INFORMATYKA EKONOMICZNA BUSINESS INFORMATICS 3(29) • 2013
ISSN 1507-3858

Radosław Klimek, Piotr Szwed, Stanisław Jędrusik
AGH University of Science and Technology, Krakow
e-mail: {rklimek, pszwed}@agh.edu.pl, jedrusik@zarz.agh.edu.pl

APPLICATION OF DEDUCTIVE REASONING TO THE VERIFICATION OF ARCHIMATE BEHAVIORAL ELEMENTS*

Abstract: The formal verification of business models has recently become an intensively researched area. It is expected that the application of formal tools may bring such benefits to organizations as the improved quality of products and services and a lower ratio of operational errors. In this paper we discuss the application of a deduction-based method for the verification of the behavioral aspects of ArchiMate models. The first step in our method consists in the translation of the ArchiMate model into Linear Temporal Logic (LTL) formulas. The resulting LTL formulas are then verified to check the expected temporal properties. The verification process is based on the semantics tableaux method and is conducted with an LTL prover. The method is discussed using an example of a business process implemented within a surveillance system.

Keywords: Deductive temporal reasoning, software verification, ArchiMate, semantics tableaux method, Linear Temporal Logic.

1. Introduction

Nowadays, enterprise IT systems follow the Service Oriented Architecture paradigm. They are distributed into a set of services that can be composed using the choreography pattern or orchestrated with business processes. As the systems become more and more complex, various languages and notations aimed at modeling enterprise architectures are used. They include process-oriented BPMN and EPCs, UML diagrams and, more recently, ArchiMate, the language allowing to build coherent models expressing relations between various elements of an enterprise architecture: processes, roles, artifacts, services, software components and infrastructure.

An advantage of business notations is that they are relatively simple and intuitive to use. They can be used in communication between various involved stakeholders, define models that can be further ported to executable process languages, e.g. BPEL or XPDL as well, as may be considered a formal business reference that is required for various certification authorities.

* This work was supported by the AGH UST internal grant no. 11.11.120.859.

Recently, with the growing popularity of business modeling languages, various attempts to apply in this field the tools and techniques of formal verification can be observed. Apparently, formal methods may bring such benefits to business as the improved quality of products and services, smaller ratio of operational errors, reduced cost and greater competitiveness.

Nevertheless, the application of formal methods to the verification of business models or its parts, e.g. processes, encounter various barriers. The first problem is related to ambiguities in specifications. Actually, most business modeling notations lack formal semantics. It is provided when translating the models into the statements of executable languages.

The second issue that hinders the application of formal methods is the lack of tools, preferably integrated with modeling platforms, that would automatically build formal specifications and perform verification indicating errors or issuing warnings about potential problems.

In this paper we aim at making a step towards the automated verification of a business model. We investigate an application of formal deduction-based techniques to an automated verification of behavioral descriptions embedded within ArchiMate models. We propose a verification system comprising of three tools: an off-the-shelf ArchiMate modeler, a generator of temporal logic formulas describing the process model, and a tableaux based prover.

The motivation for the work is the lack of tools for the deduction-based formal verification of business models. Another motivation is the lack of tools for the automatic generation of logical specifications from ArchiMate models.

The contribution of the work arises from the following: rules for the automatic generation of logical specifications considered as sets of temporal logic formulas are defined and a complete deduction-based system, which enables an automated and formal verification of ArchiMate business models is proposed. The reasoning process is performed using the semantic tableaux method for temporal logic. An example of the approach is provided.

The paper is organized as follows: Section 2 discusses the problems of the application of formal methods (and in particular Temporal Logic) to verification. It is followed by Section 3 briefly describing the ArchiMate language. Then, in Section 4, an example of a process specification using ArchiMate is provided. Section 5 defines the rules governing the translation of ArchiMate models to LTL formulas. In Section 6 an architecture of the verification system is described and an example of a checked property is given. Section 7 presents the known approaches to the verification of business models. Finally, Section 8 gives the concluding remarks.

2. Formal methods

Formal methods are understood as a set of principles for the precise formulation of the important artifacts formed when developing and refining software models. They enable revealing, questioning and removing ambiguities and flaws in specifications.

Moreover, the formality of the used languages leads to rigorous analysis and verification.

System verification methods fall into two categories [Huth et al. 2004; Clarke et al. 1996]: *proof-based* and *model-based*. The general idea of proof-based methods is very simple. The system under verification is expressed as a set of formulas Γ in an appropriate symbolic logic. The specification is represented by another formula ϕ. The verification method is to try to prove that $\Gamma \vdash \phi$. In the model-based approach the system is expressed by a model M in an appropriate symbolic logic. The specification is represented by a formula ϕ. The verification methods is try to check if model M satisfies ϕ, i.e. $M \models \phi$.

The key issue in formal verification methods is the choice of the symbolic logic language. The language should be strong enough to express all the significant properties of the verified system. On the other hand, it must be ensured that the logical system is sound and complete. When selecting the logical language, one needs to keep in mind that the increase of a language's expressiveness may lead to the loss of soundness or completeness.

One of the most frequently used logics in system verification is *Temporal Logic* (TL). It brings in logical symbols for reasoning about varying logical valuations of formulas throughout the flow of time. Two basic unary operators are $\Diamond$ for "sometime (or eventually) in the future" and $\Box$ for "always in the future". The distinguishing feature of TL formulas is that the value of the TL formula is not fixed as true or false in any given model as in predicate, propositional or other classic logic. Any model in TL may have many states. The TL formula may be true in some of them and false in the others. TL is also a well-established formalism for the specification and verification of reactive and concurrent systems. It allows to describe both temporal relations between the reached states or events occurring within a system and to specify the expected properties.

Liveness and safety are standard elements of a taxonomy of system properties. *Liveness* means that the computational process achieves its goals, i.e. something good eventually happens. *Safety* means that the computational process avoids undesirable situations, i.e. something bad never happens.

In recent years a number of temporal logics has been proposed. Temporal logic exists in many varieties, however, these considerations are limited to the *linear-time temporal logic* (LTL). Linear temporal logic refers to infinite sequences of computations considered as linear structures and our attention is focused on the *propositional linear time logic* PLTL. These sequences are formally represented as Kripke structures, which define semantics of TL, i.e. a syntactically correct, or a well-formed, formula can be satisfied by an infinite sequence of truth evaluations over a set of *atomic propositions* AP. The basic issues related to temporal logics and their syntax and semantics are discussed in many works, e.g. [Emerson 1990; Wolter, Wooldridge 2011].

The properties of time structure are fundamental to logic. Of particular significance is minimal temporal logic, e.g. [van Benthem 1993-95], also known as temporal logic of the class K. Minimal temporal logic is an extension to a classical calculus defining the axiom $\Box(P \Rightarrow Q) \Rightarrow (\Box P \Rightarrow \Box Q)$ and the inference rule $\vdash P \Longrightarrow \vdash \Box P$. The essence of the logic is the fact that there are no specific assumptions pertaining to the time structure order. The following formulas may be considered as typical examples of this logic: $action \Rightarrow \Diamond reaction$, $\Box(send \Rightarrow \Diamond receive)$, etc. The considerations of this work are limited to this logic since it allows to define many system properties (safety, liveness) and it is also easier to build a deduction engine or use the existing verified provers.

The application of deductive approach to the validation of business processes faces the problem of the automatic obtaining of logical specifications from business models. The need to build them manually can be recognized as a major obstacle to untrained users, due to the fact that the process of specifying a large collection of formulas is difficult and monotonous.

For temporal logic, that is a suitable language for expressing behavior and reasoning about it, such specifications are constituted by the set of temporal logic formulas $\Gamma = \{F_1, ..., F_n\}$ When the number of formulas is large, which is not an extraordinary situation, then in practice it is not possible to build a logical specification manually. It follows that this process usually requires (very) skilled human intervention. Thus, in order to move the deductive-based formal verification from a pen-and-paper approach to the engineers' needs, the automation of the generation process seems particularly important.

3. ArchiMate

ArchiMate [The Open Group 2012; Van Den Berg et al. 2007] is a contemporary, open and independent language for the description of enterprise architectures. It comprises three main modeling layers: business, application and technology. The *business* layer includes business processes and objects, functions, events, roles and services. The *application* layer contains components, interfaces, application services and data objects. The *technology* layer gathers such elements as artifacts, nodes, software, devices, communication channels and networks. ArchiMate allows to present an architecture in the form of views which, depending on the needs, can include only items in one layer or can show the vertical relations between layers, e.g. a relationship between a business process and a function of the component software.

ArchiMate was built in opposition to UML [Rumbaugh et al. 2004], which can be seen as a collection of unrelated diagrams, and Business Process Modeling Notation BPMN [OMG, January 2011] which covers mainly the behavioral aspect of enterprise architecture. The definition of a language has been accompanied by the 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. ArchiMate goes beyond UML [Nick 2009]: it defines a metamodel on the basis of which a user can create and illustrate the relationships between elements of different layers.

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 the use of the *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 (sub-processes or functions). It should be also noted that the *Business Activity* present in the 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 [OMG, January 2011], after adopting a certain modeling convention its expressiveness can be similar [Szwed et al. 2013]. An advantage of the language is that it 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) [Scheer 1999; Scheer, Nüttgens 2000]. Indeed, all the behavioral elements of both languages are exactly the same: events, functions (or processes in ArchiMate) and various joins and splits (XOR, OR and AND). In spite of almost the 20 year presence of EPC tools on the market and thousands of deployments in modeling business organizations, there is no consensus of semantics of EPCs. Analyses of several semantics variations have revealed certain erroneous patterns, e.g. the famous vicious circle [Van der Aalst et al. 2002] resulting in a deadlock caused by the improper use of synchronization joins. Due to the correspondence between EPC and ArchiMate constructs, discussions and discovered problems related to EPCs semantics apply also to ArchiMate models.

4. Example of a business model

In this section we present an exemplary ArchiMate model of a process developed and used within a surveillance system. The goal of the process is to coordinate the actions of intervention groups whose aim is to apprehend an object violating a restricted area. Such processes are often implemented in facilities for which the protection of restricted areas is particularly important, e.g. airports or military zones. This model is a result of our work on an intelligent surveillance system based on the analysis of digital images developed within the SIMPOZ project[1]. One of the

[1] http://www.simpoz.pl/..

solutions adopted in the system was to use a workflow defined in XPDL language in order to integrate various system components. The development of the workflow was preceded by a business analysis step, in which a model of a surveillance system was elaborated using the ArchiMate language [Szwed et al. 2013].

The entire model of the workflow is far too complex to be presented here. In order to illustrate the verification procedure we selected only an excerpt. The extracted part consists of one high-level process (*Apprehending an object violating a restricted zone*) and two extending sub-processes (*Request for support*, *Establish contact with the intervention group*). We assume that every process starts and ends with an event as well as that the events may be triggered by the environment or by sub-processes.

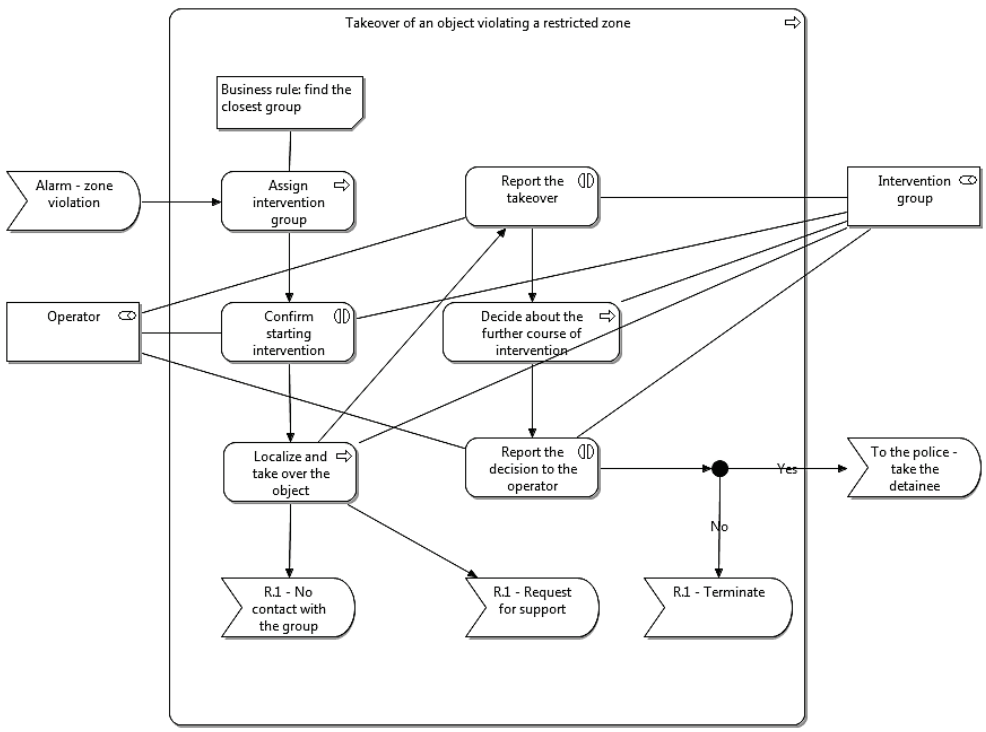


Figure 1. Model of the process of apprehending an object violating restricted area

Source: own elaboration.

The high-level process of apprehending an object violating a restricted area is presented in Figure 1. It is triggered by the zone violation event (*Alarm – zone violation*). The first task is to assign an intervention group (*Assign intervention group*). The assignment is based on a business rule, e.g. *find the closest group*. The next step requires some form of interaction between the operator and the intervention

group in order to confirm starting (*Confirm starting intervention*). Then the intervention group is trying to localize and apprehend an object violating the restricted area (*Localize and apprehend the object*). While performing this task the intervention group may request support. Contact with the intervention group may also be lost. All these exceptional cases are modeled as events (*R.1 − Request for support*, *R.1 − No contact with the group*). The appropriate event handling procedures are presented in Figures 2 and 3. The successful completion of latter task should be reported to the operator (*Report the apprehension*). Then the intervention group must decide about the further course of intervention (*Decide about the further course of intervention*). The operator should be informed about this decision (*Report the decision to the operator*). The main flow ends with two alternative events: either calling the police to take the detainee (*To the police − take the detainee*) or with a simple termination in the case, when police participation is not required (*R.1 − Terminate*). The terminated event has special semantics. It terminates all instances of the process and its child sub-processes and removes the unhandled events from the system queues. The internal *Stop* event occurring in the two other sub-processes terminates only a given sub-process.

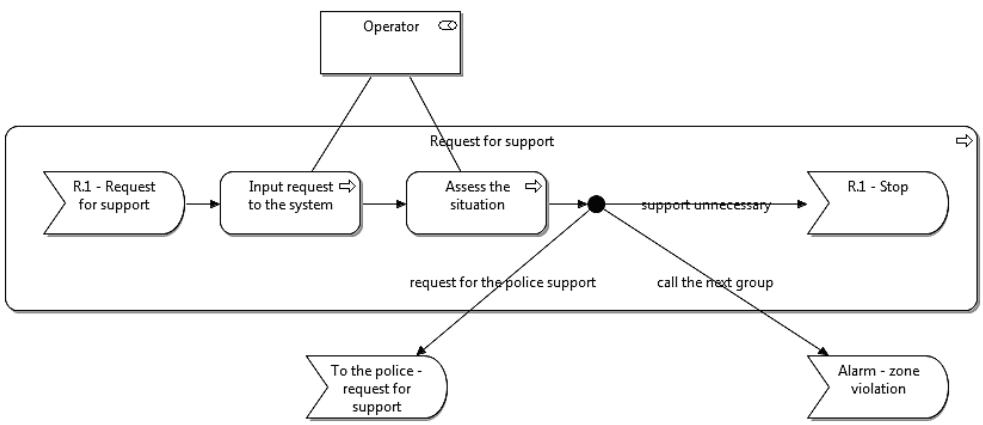


Figure 2. Model of the process handling support requests

Source: own elaboration.

The model in Figure 2 presents the procedure of handling requests for support (*R.1 − Request for support*). This event is triggered by the intervention group. The procedure begins with entering the request to the system (*Input request to the system*). Then the operator makes an assessment of the situation (*Assess the situation*), which can yield three possible outcomes. Firstly, the operator may find that support is not necessary; this is modeled as triggering stop event (*R.1 − Stop*). Secondly, the operator may conclude that additional intervention group is needed. The process of sending the next intervention group may be simply modeled by triggering recursively

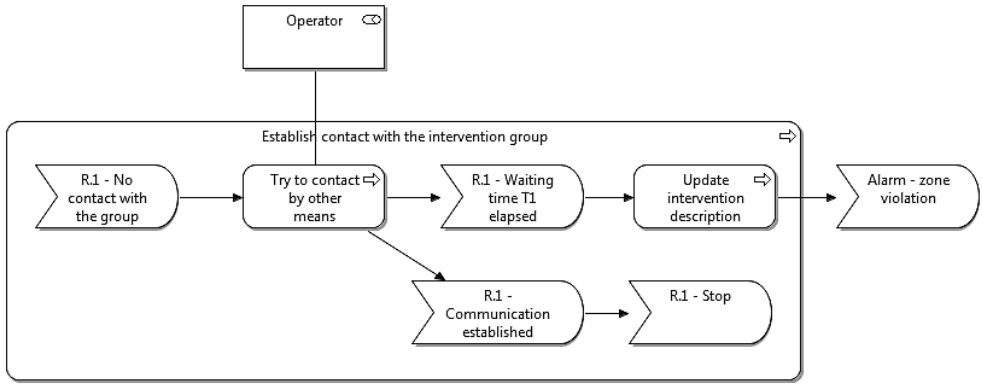


Figure 3. Model of the process of establishing contact with the intervention group

Source: own elaboration.

the zone violation event (*Alarm - zone violation*). Finally, in a severe case, a request for the police support may be issued (*To the police − request for support*).

The last out of the set of extending sub-processes is presented in Figure 3. It handles cases when there is no contact with the intervention group (*R.1 − No contact with the group*). The simplest and most obvious solution is to contact using other communication channels (*Try to contact by other means*). Establishing the contact is modeled as receiving appropriate event (*R.1 − Communication established*). This path ends with simple stop event (*R.1 − Stop*). In the case of time out (*R.1 − Waiting time T1 elapsed*), the dispatcher should update the intervention description and send the next intervention group (*Update intervention description and send a new group*). Similarly to the previous sub-process, this is modeled as triggering the zone violation event (*Alarm − zone violation*).

5. Modelling ArchiMate behavioural constructs

This section gives the formally defined rules for the translation of behavioral elements within an ArchiMate specification into LTL formulas. 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 an element or a relation. While generating the LTL formulas describing the behavioral aspects of ArchiMate model, we focus on the components of the Business layer: processes (or functions), events and various junctions. We apply a linear procedure which visits nodes, analyzes their successors and generates the LTL formulas describing control flows.

It should be noted that the ArchiMate behavioral constructs have no precisely defined semantics. In fact, the translation from an ArchiMate specification to LTL

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 multiple $\langle V, E, C, R, vt, et \rangle$ where

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

As the considerations in the work 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 $R=\{triggering, association, composition, other\}$.

5.1. Modelling atomic activities

By an atomic process (function, interaction) we mean a process that is not linked with other elements by a composition relation. It represents a basic unit of behavior which corresponds to the activity concept of other languages, e.g. UML. A process can be executed if its environment is in a state enabling its activation. After a process terminates, it causes state changes in the surrounding world [Gruninger, Fox 1994]. While defining LTL formulas describing processes and other elements, we follow the directions of relations and specify only the transitions between the internal states of elements and the caused states. In turn, the reached caused states enable the activation of other elements. Hence, after processing all the relevant elements, a complete network of states of the whole system specified in LTL is obtained. To model the execution of an atomic process, two states (and the corresponding propositions in LTL): start and end are used. A process is considered imperfect even if it has a name in imperative mood suggesting an achievement, e.g. *register invoice*, *scan document* or *send message*. Once invoked (the start state becomes active), the process can successfully complete reaching the end state or be interrupted by an event starting an alternative flow of control. Such an approach to modeling business processes can be explicitly supported by language constructs. In particular, the BPMN notation allows to attach various types of interrupting events to activities, e.g. timer, error or cancellation. In ArchiMate, an association relation between processes and events can be used to distinguish the events triggered upon process completion and those interrupting the normal flow. Figure 4 illustrates this approach. The *end* state and *Interrupting event* are successors of the process *start* state. On the other hand, the states of the surrounding elements that can be reached by the normal triggering relation are successors of the *end* state.

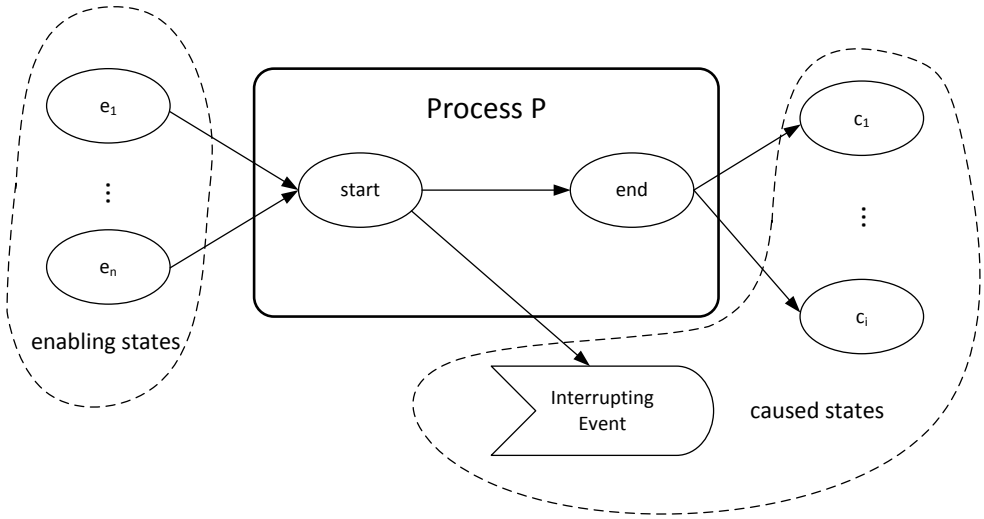


Figure 4. Two states: start and end used in a model of an atomic process

Source: own elaboration.

Atomic processes, functions and interactions (ArchiMate equivalents to activities) are imperfect and require two states (and propositions) to model their behavior. In turn, events and junctions are perfect and their activation can be modeled by singular states (truth values of propositions). To describe all behavioral elements in a uniform manner we define two functions, $start$ and end, that map vertices from ArchiMate model V to a set of propositions $Props$. It is assumed that if a certain vertex v represents a process, a function or an interaction, i.e. $v(v) \in \{Process; Funtion; Interaction\}$, then $start(v) \neq end(v)$. For the other elements: events and junctions $start(v) = = end(v)$ holds. We extend these functions to sets of vertices, i.e. $start(X) = = \bigcup_{x \in X} start(x)$ and $end(X) = \bigcup_{x \in X} end(x)$.

By $T(v) = \{v' : (v', v) \in E \wedge et(v;\ v') = triggering\}$ we will denote a set of behaviors that are triggered by v. $A(v) = \{v' : vt(v') = Event \wedge (v, v') \in \in E \wedge et(v, v') = association\}$ is a set of events linked with v by association relation.

$C(v) = \{v' : (v, v') \in E \wedge et((v, v')) = composition\}$ is a set of children of v as defined by composition relation. Let $\mathcal{F}(Props)$ be a set of LTL formulas obtained from a set of propositions $Props$ by applying classical or temporal operators and using parentheses. For brevity of notation we will further omit $Props$ and write simply $\mathcal{F}$. We define two auxiliary functions: $\delta_{ij}(p)$ mapping formulas $\mathcal{F} \times \mathbb{N} \times \mathbb{N} \to \mathcal{F}$ and $oneof\ (P)$ converting a set of propositions P into a formula in disjunctive normal form (2).

$$\delta_{ij}(p) = \begin{cases} p & \text{if } i \neq j \\ \neg(p) & \text{if } i = j \end{cases}, \tag{1}$$

$$oneof\,(P) = \bigvee_{i=1}^{|P|} \bigwedge_{j=1}^{|P|} \delta_{ij(p)}. \tag{2}$$

5.2. Atomic process, function or interaction

LTL formulas defining temporal relations for atomic activities (processes, functions and interactions) are generated according to Rule 1. Rules for other ArchiMate elements have a similar form. Each rule contains a precondition part separated from its postcondition by a horizontal line. The generated formulas are placed in double square brackets ⟦ ⟧.

Rule 1. Atomic process, function or interaction

$$\frac{\begin{array}{c} v \in V \\ vt(v) \in \{Process,\ Function, Interaction\} \\ C(v) = \emptyset \end{array}}{\begin{array}{c} [\![\Box\ (start(v) \Rightarrow \Diamond\, oneof(start(A(v)) \cup \{end(v)\})]\!] \in \mathcal{F} \\ p \in A(v) \cup \{end(v)\} \rightarrow [\![\Box\neg\,(start(v) \wedge p)]\!] \in \mathcal{F} \\ T(v) \neq \emptyset \rightarrow [\![\Box\,(end(v) \Rightarrow \Diamond\, oneof(start(T(v))))]\!] \in \mathcal{F} \\ p \in T(v) \rightarrow [\![\Box\neg\,(end(v) \wedge p)]\!] \in \mathcal{F} \end{array}}$$

LTL formulas describing the behavior for the sequence of two active elements *Confirm starting intervention* and *Assign intervention group* in Figure. 1 are presented below (original transcription is preserved). They were generated according to Rule 1.

```
%BusinessInteraction Confirm_starting_intervention (Confirm starting
intervention)
[]( Confirm_starting_intervention_start => <> Confirm_starting_intervention_
end ) &
[]~( Confirm_starting_intervention_start & Confirm_starting_intervention_end )
&
[]( Confirm_starting_intervention_end => <> Localize_and_take_over_the_
object_start ) &
[]~( Confirm_starting_intervention_end & Localize_and_take_over_the_object_
start ) &
% BusinessProcess Assign_intervention_group (Assign intervention group)
[]( Assign_intervention_group_start => <> Assign_intervention_group_end ) &
[]~( Assign_intervention_group_start & Assign_intervention_group_end ) &
[]( Assign_intervention_group_end => <> Confirm_starting_intervention_start ) &
[]~( Assign_intervention_group_end & Confirm_starting_intervention_start ) &
```

Figure 5. An excerpt of the model with a sequence of two active elements

Source: own elaboration.

5.3. Events

According to the ArchiMate specification [The Open Group 2012], a business event is something that happens and influences behavioral elements (processes, functions and interactions). The events has no duration, thus they can be modeled as single boolean variables.

Functions $start(v)$ and $end(v)$ map an event v to the same proposition, which change value to true if the event occurs. An event can be linked by triggering relations with multiple recipients (or sinks in the Event Driven Architecture). Events are somehow similar to *AndJunctions*. The occurrence of both activates all elements linked by a triggering relation. However, we assume that, unlike *AndJunctions,* the activation of elements triggered by an event is not synchronized (c.f. Rule 2).

Rule 2. Event

$$\frac{v \in V \wedge vt(v) = Event}{\begin{array}{c} p \in T(v) \rightarrow [\![\Box (end(v) \Rightarrow \Diamond start(p))]\!] \in \mathcal{F} \\ p \in T(v) \rightarrow [\![\Box (start(p) \Rightarrow \Diamond \neg end(v))]\!] \in \mathcal{F} \end{array}}$$

5.4. Junctions

The ArchiMate language defines three types of connectors:

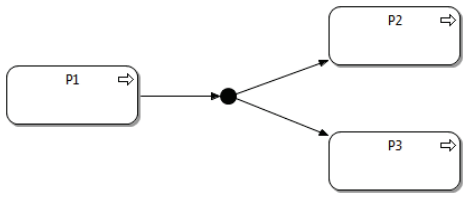
- *Junction* that can be considered a typical XOR connector, i.e. it activates exactly one output.
- *OrJunction* being a typical OR connector activating at least one output.
- *AndJunction* that can be used in two modes: when used to merge flows on input it requires their synchronization. In the second mode it starts a parallel execution of output flows.

ArchiMate junctions have counterparts in EPC, BPMN (exclusive, inclusive and parallel gateways) and XPDL transition restrictions [The Workflow Management Coalition 2008]. Similarly to events, junctions are modeled by single state variables. If a junction is activated, in a subsequent step elements linked by triggering relations should be activated according to assumed semantics. As there are three types of junctions, we define three rules (Rule 3-5) for translating them to LTL formulas.

Rule 3. Junction

$$\frac{v \in V \wedge vt(v) = Junction}{\begin{array}{c} T(v) \neq \emptyset \rightarrow [\![\Box (end(v) \Rightarrow \Diamond oneof(start(T(v))))]\!] \in \mathcal{F} \\ p \in T(v) \rightarrow [\![\Box (\neg end(v) \wedge p)]\!] \in \mathcal{F} \end{array}}$$

Figure 6 gives an example of a junction and LTL formulas generated according to Rule 1 for the process P1 and Rule 3 for the junction element.




```
% BusinessProcess P1 (P1)
[] ( P1_start => <> P1_end ),
[]~( P1_start & P1_end ),
[] ( P1_end => <> Junction ),
[]~( P1_end & Junction ),
% Junction Junction (Junction)
[] ( Junction => <>
(( P2_start & ~P3_start )  |  ( ~P2_
start & P3_start )  )),
[]~( Junction & P2_start ),
[]~( Junction & P3_start ),
```

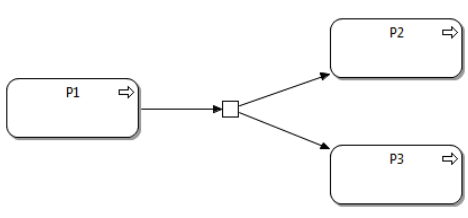
Figure 6. An example of junction (left) and corresponding LTL formulas (right)

Source: own elaboration.

Rule 4. OrJunction

$$\frac{v \in V \wedge vt(v) = OrJunction}{\begin{array}{l} T(v) \neq \emptyset \rightarrow \left[\!\left[\Box\,(end(v) \Rightarrow \Diamond(\bigvee_{z \in T(v)} start(z))) \right]\!\right] \in \mathcal{F} \\ p \in T(v) \rightarrow \left[\!\left[\Box\,(\neg end(v) \wedge (\bigwedge_{z \in T(v)} start(z))) \right]\!\right] \in \mathcal{F} \end{array}}$$

An excerpt of the model in which OrJunction appears is shown in Figure 7. The formulas generated according to Rule 4 state that after the junction is activated at least one of the processes P2 and P3 must be started.




```
% BusinessProcess P1 (P1)
[] ( P1_start => <> P1_end ),
[]~( P1_start & P1_end ),
[] ( P1_end => <> Or_Junction ),
[]~( P1_end & Or_Junction ),
% OrJunction Or_Junction (Or Junction)
[] ( Or_Junction =>
<> ( P2_start | P3_start ) ),
[]~( Or_Junction & (P2_start & P3_start) ),
```

Figure 7. An example of ORJunction (left) and corresponding LTL formulas (right)

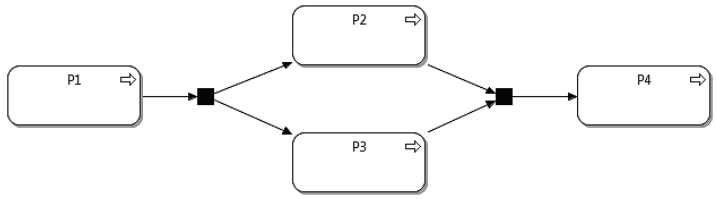
Source: own elaboration.

Let us define the relation $T^{-1}(v) = \{v' : (v', v) \in E \wedge et(v', v) = triggering\}$ describing a set of input for the element v.

Rule 5. AndJunction

$$\frac{v \in V \wedge vt(v) = AndJunction}{\begin{array}{l} T^{-1}(v) \neq \emptyset \rightarrow \left[\!\left[\Box\,((\bigwedge_{z \in T^{-1}(v)} end(z)) \Rightarrow \Diamond start(v)) \right]\!\right] \in \mathcal{F} \\ T^{-1}(v) \neq \emptyset \rightarrow \left[\!\left[\Box\,\neg(\bigwedge_{z \in T^{-1}(v)} end(z)) \wedge start(v)) \right]\!\right] \in \mathcal{F} \\ T(v) \neq \emptyset \rightarrow \left[\!\left[\Box\,(end(v) \Rightarrow \Diamond(\bigwedge_{z \in T(v)} start(z))) \right]\!\right] \in \mathcal{F} \\ T(v) \neq \emptyset \rightarrow \left[\!\left[\Box\,\neg(end(v) \wedge (\bigwedge_{z \in T(v)} start(z))) \right]\!\right] \in \mathcal{F} \end{array}}$$

Figure 8 shows a typical pattern of parallel process execution. After the process P1 terminates the first *AndJunction* becomes active. Then the processes P2 and P3 are launched. Activation of the process P4 requires the synchronization of P2 and P3, which is provided by the application of the second junction (*AndJunction_0*). The corresponding LTL formulas are given below.




```
% BusinessProcess P1 (P1)
[] ( P1_start => <> P1_end ),
[]~( P1_start & P1_end ),
[] ( P1_end => <> And_Junction ),
[]~( P1_end & And_Junction ),
% AndJunction And_Junction (And Junction)
[] ( And_Junction => <> ( P2_start & P3_start )),
[]~( And_Junction & P2_start ),
[]~( And_Junction & P3_start ),
% BusinessProcess P2 (P2)
[] ( P2_start => <> P2_end ),
[]~( P2_start & P2_end ),
[] ( P2_end => <> And_Junction_0 ),
[]~( P2_end & And_Junction_0 ),
% BusinessProcess P3 (P3)
[] ( P3_start => <> P3_end ),
[]~( P3_start & P3_end ),
[] ( P3_end => <> And_Junction_0 ),
[]~( P3_end & And_Junction_0 ),
% AndJunction And_Junction_0 (And Junction)
[]( P2_end & P3_end ) => <> And_Junction_0 ),
[] ( And_Junction_0 => <> P4_start ),
[]~( And_Junction_0 & P4_start ),
% BusinessProcess P4 (P4)
[] ( P4_start => <> P4_end ),
[]~( P4_start & P4_end ),
```

Figure 8. An example of ANDJunction used to implement parallel execution pattern (top) and corresponding LTL formulas (bottom)

Source: own elaboration.

5.5. Discussion

In this section, the formal rules governing the translation of the basic ArchiMate behavioral elements of business layer into LTL formulas were defined. It should be noted that ArchiMate syntax strictly defines how elements of various types can be combined, however without giving semantics to them (at least in cases of behavioral elements). Hence, the rules can be considered to be an assignment of semantics to the language patterns.

After an analysis of several examples, we decided not to define the rules for elements composed of lower level activities. The ArchiMate syntax seems to manifest some flaws in this case. An activity modeled as a process, function or interaction can be a part (as defined by the composition relation) of several high-level behavioral elements. Moreover, a complex process can be composed of lower level activities, but junctions and internal events are not included into the composition. Therefore we decided to treat complex processes as kinds of views helping to organize the models, rather than manageable entities.

6. Deduction-based verification

System specification in the form of LTL formulas $F_1, \ldots, F_n$ obtained by applying rules defined in the previous section can be checked for either its validity or entailment: $F_1, \ldots, F_n \vDash G$. The second case is particularly interesting, as G can express a desired system property pertaining to the temporal ordering of states and events. The approach proposed in this work consists in applying a semantic tableaux method to reason about entailment. The method is described briefly in Section 6.1, which is followed by Section 6.2 giving an outline of a verification system architecture. Finally we present an example of specification in Section 6.3.

6.1. Semantic tableaux method

Semantic tableaux is a decision-making procedure for checking satisfiability of a formula. To do so, it shows that the negation of an initial formula cannot be satisfied, hence, the initial formula is a tautology. To verify an entailment $F_1, \ldots, F_n \vDash G$ it suffices to prove that $\{F_1, \ldots, F_n, \neg G\}$ is unsatisfiable.

The main principle of propositional tableaux is to "break" complex formulae into smaller ones until complementary pairs of literals are produced or no further expansion is possible The method originates from the classical logic, but it can be also used for temporal logics [d'Agostino et al. 1999]. Generally speaking, the method is based on well-defined rules of formula decomposition and expansion. They allow to handle each of the logical connectives. When the rules are applied, branches of the inference tree are built. They correspond to alternatives appearing in formulas placed at the tree nodes. The inference tree is completed when no formula

can be further broken down, i.e. no complementary pairs of literals can be produced. The branches in the tree can be of two kinds: open or closed. A branch is closed if it can be established that a set of literal formulas, i.e. atomic formulas or its negations, on this branch has no model. In practice, this corresponds to a condition that a pair of contradictory formulas can be found on the branch. If all branches of the tree have contradictions, the whole inference tree is closed. If the negation of the initial formula is placed in the root, this leads to the statement that the initial formula is true. A very simple, yet illustrative example of the reasoning tree, is shown in Figure 10. The negation of the initial formula $(a \Rightarrow \Diamond b) \wedge (b \Rightarrow \Diamond c) \Rightarrow (a \Rightarrow \Diamond c)$ is placed in the root of the tree. All branches are closed (red nodes) which means that the initial formula is always satisfied.

6.2. Deduction-based verification system

The architecture of the deduction-based verification system is shown in Figure 9. The system consists of three components:

1) *Modeler* which allows to prepare and develop business models using ArchiMate language. In this case the Archi software, an excellent free modeling tool [Archi 2013] was used.

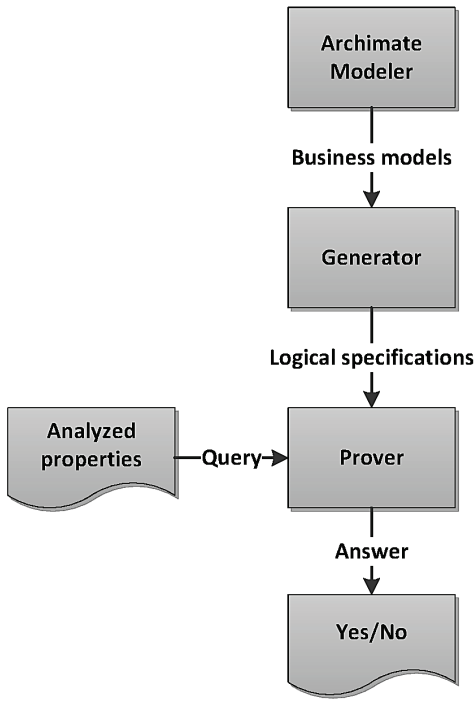


Figure 9. An architecture of deduction system

Source: own elaboration.

2) *Generator* which generates logical specifications from ArchiMate models. We have implemented a component that applies the rules described in Section 5 to elements of a business layer and yields a set (a conjunction) of LTL formulas. It is deployed as a plugin to the Archi modeler.

3) *Prover,* takes as input logical specifications (a set of temporal logic formulas describing a verified system) and a query, i.e. an examined property represented by a single formula, checks its validity and issues a response (*Yes* or *No*).

The prover is a crucial component of the verification system. Recently, a prototype reasoning engine for linear and future time minimal temporal logic was implemented[2], c.f. Figure 10. It allows to examine logical validity for formulas expressing liveness or safeness, as described above. Internally, the prover applies the semantic tableaux method customized to the requirements of reasoning on validity of LTL formulas.

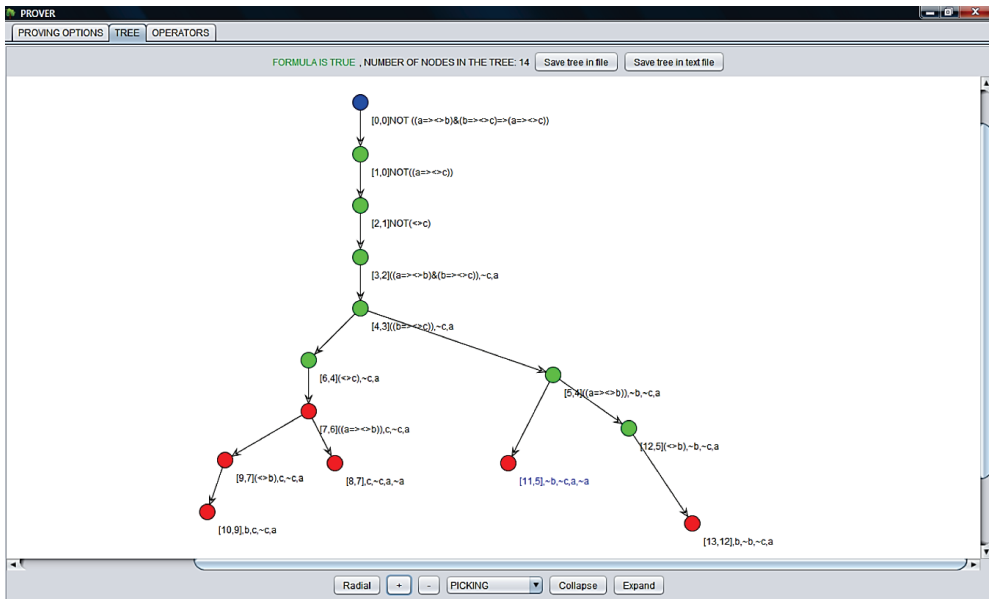


Figure 10. A prototype system of inference using the semantic tableaux method

Source: own elaboration.

An advantage of the described system (Figure 9) is that it can give an instantaneous response whenever the specification of a model is changed or when there is a need for a new inference due to a newly introduced property.

[2] The engine was implemented as a student project under supervision of one of the authors of this work.

6.3. Example of verification

In this section we return to the ArchiMate model of apprehending an object violating restricted area as presented in Section 4. For this model, 71 temporal formulas were generated. Due to the limited size of the work it is not possible to show them all. Thus, a subset of the whole logical specification Γ is shown below.

```
% BusinessProcess Update_intervention_description_and_send_a_new_group (Update
intervention description and send a new group)
[]( Update_intervention_description_and_send_a_new_group_start => <> Update_
intervention_description_and_send_a_new_group_end ) &
[]~( Update_intervention_description_and_send_a_new_group_start & Update_
intervention_description_and_send_a_new_group_end ) &
[] ( Update_intervention_description_and_send_a_new_group_end => <> Alarm___
zone_violation_1 ) &
[]~( Update_intervention_description_and_send_a_new_group_end & Alarm___zone_
violation_1 ) &
% BusinessInteraction Confirm_starting_intervention (Confirm starting intervention)
[] ( Confirm_starting_intervention_start => <> Confirm_starting_intervention_end ) &
[]~( Confirm_starting_intervention_start & Confirm_starting_intervention_end ) &
[] ( Confirm_starting_intervention_end => <> Localize_and_take_over_the_object_
start ) &
[]~( Confirm_starting_intervention_end & Localize_and_take_over_the_object_start ) &
% BusinessProcess Assign_intervention_group (Assign intervention group)
[] ( Assign_intervention_group_start => <> Assign_intervention_group_end ) &
[]~( Assign_intervention_group_start & Assign_intervention_group_end ) &
[] ( Assign_intervention_group_end => <> Confirm_starting_intervention_start ) &
[]~( Assign_intervention_group_end & Confirm_starting_intervention_start ) &
% BusinessProcess Try_to_contact_by_other_means (Try to contact by other means)
[] ( Try_to_contact_by_other_means_start => <> Try_to_contact_by_other_means_
end ) &
[]~( Try_to_contact_by_other_means_start & Try_to_contact_by_other_means_end ) &
[] ( Try_to_contact_by_other_means_end => <> ( ( R.1___Waiting_time_T1_elapsed
& ~R.1___Communication_established ) | ( ~R.1___Waiting_time_T1_elapsed &
R.1___Communication_established ) )) &
[]~( Try_to_contact_by_other_means_end & R.1___Waiting_time_T1_elapsed ) &
[]~( Try_to_contact_by_other_means_end & R.1___Communication_established ) &
% BusinessProcess Input_request_to_the_system (Input request to the system)
[] ( Input_request_to_the_system_start => <> Input_request_to_the_system_end ) &
[]~( Input_request_to_the_system_start & Input_request_to_the_system_end ) &
[] ( Input_request_to_the_system_end => <> Assess_the_situation_start ) &
[]~( Input_request_to_the_system_end & Assess_the_situation_start ) &
% BusinessEvent R.1___Communication_established (R.1 - Communication established)
[] ( R.1___Communication_established => <> R.1___Stop_0) &
[] ~( R.1___Communication_established & R.1___Stop_0) &
% Junction Junction (Junction)
[] ( Junction => <> ( ( To_the_police___take_the_detainee & ~R.1___Terminate )
| ( ~To_the_police___take_the_detainee & R.1___Terminate ) )) &
[]~( Junction & To_the_police___take_the_detainee ) &
[]~( Junction & R.1___Terminate ) &
% BusinessEvent To_the_police___request_for_support (To the police - request for
support)
```

Figure 11. A subset of logical specification of the verified system

Source: own elaboration.

Let us consider a liveness property expressed formally by the following formula:

$$\begin{aligned} &\Box\Big(Alarm\_zone\_violation \\ &\Rightarrow \Diamond\big((To\_the\_police\_take\_the\_detainee \wedge \neg R.1\_Terminate) \\ &\vee (\neg To\_the\_police\_take\_the\_detainee \wedge R.1\_Terminate)\big)\Big) \end{aligned} \tag{3}$$

which can be understood that, *if a zone violation alarm is triggered then sometime in the future only one of the following events will be triggered: the police will take the detainee or the terminate event will be triggered*.

When analyzing if a specification Γ satisfies the property expressed by the formula (3), a new formula (4), is constructed and submitted to the prover. (Γ)

$$\begin{aligned} &\Gamma \Rightarrow \Big(\Box\Big(Alarm\_zone\_violation \\ &\Rightarrow \Diamond\big((To\_the\_police\_take\_the\_detainee \wedge \neg R.1\_Terminate) \\ &\vee (\neg To\_the\_police\_take\_the\_detainee \wedge R.1\_Terminate)\big)\Big)\Big) \end{aligned} \tag{4}$$

A presentation of a full inference tree which contains more than two thousand nodes, would exceed the size of the work. All branches of the semantic trees are closed, i.e. formula 4 is satisfied in the considered model. If the tree had open branches, this would indicate that the input formula cannot be satisfied. In this case the prover would provide information about the source of the error, which can be considered an important advantage of the method.

7. Related work

The recent work by Morimoto [Morimoto 2008] surveys the formal verification tools for business processes. It discusses, in the context of business process management, the applications of such formalisms as automata, model checking, process algebras and Petri nets. The described approaches can be considered as variations of either model checking or simulation. In particular, model checking seems to be the most often used. There are several reports on the application of the model checking approach, e.g. to perform a verification of e-business processes [Anderson et al. 2005], or BPMN models extended with resource constraints [Watahiki et al. 2011]. In the work by Deutsch et al. [Deutsch et al. 2009], the verification of data-centric business processes is studied. 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. A salient consequence of modeling operations on data are infinite domains. Hence, the problem of correctness verification can be undecidable.

An application of CTL to the verification of BPEL processes was reported in the work by Mongiello and Castelluccia [Mongiello, Castelluccia 2006]. Three types of

correctness properties were analyzed: invariants, properties of final states and temporal relations between activities. The first two can be classified as the safeness, the last as the liveness property. Similarly, in the work by Fu et al. [Fu et al. 2002], CTL was applied to the verification of e-services and workflows with both a bounded and unbounded number of process instances.

The application of deduction-based approach is rare in the area of business models verification. The work by Shankar [Shankar 2009] contains a comprehensive study for the area of verification using automated deduction and deduction-based techniques. To the best of our knowledge, no attempts have been made to define formally semantics and perform a verification for the behavioral elements of ArchiMate. Some suggestions and research direction can be found in an early document [De Boer et al. 2003]. On the other hand, in a few publications [Ettema, Dietz 2009; Azevedo et al. 2011] ontologies were applied to define semantics for subsets of ArchiMate elements and relations. However, all of the research themes mentioned above are different from the approach presented in the work.

8. Conclusion

This paper presents a new framework for the automated verification of the behavioral aspects of ArchiMate models. The main contributions of our approach are the translations rules from the ArchiMate model into LTL formulas and an application of deductive reasoning and semantic tableaux methods. More tangible results of our work are the following: a plugin into ArchiMate modeler allowing for automatic LTL formula generation and an LTL prover used here for system verification purposes. Our approach fits into the general scheme of system verification. It assumes that the system under consideration is translated into a set of formulas in a certain symbolic logic, and then is verified whether another formula (called specification of the system) can be entailed from this set of formulas. The usefulness of such verification approaches largely depends on whether translation rules adequately represent the semantics of selected aspect of the verified system.

Although the considerations in this work are focused on deductive reasoning and the semantic tableaux method, automatically generated LTL specifications can be verified with other methods, e.g. the resolution method.

The defined set of rules for transforming ArchiMate models into LTL formulas considers only atomic processes and functions. It is an open question how to give semantics to explicitly specified high-level behavioral elements aggregating low-level behaviors. At present they are treated as views of organizing models, however we are analyzing alternative approaches.

References

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

Archi, *ArchiMate modelling tool*, 2013, http://archi.cetis.ac.uk/download.html [accessed: 23 April 2013].

Azevedo C., Almeida J., Van Sinderen M., Quartel D., Guizzardi G., *An ontology-based semantics for the motivation extension to ArchiMate*, [in:] *Enterprise Distributed Object Computing Conference (EDOC)*, 2011 15th IEEE International, 2011, pp. 25-34.

Clarke E., Grumberg O., Peled D., *Model Checking*, MIT Press, 1999.

Clarke E., Wing J. et al., *Formal methods: State of the art and future directions*, "ACM Computing Surveys" 1996, vol. 28 (4), pp. 626-643.

d'Agostino M., Gabbay D., Hähnle R., Posegga J., *Handbook of Tableau Methods*, Kluwer Academic Publishers, 1999.

De Boer F., Bonsangue M., der Doest H., Groenewegen L., Jonkers H., Stam A., van der Torre L., *Analysis of Enterprise Architectures (q4)*, 2003, https://doc.telin.nl/dscgi/ds.py/Get/File-31618 [accessed: 15 June 2013].

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

Emerson E., *Handbook of Theoretical Computer Science*, Elsevier, MIT Press, 1990, vol. B, *Temporal and Modal Logic*, pp. 995-1072.

Ettema R., Dietz J., *ArchiMate and DEMO − Mates to date?,* [in:] *Advances in Enterprise Engineering III*, ser. Lecture Notes in Business Information Processing, eds. A. Albani, J. Barjis, J. Dietz, Springer Berlin Heidelberg, 2009, vol. 34, pp. 172-186, http://dx.doi.org/10.1007/978-3-642-01915-9_13 [accessed: 20 June 2013].

Wolter F., Wooldridge M., *Temporal and dynamic logic*, "Journal of Indian Council of Philosophical Research" 2011, vol. XXVII(1), pp. 249-276.

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

Gruninger M., Fox M.S., *An activity ontology for enterprise modelling*, Department of Industrial Engineering, University of Toronto, 1994.

Huth M., Ryan M., *Logic in Computer Science. Modelling and Reasoning about Systems*, Cambridge University Press, 2004.

Mongiello M., Castelluccia D., *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, p. 5.

Morimoto S., *A survey of formal verification for business process modeling*, [in:] *Proceedings of the 8th International Conference Computational Science (ICCS 2008)*, June 23-25, 2008, Kraków, Poland, Part II, ser. Lecture Notes in Computer Science, eds. M. Bubak, G.D. van Albada, J. Dongarra, P.M.A. Sloot, vol. 5102, Springer Verlag, 2008, pp. 514-522.

Nick M., *Will there be a battle between ArchiMate and the UML?*, 2009, http://blogs.msdn.com/b/nickmalik/archive/2009/04/17/will-there-be-a-battle-between-ArchiMate-and-the-uml.aspx [accessed 10 June 2013].

OMG, *Business Process Model and Notation (BPMN) version 2.0*, OMG, Tech. Rep., January 2011. http://www.omg.org/spec/BPMN/2.0 [accessed 10 June 2013].

Rumbaugh J., Jacobson I., Booch G., *Unified Modeling Language Reference Manual*, 2nd edition. Pearson Higher Education, 2004.

Scheer A., *Aris – Business Process Modeling*, ser. ARIS – Business Process Modeling, Springer, 1999.
Scheer A.-W., Nüttgens M., *ARIS architecture and reference models for business process management*, [in:] Business Process Management, Springer, 2000, pp. 376-389.
Shankar N., *Automated deduction for verification*, "ACM Computing Surveys" 2009, vol. 41 (4), pp. 20:1-20:56.
Szwed P., Chmiel W., Jedrusik S., Kadluczka P., *Business processes in a distributed surveillance system integrated through workflow*, "Automatyka/Automatics", in press, 2013.
The Open Group, *Open Group Standard*, *ArchiMate 2.0 Specificattion*, 2012, http://www.opengroup.org [accessed: 11 July 2013].
The Workflow Management Coalition, *Process Definition Interface – XML Process Definition Language, Workflow Management Coalition Workflow Standard, version 2.1a*, The Workflow Management Coalition, Tech. Rep., 2008, http://www.wfmc.org/Download-document/.
Van Benthem J., *Handbook of Logic in Artificial Intelligence and Logic Programming*, ser. 4. Clarendon Press, 1993-95, ch. Temporal Logic, pp. 241-350.
Van Den Berg H., Bosma H., Dijk G., Van Drunen H., Van Gijsen J., Langeveld F., Luijpers J., Nguyen T., Oosting R., Gerand Slagter and et al., *ArchiMate made practical*,Work, 2007.
Van der Aalst W.M., Desel J., Kindler E., *On the semantics of EPCs: A vicious circle*, [in:] *Proceedings of the EPK*, 2002, pp. 71-80.
Watahiki K., Ishikawa F., Hiraishi K., *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.
WFMC-TC-1025-Oct-10-08-A-Final-XPDL-2.1-Specification.html [accessed: 11 July 2013].

ZASTOSOWANIE WNIOSKOWANIA DEDUKCYJNEGO DO WERYFIKACJI OPISÓW ZACHOWANIA W JĘZYKU ARCHIMATE

Streszczenie: Formalna weryfikacja modeli biznesowych stała się ostatnio przedmiotem intensywnych badań. Oczekuje się, że zastosowanie metod formalnych może przynieś takie korzyści, jak zwiększenie jakości produktów i usług oraz zmniejszenie liczby błędów operacyjnych. W pracy omówiono zastosowanie metody wykorzystującej wnioskowanie dedukcyjne do weryfikacji elementów behawioralnych w modelach języka ArchiMate. Pierwszy krok zaproponowanej metody polega na translacji modelu ArchiMate do postaci formuł liniowej logiki temporanej (LTL). Następnie weryfikuje się, czy spełniają one założone własności temporalne. W procesie weryfikacji używane jest narzędzie dowodzenia, w którym zastosowano technikę tablic semantycznych. Opisując metodę weryfikacji, wykorzystano przykład procesu biznesowego zaimplementowanego w systemie nadzoru.

Słowa kluczowe: wnioskowanie dedukcyjne, weryfikacja oprogramowania, ArchiMate, metoda tablic semantycznych, liniowa logika temporalna.