Proceedings of the 2013 Federated Conference on
Computer Science and Information Systems pp. 1103–1110

Verification of ArchiMate process specifications based on deductive temporal reasoning

Radosław Klimek* and Piotr Szwed*
*AGH University of Science and Technology
E-mail: {rklimek,pszwed}@agh.edu.pl

***Abstract*—Formal verification of business models has become recently an intensively researched area. Application of formal methods in this field necessities in overcoming several problems. Firstly, business analyst and designers rarely have enough skills and motivation to manually build abstract and formal specifications, hence, it arises the need to provide tools for an automated translation of business models into a suitable form ready for formal verification. Moreover, notations and languages used to describe enterprises usually have no clear semantics. Finally, the verification itself must be supported by an efficient tool. In this paper we investigate an application of formal and deduction-based techniques to automated verification of behavioral description embedded within ArchiMate models. We describe a set of rules that governs translation of processes specified in ArchiMate language into Linear Temporal Logic (LTL) formulas. The translation step is achieved with the developed software, as a plugin into a popular the Archi modeler. Formal verification of a business process properties is achieved with another tool, the LTL prover based on the semantic tableaux technique. Application of the method is discussed on a small, yet illustrative, example of a taxi service.**

I. Introduction

Formal verification of business models has become recently an intensively researched topic. The growing interest in this area stems to some extend form a historical fact. Issued in 2000 revisions to ISO 9001 and 9004 norms recommended process-oriented approach to quality management. Since then, many organizations have started to identify and describe their processes to fulfill certification requirements, but also they have realized that coherently and unambiguously specified business processes can bring such benefits, as quality of products and services, smaller ratio of operational errors, reduced cost and greater competitiveness.

In this paper we investigate an application of formal deduction-based techniques to automated verification of behavioral description embedded within ArchiMate models. ArchiMate is an lightweight and scalable language supporting analysis and modeling of business and enterprise domains [23]. It has been developed to provide a uniform representation for architecture descriptions. The popularity of ArchiMate language within the enterprise architecture modeling community grows.

Formal methods are understood as a set of principles for precise formulation of important artifacts formed when developing and refining software models. They enable revealing, questioning and removing ambiguities and flaws in specifications. Moreover, the formality of used languages leads to the rigorous analysis and verification. There are two established approaches to formal reasoning and system verification [5], i.e. model checking and deductive reasoning. Model checking is based on the state exploration and is an operational rather than analytic approach [4]. Deductive inference enables analysis of infinite computation sequences.

Temporal logic TL 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". Temporal logic is a well-established formalism for specification and verification of reactive and concurrent systems. It allows to describe both temporal relations between reached states or events occurring within a system and to specify 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. [9].

The properties of time structure are fundamental to a logic. Of particular significance is the *minimal temporal logic*, e.g. [25], also known as temporal logic of the class K. The 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 $\vdash P \implies \vdash \Box P$. 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: $action \Rightarrow \Diamond reaction$, $\Box(send \Rightarrow \Diamond receive)$, $\Diamond alive$, $\Box\neg(badevent)$, 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

978-1-4673-4471-5/$25.00 © 2013, IEEE

to build a deduction engine, or use the existing verified provers.

Application of deductive approach to validation of business processes faces the problem of automatic obtaining 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 of a large collection of formulas is difficult and monotonous, c.f. also the requirements engineering process [13].

For temporal logic, that is a suitable language for expressing behavior and reasoning about it, such specifications are constituted by set of temporal logic formulas $\{F_1, ..., F_n\}$ When the number of formulas is large, what 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 form a pen-and-paper approach to engineers' needs an automation of the generation process seems particularly important.

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 are the following: rules for 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. 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: the next Section II discusses approaches to verification of business models, it is followed by Section III briefly describing ArchiMate language. Then, in Section IV an example of a process specification using ArchiMate is provided. Section V defines rules governing translation of ArchiMate models to LTL formulas. In Section VI an architecture of the verification system is described and an example of a checked property is given. Finally Section VII gives concluding remarks.

II. Related work

Recent work by Morimoto [15] surveys formal verification tools for business processes. It discusses in the context of business process management application 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 application of model checking approach, e.g. to perform verification of of e-business processes, work by Anderson et al. [2], or BPMN models extended with resource constraints, c.f. work by Watahiki et al. [28]. In work by Deutsch at al. [8] 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. Application of CTL to verification of BPEL processes was reported in work by Mongiello and Castelluccia [14]. 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. [11] CTL was applied to the verification of e-services and workflows with both bounded and unbounded number of process instances. Application of deduction based approach is rare in the area of business models verification. The work by Shankar [21] contains a comprehensive study for the area of verification using automated deduction and deduction-based techniques. Up to our best knowledge, no attempts has been made to define formally semantics and perform verification for behavioral elements of ArchiMate. Some suggestions and research direction can be found in an early document [7]. On the other hand, in a few publications [10], [3] 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.

III. Archimate

ArchiMate [23], [26] is a contemporary, open and independent language for 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 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 [18], which can be seen as a collection of unrelated diagrams, and Business Process Modeling Notation BPMN [17] which covers mainly behavioral aspect of enterprise architecture. The definition of a language 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. ArchiMate goes beyond UML [16]: 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 be-

tween 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 [17], after adopting a certain modeling convention its expressiveness can be similar [22]. 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) [20], [19]. Indeed, 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). In spite of almost 20 years presence of EPC tools on a market and thousands of deployments in modeling business organizations, there is no consensus of semantics of EPCs. Analyzes of several semantics variations have revealed certain erroneous patterns, e.g the famous vicious circle [27] resulting in a deadlock caused by 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.

IV. Example of a business model

In this section we give an example of ArchiMate model for a small business process defining Taxi service allowing a client to order a taxi that will carry him or her to the desired destination.

The example shows typical constructs that can be used to model business behavior in ArchiMate. We assume that each high-level process starts and ends with an event. A triggering event originate from the environment, e.g. *Client trip order* in Fig. 1 or *Order cancellation request* in Fig. 4. A complex process can be decomposed into several subprocesses, which are usually presented in separate views. Such processes are linked by intermediate events, e.g. *Trip accepted* appearing on the output of a subprocess in Fig. 1 and on the input of Fig. 3b. Finally, a process stops at *end events*, which may trigger external processes or terminate a thread of execution, as events *Stop 1 – Stop 7*.

For clarity, we omitted in diagrams additional information, which is usually present in ArchiMate models of business layer, e.g. assigned roles and accessed business objects, functions, services, actors and locations. ArchiMate specifications can be much richer, what justifies common practice of decomposing behavior descriptions into smaller patterns presented in multiple views. In this way specifications can cover various important aspects of an enterprise architecture, but control flows become harder to be verified manually, as they require switching between multiple views. In fact, specifications presented in Fig. 1–4 are excerpts restricted to behavioral elements presented in eight views.

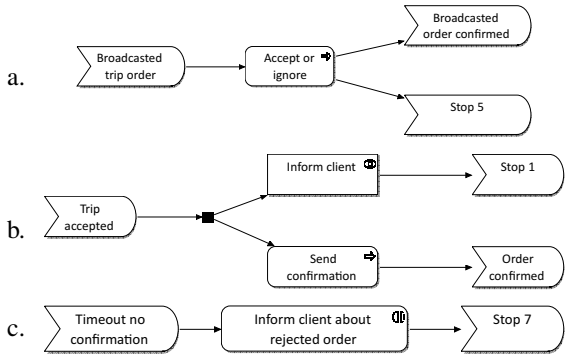

Fig. 3. Extensions to the main operator processes: a. Broadcasting trip order; b. Reaction on accepted order; c. Handling timeout (an order cannot be realized) Broadcast

The main flow of activities within dispatcher's office is shown in Fig. 1.

When a client requests a taxi (*Client trip order*), then the order is handled by a dispatcher. Firstly, all taxicab stands located closest to the present customer's position are searched. If contact with the driver (*Contact taxi driver*), and details of the order, leads to its rejection (*Trip rejected*), then the next taxicab form stands is selected. If a free taxi is found and the trip request is accepted (*Trip Accepted*), then the customer and the driver are provided with information about each other as well as trip's details, and the trip order is stored in the database as approved. If *Look for free taxi cabs* process takes too long time (*Timeout*), then dispatcher broadcasts the trip order to all drivers (in the city area). Such order may also be accepted on the general principles.

Main process of the driver is shown in Fig. 2.

After an order confirmation, the agreed driver goes to the the trip start location (*Reach location*) to pick up the passenger (*Pick up passenger*). Along the way to the pick-up location difficulties may arrise, e.g. heavy traffic or passenger absence. After the completion of the ride (*Trip finished*), its status is changed from approved to finished.

Extensions of the dispatcher processes are shown in Fig. 3 and discussed below.

(a) Waiting for the first declaration which originates from the broadcast.
(b) A kind of a virtual handshake between a taxicab driver (*Send confirmation*) and the passenger (*Inform client*) with an intermediary of dispatcher.
(c) If none of the methods result, the customer is notified and the order is rejected (*Timeout no confirmation*).

Further extensions of the dispatcher processes are shown in Fig. 4 and discussed below.

(a) If there are obstacles when reaching the client, then he/she should be informed about this fact.
(b) When order is canceled, the driver must be notified.

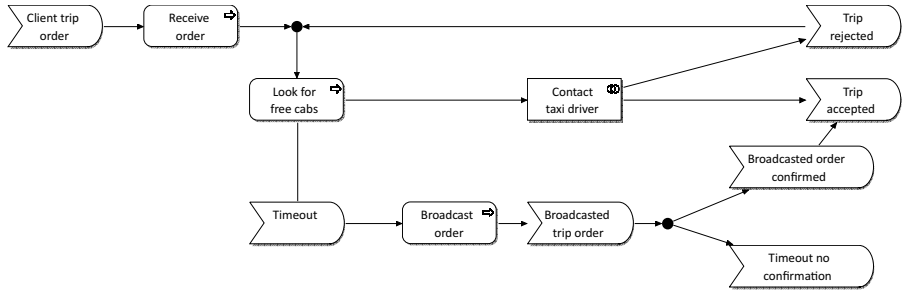

Fig. 1. Main dispatcher process. After receiving a client order, operator looks for registered free cabs. If not found, broadcasts information about the order

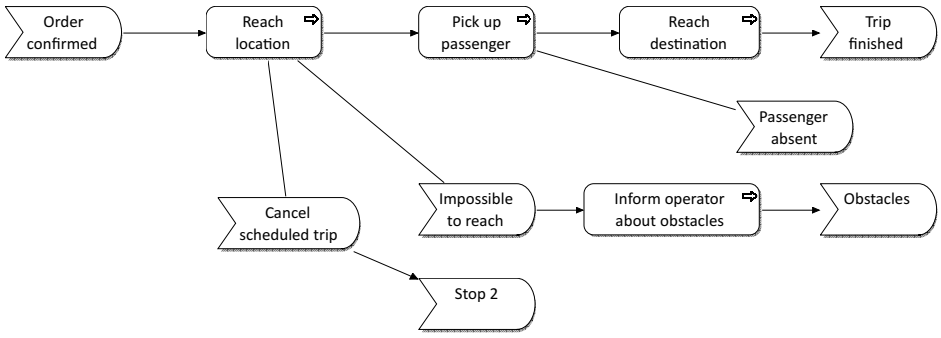

Fig. 2. Main driver process.

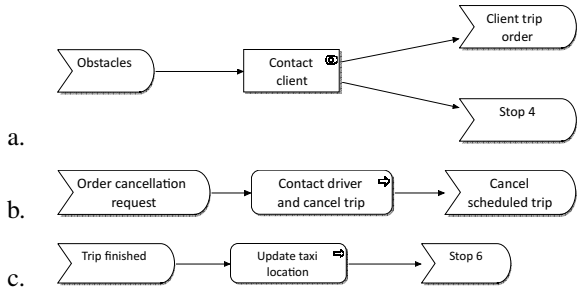

Fig. 4. Further extensions to the main operator processes: a. Contacting client after a taxi driver reports obstacles, e.g. delays ; b. Handling cancellation; c. updating information on taxi location after a finished trip

(c) When the trip is successfully finished, then the new taxicab location is introduced.

V. Modeling Archimate behavioral constructs

This section gives formally defined rules for 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 a type of element or relation. Generating LTL formulas describing behavioral aspects of Archimate model we focus on 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 LTL formulas describing control flow.

It should be noted that Archimate behavioral constructs have no precisely defined semantics. In fact, translation from 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 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 the considerations in the work focus on business layer elements used to specify behavior, it is assumed that $C = \{Process, Function, Interaction, Event, Junction, And\text{-}Junction, OrJunction, Other\}$ and $R = \{triggering, association, composition, other\}$.

A. Modeling atomic activities

By 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 [12]. While defining

LTL formulas describing processes and other elements, we follow directions of relations and specify only transitions between internal states of elements and caused states. In turn, the reached caused states enable activation of other elements. Hence, after processing all relevant elements, a complete network of states of the whole system specified in LTL is obtained.

To model execution of an atomic process two states (and corresponding propositions in LTL): *start* and *end* are used. A process is considered *imperfect*, even if has a name in imperative mood suggesting achievement, e.g. *register invoice*, *scan document*, *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 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 events triggered upon process completion and interrupting a normal flow.

Fig. 5 illustrates this approach. The *end* state and *Interrupting event* are successor of the process *start* state. On the other hand, states of surrounding elements that can be reached by the normal triggering relation are successors of the *end* state.

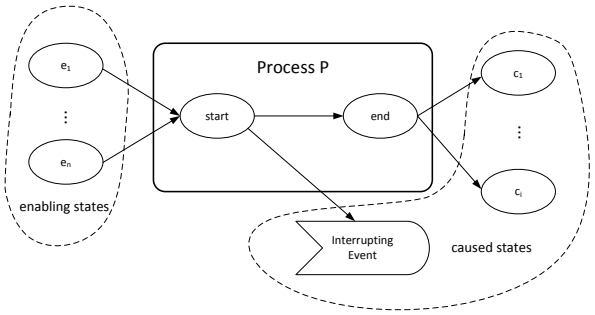

Fig. 5. Two states: *start* and *end* used to model a process

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 an 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 other elements: events and junctions $start(v) = end(v)$ holds. We extend these functions to sets of vertices, i.e. $start(X) = \bigcup_{v \in X} start(v)$ and $end(X) = \bigcup_{v \in X} end(v)$.

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. $A(v) = \{v' : vt(v') = (Event) \wedge (v, v') \in E \wedge et(v, v') = association\}$ 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 parentheses. For the brevity of notation we will further omit $Props$ and write simply $\mathcal{F}$

We define two auxiliary functions $\delta_{ij}(p)$ mapping formulas $\mathcal{F} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{F}$ (1) and $oneof(P)$ converting a set of propositions P into a formula in disjunctive normal form (2).

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

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

B. 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 similar form. Each rule contains precondition part separated from its postcondition by a horizontal line. Generated formulas are placed in 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 *Look for free cabs* and *Contact taxi driver* in Fig. 1 are presented in Fig. 6 (original transcription is preserved). They were generated according to Rule 1.

```
% BusinessProcess (Look for free cabs)
[] (Look_for_free_cabs_start =>
   <> ((Look_for_free_cabs_end & ~Timeout)
   |  (~Look_for_free_cabs_end & Timeout))),
[]~(Look_for_free_cabs_start & Look_for_free_cabs_end),
[]~(Look_for_free_cabs_start & Timeout),
[] (Look_for_free_cabs_end => <>Contact_taxi_driver_start),
[]~(Look_for_free_cabs_end & Contact_taxi_driver_start),
% BusinessInteraction (Contact taxi driver)
[] (Contact_taxi_driver_start => <>Contact_taxi_driver_end),
[]~(Contact_taxi_driver_start & Contact_taxi_driver_end ),
[] (Contact_taxi_driver_end =>
   <> ((Trip_accepted & ~Trip_rejected)
   |  (~Trip_accepted & Trip_rejected ))),
[]~(Contact_taxi_driver_end & Trip_accepted),
[]~(Contact_taxi_driver_end & Trip_rejected),
```

Fig. 6. An excerpt of generated formulas for the main dispatcher process

C. Event

According to Archimate specification [23] business event is something that happens and influences behavioral elements

(processes, functions and interactions). 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. An occurrence or the both activates all elements linked by a *triggering* relation. However, we assume that, unlike AndJunctions, activation of elements triggered by an event is not synchronized (c.f. Rule 2).

Rule 2. Event

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

D. Junctions

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 has counterparts in EPC, BPMN (exclusive, inclusive and parallel gateways) and XPDL transition restrictions [24].

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

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

Rule 4. OrJunction

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

Define: $T^{-1}(v) = \{v' : (v', v) \in E \wedge et(v', v) = triggering\}$ - set of input

Rule 5. AndJunction

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

E. Discussion

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

After analysis of several examples we decided not to define rules for elements composed of lower level activities. 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.

VI. Deduction-based verification

System specification in form of LTL formulas $F_1, ...F_n$ obtained by applying rules defined in previous section can be checked for either its validity or entailment: $F_1, ...F_n \models G$. The second case is particularly interesting, as G can express a desired system property pertaining to 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 VI-A, which is followed by Section VI-B giving an outline of a verification system architecture. Finally we present an example of specification in Section VI-C.

A. 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 $F_1, ...F_n \models G$ it suffice to prove that $\{F_1, ..., F_n, \neg G\}$ 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 form classical logic but it can be also used for temporal logics [6]. 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. An inference tree is *finished*, 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 Fig. 8. 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) what means that the initial formula is always satisfied.

B. Deduction based verification system

The architecture of the deduction-based verification system is shown in Fig. 7. The system consists of three components:

1) *Modeler* that allows to prepare and develop business models using ArchiMate language. In this case the Archi software, an excellent free modeling tool [1] was used.
2) *Generator* generates logical specifications from ArchiMate models. We have implemented a component that applies rules described in Section V to elements of a business layer and yields a set (a conjunction) of LTL formulas. It is deployed as a plugin to Archi.
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*).

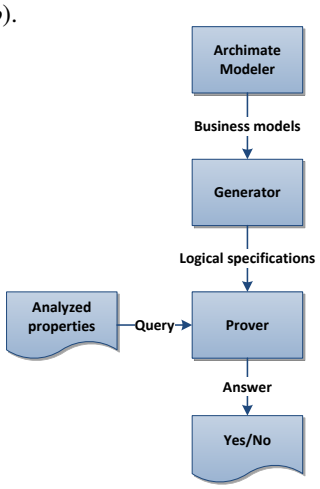

Fig. 7. An architecture of the deduction system

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[1], c.f. Fig. 8. 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 requirements of reasoning on validity of LTL formulas.

An advantage of the described system (Fig. 7) is that it can give instantaneous response whenever the specification of a model is changed or there is a need for a new inference due to a newly introduced property.

[1]The engine was implemented as a students' (Joanna Kulesza and Kamil Łopata) project under the supervision of one of the authors of the work.

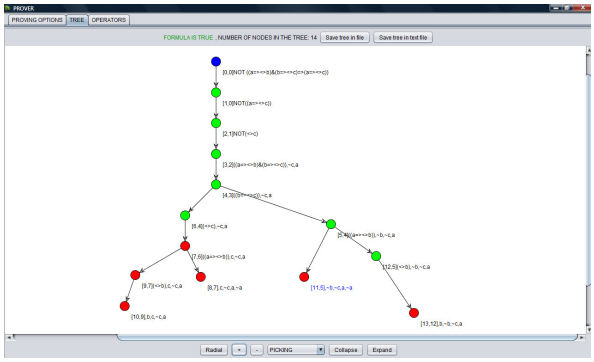

Fig. 8. A prototype system of inference using the semantic tableaux method

C. Example of verification

In this section we return to the ArchiMate model of a taxi service presented in Section IV. For this model 108 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 L referring to Fig. 1 and Fig 3.b-c, is shown below.

$$
\begin{aligned}
L = \{&\Box(Contact\_taxi\_driver\_start \Rightarrow \\
&\Diamond Contact\_taxi\_driver\_end), \Box\neg(Contact\_taxi\_driver\_end \wedge \\
&Trip\_accepted), \Box(Broadcasted\_trip\_order \Rightarrow \\
&\Diamond Junction\_0), \Box(Timeout \Rightarrow \\
&\Diamond Broadcast\_order\_start), \Box(Inform\_client\_start \Rightarrow \\
&\Diamond Inform\_client\_end), \Box(Inform\_client\_end \Rightarrow \\
&\Diamond Stop\_1), \Box(Timeout\_no\_confirmation \Rightarrow \\
&\Diamond Inform\_client\_about\_rejected\_order\_start), \\
&\Box(Trip\_accepted \Rightarrow \Diamond And\_Junction), \Box(Junction \Rightarrow \\
&\Diamond Look\_for\_free\_cabs\_start), \\
&\Box(Inform\_client\_about\_rejected\_order\_start \Rightarrow \\
&\Diamond Inform\_client\_about\_rejected\_order\_end), \\
&\Box(Inform\_client\_about\_rejected\_order\_end \Rightarrow \Diamond((Stop\_2 \wedge \\
&\neg Stop\_7) \vee (\neg Stop\_2 \wedge Stop\_7))), \Box(Client\_trip\_order \Rightarrow \\
&\Diamond Receive\_order\_start), \Box(And\_Junction \Rightarrow \\
&\Diamond(Inform\_client\_start \wedge \\
&Send\_confirmation\_start)), \Box(Look\_for\_free\_cabs\_start \Rightarrow \\
&\Diamond((Look\_for\_free\_cabs\_end \wedge \neg Timeout) \vee \\
&(\neg Look\_for\_free\_cabs\_end \wedge \\
&Timeout))), \Box(Look\_for\_free\_cabs\_end \Rightarrow \\
&\Diamond Contact\_taxi\_driver\_start), \Box(Receive\_order\_start \Rightarrow \\
&\Diamond Receive\_order\_end), \Box(Receive\_order\_end \Rightarrow \\
&\Diamond Junction), \Box(Broadcast\_order\_start \Rightarrow \\
&\Diamond Broadcast\_order\_end), \Box(Broadcast\_order\_end \Rightarrow \\
&\Diamond Broadcasted\_trip\_order), \Box(Junction\_0 \Rightarrow \\
&\Diamond((Broadcasted\_order\_confirmed \wedge \\
&\neg Timeout\_no\_confirmation) \vee \\
&(\neg Broadcasted\_order\_confirmed \wedge \\
&Timeout\_no\_confirmation))), \\
&\Box(Broadcasted\_order\_confirmed \Rightarrow \Diamond Trip\_accepted)\}
\end{aligned}
$$

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

$$\Box(Client\_trip\_order \Rightarrow \Diamond(Stop\_1 \vee Stop\_7)) \qquad (3)$$

which can be understood that, *if a client ordered a trip then sometime in the future the client is informed about assigned cab (Stop_1) or the order is rejected (Stop_7).*

When analyzing if a specification L satisfies the property expressed by the formula (3) a new formula (4) is constructed

and submitted to the prover.

$$C(L) \Rightarrow (\Box(Client\_trip\_order \Rightarrow \Diamond(Stop\_1 \vee Stop\_7))) \quad (4)$$

where $C(L)$ means logical conjunctions of formulas.

Presentation of a full inference tree, which contains more than a thousand nodes, exceeds 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 can be not satisfied. In this case the prover would provide information about the source of the error, what can be considered an important advantage of the method.

VII. Conclusion

This paper discusses a problem of automatic verification of behavioral specification embedded within ArchiMate models. We propose to apply an approach consisting in translating ArchiMate specification into a set of LTL formulas and using deductive reasoning technique to check temporal properties of the model. Firstly, on a small example we present language patterns that can be used to model processes, then rules for translation of ArchiMate behavior specification into LTL formulas are formally defined. Finally, we describe the architecture of the implemented reasoning system and show how it can be used to check desired system properties.

Although the considerations in this work are focused on deductive reasoning and 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 organizing models, however we are analyzing alternative approaches.

Acknowledgment

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

References

[1] "Archi, archimate modelling tool," 2013, [Online; accessed 23-April-2013]. [Online]. Available: http://archi.cetis.ac.uk/download.html

[2] 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.

[3] C. Azevedo, J. Almeida, M. Van Sinderen, D. Quartel, and G. Guizzardi, "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.

[4] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

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

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

[7] F. de Boer, M. Bonsangue, H. ter Doest, L. Groenewegen, H. Jonkers, A. Stam, and L. van der Torre, "Analysis of Enterprise Architectures (q4)," 2003. [Online]. Available: https://doc.telin.nl/dscgi/ds.py/Get/File-31618

[8] 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.

[9] E. Emerson, *Handbook of Theoretical Computer Science*. Elsevier, MIT Press, 1990, vol. B, ch. Temporal and Modal Logic, pp. 995–1072.

[10] R. Ettema and J. Dietz, "Archimate and demo – mates to date?" in *Advances in Enterprise Engineering III*, ser. Lecture Notes in Business Information Processing, A. Albani, J. Barjis, and J. Dietz, Eds. Springer Berlin Heidelberg, 2009, vol. 34, pp. 172–186. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01915-9_13

[11] 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.

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

[13] R. Klimek, "From extraction of logical specifications to deduction-based formal verification of requirements models," in *Proceedings of 11th International Conference on Software Engineering and Formal Methods (SEFM 2013), 25–27 September 2013, Madrid, Spain*, ser. Lecture Notes in Computer Science, R. M. Hierons, M. G. Merayo, and M. Bravetti, Eds., vol. 8137. Springer Verlag, 2013, pp. 61–75. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40561-7_5

[14] 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.

[15] S. Morimoto, "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, M. Bubak, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5102. Springer–Verlag, 2008, pp. 514–522.

[16] M. Nick, "Will there be a battle between Archimate and the UML?" 2009. [Online]. Available: http://blogs.msdn.com/b/nickmalik/archive/2009/04/17/will-there-be-a-battle-between-archimate-and-the-uml.aspx

[17] 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

[18] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

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

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

[21] N. Shankar, "Automated deduction for verification," *ACM Computing Surveys*, vol. 41 (4), pp. 20:1–20:56, 2009.

[22] P. Szwed, W. Chmiel, S. Jedrusik, and P. Kadluczka, "Business processes in a distributed surveillance system integrated through workflow," *Automatyka/Automatics*, no. to appear, 2013.

[23] The Open Group, "Open Group Standard. Archimate 2.0 Specificattion," 2012. [Online]. Available: http://www.opengroup.org

[24] 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. [Online]. Available: http://www.wfmc.org/Download-document/WFMC-TC-1025-Oct-10-08-A-Final-XPDL-2.1-Specification.html

[25] J. van Benthem, *Handbook of Logic in Artificial Intelligence and Logic Programming*, ser. 4. Clarendon Press, 1993–95, ch. Temporal Logic, pp. 241–350.

[26] H. Van Den Berg, H. Bosma, G. Dijk, H. Van Drunen, J. Van Gijsen, F. Langeveld, J. Luijpers, T. Nguyen, R. Oosting, Gerand Slagter, and et al., "ArchiMate made practical," *Work*, 2007.

[27] W. M. van der Aalst, J. Desel, and E. Kindler, "On the semantics of EPCs: A vicious circle," in *Proceedings of the EPK*, 2002, pp. 71–80.

[28] 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.