. Watahiki, F. Ishikawa, and K. Hiraishi, "Formal verification of business processes with temporal and resource constraints," in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1173–1180.

[16] B. Anderson, J. V. Hansen, P. Lowry, and S. Summers, "Model checking for e-business control and assurance," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 35, no. 3, pp. 445–450, 2005. doi: 10.1109/TSMCC.2004.843181

[17] M. Mongiello and D. Castelluccia, "Modelling and verification of BPEL business processes," in *Model-Based Development of Computer-Based Systems and Model-Based Methodologies for Pervasive and Embedded Software, 2006. MBD/MOMPES 2006. Fourth and Third International Workshop on*. IEEE, 2006, pp. 5–pp.

[18] X. Fu, T. Bultan, and J. Su, "Formal verification of e-services and workflows," in *Web Services, E-Business, and the Semantic Web*. Springer, 2002, pp. 188–202.

[19] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, "Automatic verification of data-centric business processes," in *Proceedings of the 12th International Conference on Database Theory*. ACM, 2009, pp. 252–267.

[20] M. T. Wynn, H. Verbeek, W. M. van der Aalst, A. H. ter Hofstede, and D. Edmond, "Business process verification-finally a reality!" *Business Process Management Journal*, vol. 15, no. 1, pp. 74–92, 2009.

[21] W. M. Van der Aalst and A. H. Ter Hofstede, "YAWL: yet another workflow language," *Information systems*, vol. 30, no. 4, pp. 245–275, 2005.

[22] R. Klimek and P. Szwed, "Verification of ArchiMate process specifications based on deductive temporal reasoning," in *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, Kraków, Poland, September 8-11, 2013.*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., 2013, pp. 1103–1110. [Online]. Available: http://fedcsis.org/2013/

[23] R. Klimek, P. Szwed, and S. Jedrusik, "Application of deductive reasoning to the verification of ArchiMate behavioral elements," *Informatyka Ekonomiczna*, vol. 29, pp. 76–97, 2013.

[24] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness, "Verification of the futurebus+ cache coherence protocol," *Formal Methods in System Design*, vol. 6, no. 2, pp. 217–232, 1995.

[25] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso, "Model checking early requirements specifications in tropos," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. IEEE, 2001, pp. 174–181.

[26] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "Mbp: a model based planner," in *Proc. of the IJCAIŠ01 Workshop on Planning under Uncertainty and Incomplete Information*, 2001.

[27] E. Clarke and W. Heinle, "Modular translation of statecharts to smv," Citeseer, Tech. Rep., 2000.

[28] M. Szpyrka, A. Biernacka, and J. Biernacki, "Methods of translation of Petri nets to NuSMV language," in *Proceedings of the 23th International Workshop on Concurrency, Specification and Programming, Chemnitz, Germany, September 29 - October 1, 2014.*, ser. CEUR Workshop Proceedings, L. Popova-Zeugmann, Ed., vol. 1269. CEUR-WS.org, 2014, pp. 245–256. [Online]. Available: http://ceur-ws.org/Vol-1269/paper245.pdf

[29] P. Szwed, W. Chmiel, S. Jedrusik, and P. Kadluczka, "Business processes in a distributed surveillance system integrated through workflow," *Automatyka/Automatics*, vol. 17, no. 1, pp. 127–139, 2013.

[30] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuxmv symbolic model checker," in *CAV*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, 2014. ISBN 978-3-319-08866-2 pp. 334–342.

[31] T. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, vol. SE-2, no. 4, pp. 308–320, Dec 1976. doi: 10.1109/TSE.1976.233837

[32] H. R. Andersen, "An introduction to binary decision diagrams," *Lecture notes, available online, IT University of Copenhagen*, 1997.

[33] P. Szwed and A. Ligeza, "Application of OBDD diagrams in verification of tabular rule systems," *Schedae Informaticae*, vol. 14, 2005.

[34] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society Press, 1993, pp. 42–47.

[35] B. Weiß, "Business process modelingand analysis in banks," 2011, [Online; accessed April 2015]. [Online]. Available: http://www.bpm.scitech.qut.edu.au/seminars/2011/BurkhardWeissBPMSeriesTalk.pdf