Analysis of the consistency of enterprise architecture models using formal verification methods[1]

E.A. Babkin
Professor, Department of Information Systems and Technology
National Research University Higher School of Economics
Address: 25/12, Bolshaya Pecherskaya Street, Nizhny Novgorod, 603155, Russian Federation
E-mail: eababkin@hse.ru

N.O. Ponomarev
Student, Business Informatics MSc Program
National Research University Higher School of Economics;
Software Engineer, Intel Corporation
Address: 25/12, Bolshaya Pecherskaya Street, Nizhny Novgorod, 603155, Russian Federation
E-mail: nik4nikita@gmail.com

Abstract

Enterprise architecture design is a complex process which makes it possible to synchronize the capabilities and needs of business and information technologies (IT). It can be achieved by clarifying the understanding and formalization of the business processes and the interaction of the elements of the system through their formal description. The large number of interacting business processes and enterprise architecture entities raises the question of verifying their correctness. Therefore, it is necessary to formalize the requirements for architecture and be able to automatically verify them.

In this paper, we propose a method for detecting logical contradictions in enterprise architecture models based on a model checking approach adopted in the context of business modeling. As an enterprise architecture description language, we use the modern open and independent ArchiMate standard. Developed by The Open Group, the standard provides a general specification for business processes, organizational structures, information flows, IT-systems and the technical infrastructure description of the enterprise. As a verifier, the language and tools of the MIT Alloy Analyzer system were chosen; they facilitate analysis of model constraints in terms of relational logic by automatically generating structures that satisfy the requirements of a logical model.

In this paper, we propose to simplify and automate the process of specification and verification of enterprise architecture domain models using Archi - the visual editor for ArchiMate models. We have developed the editor plug-in which translates the enterprise architecture models into the language of the MIT Alloy Analyzer system and uses the meta-model of the ArchiMate specification as the basis for constructing specific domain models. The proposed method and software solutions have been tested using the ArciSurance case and their enterprise architecture model.

[1] The research was carried out with financial support of Russian Fund of Basic Research No. 16-06-00184 A "Development and investigation models of online-discussion based on materials of political news"

Key words: enterprise architecture, ArchiMate, Alloy Analyzer, verification, consistency analysis, formal methods.

Citation: Babkin E.A., Ponomarev N.O. (2017) Analysis of the consistency of enterprise architecture models using formal verification methods. *Business Informatics,* no. 3 (41), pp. 30–40.
DOI: 10.17323/1998-0663.2017.3.30.40.

Introduction

Gartner defines the term "enterprise architecture" as a discipline for proactive and comprehensive response to a destructive force by identifying, analyzing, and making changes in the desired direction of vision and business results [1]. Enterprise architecture helps managers to find the best strategies for organizational development with respect to the information systems on which it strongly depends nowadays. However, in the process of transformation and gradual complication of the architecture, a serious problem is the lack of any practical opportunity to carry out "manual testing" of the model according to previously formulated requirements. Such testing is known as verification [2, 3].

In this paper, a general approach of formal verification is developed using the principles of model checking applied to the enterprise architecture. In this case, the required properties of the model are expressed by formulas of a certain dialect of formal logic, and the consistency checks are reduced to an exhaustive analysis of the entire space of its states [4]. In comparison with other approaches, this method has two significant advantages. It can be fully automated, and its use does not require that the business analyst should possess special knowledge in the field of mathematical logic and the theory of proofs of theorems. The paper proposes a new method for detecting logical contradictions in enterprise architecture models based on formal requirements. The formal verification of the architecture, as well as its business processes, should provide an opportunity to build a more reliable architecture.

The object of the study is the well-known case of the Open Group – the architecture of the insurance company ArchiSurance [5]. The subject of the study is the logical consistency of the main elements of the architecture, business processes and services of this enterprise. As the language of the enterprise architecture description, a modern, open and free language – ArchiMate is used in the work [6]. This standard provides a general specification for the description, construction and operation of business processes, organizational structures, information flows, IT systems and the technical infrastructure of the enterprise.

As a verification tool, the language and logic of the Alloy Analyzer system (*http://alloy.mit.edu/alloy/*) [7] is selected. The tool allows you to analyze the limitations of the model in terms of relational logic by automatically generating structures that meet the requirements of the logical model.

The authors believe that the application of formal methods has the greatest effect in the case of close integration with the software environment for modeling the enterprise architecture. The developed integrated software solution in this case allows you to automate the formalization process of the architecture model and its verification. Following this principle, within the framework of this research, it is necessary to develop a meta-model of the ArchiMate specification on the basis of which a user model will be built in a certain modeling environment.

In this article, the results are described as follows. Section 1 outlines the main problems of building an enterprise architecture and the specifics of the ArchiMate language. Section 2 shows the principle of model verification. Section 3 is devoted to the analysis of available verification tools, here the details of the

language used and the logic of the MIT Alloy Analyzer system are described in more detail. Section 4 presents the main results of the creation of the ArhiMate meta-model, and Section 5 describes the proposed general algorithm for converting the model from ArchiMate to the MIT Alloy Analyzer language construct. Section 6 describes the process of creating a formal domain model. Section 7 contains the results of the verification of the enterprise architecture view example based on the proposed algorithm. Finally, section 8 focuses on the details of integrating the presented approach into the Archi modeling tool. The Conclusion sums up the results and determines paths for further research.

1. Enterprise architecture and the problems of its construction

In practice, the enterprise architecture is usually developed because some stakeholders have certain doubts about the functioning of business and IT systems. The role of the architect of the enterprise is to eliminate these problems by defining and specifying the requirements of the stakeholders, developing architectural presentations that show how the problems will be solved taking into account the requirements and trade-offs that need to be coordinated with the potentially conflicting interests of the parties and, in the end, synchronize the business opportunities and needs and IT [6, 8]. The solution to this problem is achieved by clarifying the understanding and formalization of the description of business processes and the interaction of the elements of the system through their formal description.

As a language for describing the enterprise architecture, this work uses the modern, open and independent language of ArchiMate 2.1. Its applicability and popularity is confirmed by a large number of certified organizations in the world (4,314) (*http://www.togaf.info/archimate-visualmap.html*), and the number of participants in the annual ArchiMate Forum reaches 121 (*http://reports.opengroup.org/membership_report_archimate_forum.pdf*). There are such famous organizations among them as Boeing, Dell, IBM, Philips and many others. This language allows a business analyst to represent the architecture in the form of a set of views that, depending on the needs, can only include elements on one level or can show vertical relationships between levels.

The levels include:

✦ a business level that offers products and services to external customers;

✦ the level of the application that supports the business layer with application services implemented by software applications;

✦ a technological level that provides the infrastructure services necessary to support software systems.

Aspects:

✧ the aspect of the active structure is various components that reflect actual behavior, i.e. “subjects” of activity;

✧ the aspect of behavior represents processes, functions, events performed by subjects;

✧ The aspect of a passive structure represents objects (physical or informational) on which behavior is performed.

2. Construction and verification of models

The presence of a large number of interacting business processes and enterprise architecture entities in the ArchiMate models raises the task of verifying their correctness. This task can be performed using the model checking method. This is a method of verifying that a given logical formula is satisfied on a given formal model of the system, that is, it takes a true value [9].

The method includes several stages: modeling, specification and verification. The first task is to bring the projected architecture model to a formal form that is acceptable for the tools of verification of program models. At the stage of the specification, it is necessary to formulate the properties that the designed model of the

enterprise architecture must have in the language of formal logic.

Verification of the model can show whether the projected system corresponds to a given formal specification, but to determine whether the given specification covers all the properties of the system is not possible. The verification phase, ideally, should be done automatically, but in practice, human intervention is most often needed. In the case of a negative verification result, a counter-example will be generated that will allow the user to track where the error occurred and fix it. It is also possible that the formal form of the system or its requirements have been described incorrectly. The result of verification should also help to identify these problems [10].

3. Comparative analysis of verification tools

In the course of the work, the most popular languages of modeling and analysis of abstractions were considered: B [11], OCL [12], VDM [13], Z [14] and Alloy [7]. All of them are able to describe any complex structure in a concise and abstract form, and each has an active community of users and researchers.

Z. Language Z was first developed in 1977 by Jean-Raymond Abrial at Oxford University and is based on logic and set theory. One of the advantages of Z is that it has a rich mathematical notation, making it an expressive language. A clear style of computational notation makes it possible to maintain many different idioms. However, the automatic proof of the theorems in Z is bounded. It is automated only to a certain extent, and complex proofs often require guidance from an experienced user.

OCL. Object constraint language (OCL) is the UML constraint language developed at IBM and ObjecTime Limited and added to the UML in 1997, which was originally developed as the annotation language for UML class diagrams. OCL is based on the logic of predicates of the first order, but uses a syntax similar to programming languages, and is very closely related to the UML syntax. OCL allows you to mix declarative elements and elements of operations. However, this language is too implementation-oriented and therefore not suitable for conceptual modeling.

VDM. VDM (Vienna Development Method) is a set of methods for the development of computer systems. It originated from IBM's lab in the mid-1970s and was developed by Cliff Jones and Deans Björner. However, all existing tools do not provide a fully automatic analysis in the style of the model check. VDM supports an object-oriented paradigm and parallelism. Although this is one of the first formal methods of developing IT systems, it has been improved, standardized and is still widely used in the industry.

B. Language B was developed by Jean-Raymond Abrial, one of the creators of Z. It includes the language and method of obtaining implementations from abstract models by stage-by-stage processing. The specification language, Abstract Machine Notation (AMN), reflects its essence in the title: the system for the language is considered (as in VDM and Z) as a finite state machine with operations over the global state. Starting with a very abstract automaton, the details are added one layer at a time until an automaton is obtained that can be translated directly into the code. However, compared to the same Z, B is a lower-level language, similar to an abstract programming language, and is more focused on refinement of code, rather than on system specifications.

Alloy. MIT Alloy Analyzer, developed at the Massachusetts Institute of Technology under the leadership of Daniel Jackson, allows us to identify and detect contradictions in the projected models of systems.

Alloy is a language of structural modeling based on first-order logic for expressing complex structural constraints and behavior. The Alloy language derives from the Z specification language and Tarski's relational calculi, treating relationships as the main unit of analysis, and

uses the relational composition as a powerful operator for combining various structured data.

MIT Alloy Analyzer is a tool for analyzing models written on Alloy. It supports two types of automatic analysis: searching for an entity that satisfies the constraint, and searching for a counter-example for the given model judgments.

To limit the size of the search, the scope function is used; this fixes the number of entities and counter-examples analyzed by this command. Alloy analyzes the limitations of the model and selects the structures that satisfy them. Structures are displayed graphically, and their appearance can be adjusted manually.

During the assessment, the language Alloy was chosen. It is a full-fledged structured declarative modeling language that can express all sorts of complex structural constraints, reflect the logic of model behavior and conduct automatic analysis.

4. Meta-modeling

To find the contradictions in the enterprise architecture models using the MIT Alloy Analyzer tool, we must create a meta-model of the ArchiMate specification, on the basis of which specific domain models will be constructed later. A meta-model is a kind of language model that captures its basic properties, language concepts and semantics [4].

The core of the developed meta-model contains several key entities of the ArchiMate 2.1 specification common to all levels of representations. From them, all other elements on each level of the language are inherited and expanded by their properties. The keyword "abstract" of the Alloy language emphasizes the signature property that this entity does not include elements outside its extensions. In the proposed meta-mode architecture, all the signatures of the formal model of the system extend the essence of Element.

Graphically, the corresponding fragment of the upper level meta-model ArchiMate is shown in *Figure 1*.

Having defined all the necessary basic entities, we expand the hierarchy of the meta-model towards the business level, which reflects all the entities and links from the ArchiMate 2.1 specification[2].

Further, by the same principle, we set the meta-model of the application level in the Alloy language from the ArchiMate 2.1 specification[3]. In this case, the functionality of some types is similar, for example, ApplicationFunction and ApplicationInteraction. In order not to duplicate the links and code of the model, we select a base class with all common types of communication and add its functionality for each type.

The last level of the meta-model is the technology level. It is described by analogy with previous levels in accordance with the specification ArchiMate 2.1 in the language Alloy[4]. Thus, the meta-model architecture has a hierarchical

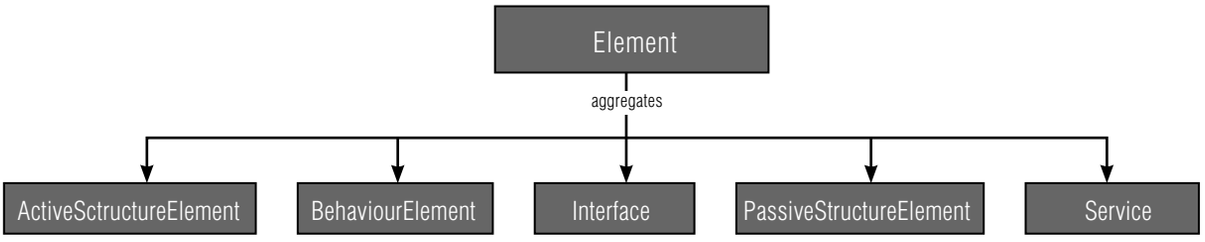

Fig. 1. Graphical representation of the top-level meta-model AcrhiMate

[2] Graphical representation of the meta-model is available at: *https://goo.gl/ZLr4Qv*
[3] Graphical representation of the meta-model is available at: *https://goo.gl/J9TTHX*
[4] Graphical representation of the meta-model is available at: *https://goo.gl/Yo44Dw*

structure: there is an upper common layer from which three ArchiMate level modules are inherited. These, in turn, will be used by the entities of the user case.

5. Model transformation algorithm

Modeling of the entities of the subject domain is completely based on the entities of the created meta-model [4, 10]. Therefore, before describing the formal architecture model, you need to import the module of the developed ArchiMate specification meta-model with the Alloy command "*open ArchimateMetaModel*".

The algorithm for converting from ArchiMate to Alloy is proposed as follows. Each entity of the view model is transformed into a signature with an extension corresponding to the type of the entity. In the signature name, spaces are replaced with a lower slash, and uppercase letters are replaced with the corresponding lowercase letters. Thus, a unique name is created for the signature. Connections that come from the essence are transformed into limitations (facts) of the signature. If the entity does not have any relationships that are described in the signature of the meta-model from which it is inherited, then the constraint for this relationship is set to "*none*", that is, it forbids creating a counterexample with this connection when verifying (*Table 1*).

Table 1.

Displaying entities ArchiMate in Alloy

ArchiMate	Alloy
Model elements	The signature (sig "name") with an extension (extends) the entity type (ex. BusinessFunction)
Communication	The list of restrictions (fact) for a particular signature.

6. Creating a formal model of the subject field

As a verification case, consider a classic example designed to illustrate the use of the ArchiMate modeling language in the context of the TOGAF structure. It describes the company's basic architecture, as well as a number of change scenarios [5].

The case concerns the insurance company ArchiSurance, which was formed as a result of the merger of three previously independent companies:

- ✦ Home & Away (homeowners and travel insurance);
- ✦ PRO-FIT (auto insurance);
- ✦ Legally Yours (insurance of legal expenses).

At present, the company consists of three divisions with the same names and headquarters as their independent predecessors.

ArchiSurance was created to use the synergy effect between the three organizations to control their costs, maintain customer satisfaction and invest in new technologies. The new company offers all insurance products of the three companies that were merged, and its organizational structure looks like this (*Figure 2*).

To completely cover all aspects of the enterprise architecture on one diagram – the task is not simple and voluminous. More often, the user is only interested in a particular aspect of architecture. Therefore, in ArchiMate there is a concept of "representation". This is a kind of point of view that allows you to work flexibly in a common architecture, focusing on important aspects, both individual and in a bunch of different levels.

In this paper, we will consider the multilayered and most general representation of the ArchiSurance architecture, divided into three levels according to the ArchiMate specification[5].

[5] Graphical representation of the meta-model is available at: *https://goo.gl/XdvNL*

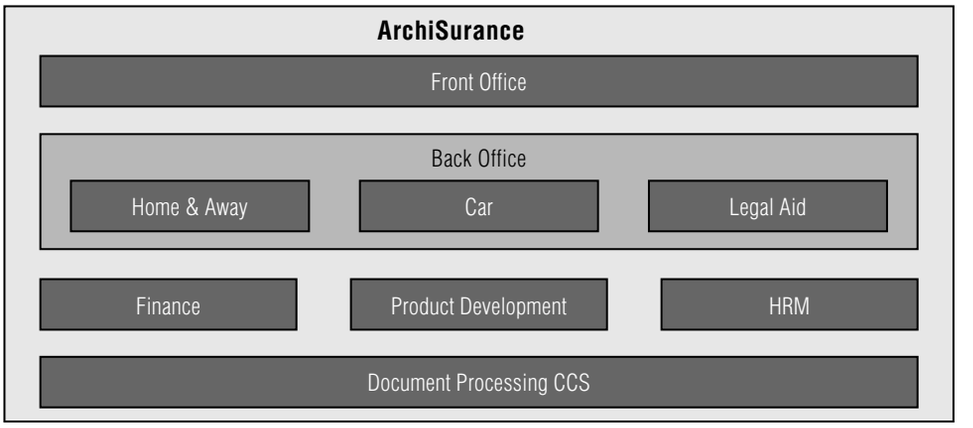

Fig. 2. The organizational structure of the company ArciSurance

In accordance with the transformation algorithm described earlier, we obtain a model in the language of Alloy for subsequent formal analysis. The model consists of 67 signatures and, accordingly, of the same number of facts. Every fact has from zero to six constraints on connections.

7. Determination of the formal requirements of domain and verification in the MIT Alloy Analyzer

At the architecture design stage, the properties that the future enterprise architecture model should have are formulated. One of these properties is the following statement: "Are there any application components or their functionality that use data, but do not have access to the technology services for their extraction?"

To conduct verification, it is necessary to describe this statement in the language of Alloy's formal logic, and to determine the scope of the search. It is necessary that the search for a solution be carried out with all the presentation entities, so the query is supplemented with a list of all the elements of the model with the keyword "exactly".

When starting the search for a counter-example for a given judgment of the model, a significant role is played by scaffolding. Often quantified formulas can be reduced to equivalent formulas without the use of quantifiers. This abbreviation is called scolemization and is based on the introduction of one or more Skolem constants or functions that fix the limitation of the quantitative formula by their values.

In this example, the MIT Alloy Analyzer finds a counter example for a given restriction and assigns the *app_comp* Skolem relation of the ApplicationComponent type to the name "$DataAccess_app_comp" for the existing "Policy Data Management" entity, and the *app_funct*of the ApplicationFunction type is named "$DataAccess_app_funct" for the existing "Create Policy" entity. Thus, in our example, two entities are found that do not meet the specified formal requirements (*Figures 3 and 4*).

After a negative verification result, there are two possible solutions to the problem: changing the current architecture or formal requirements. If the formal requirements are determined by the stakeholders and, therefore, are not subject to adjustment, then the architecture of the enterprise needs to be reconstructed until verification yields a positive result.

8. Developed integration approach for Archi

One of the main goals of this work is the development of a software tool that would automate part of the verification process of

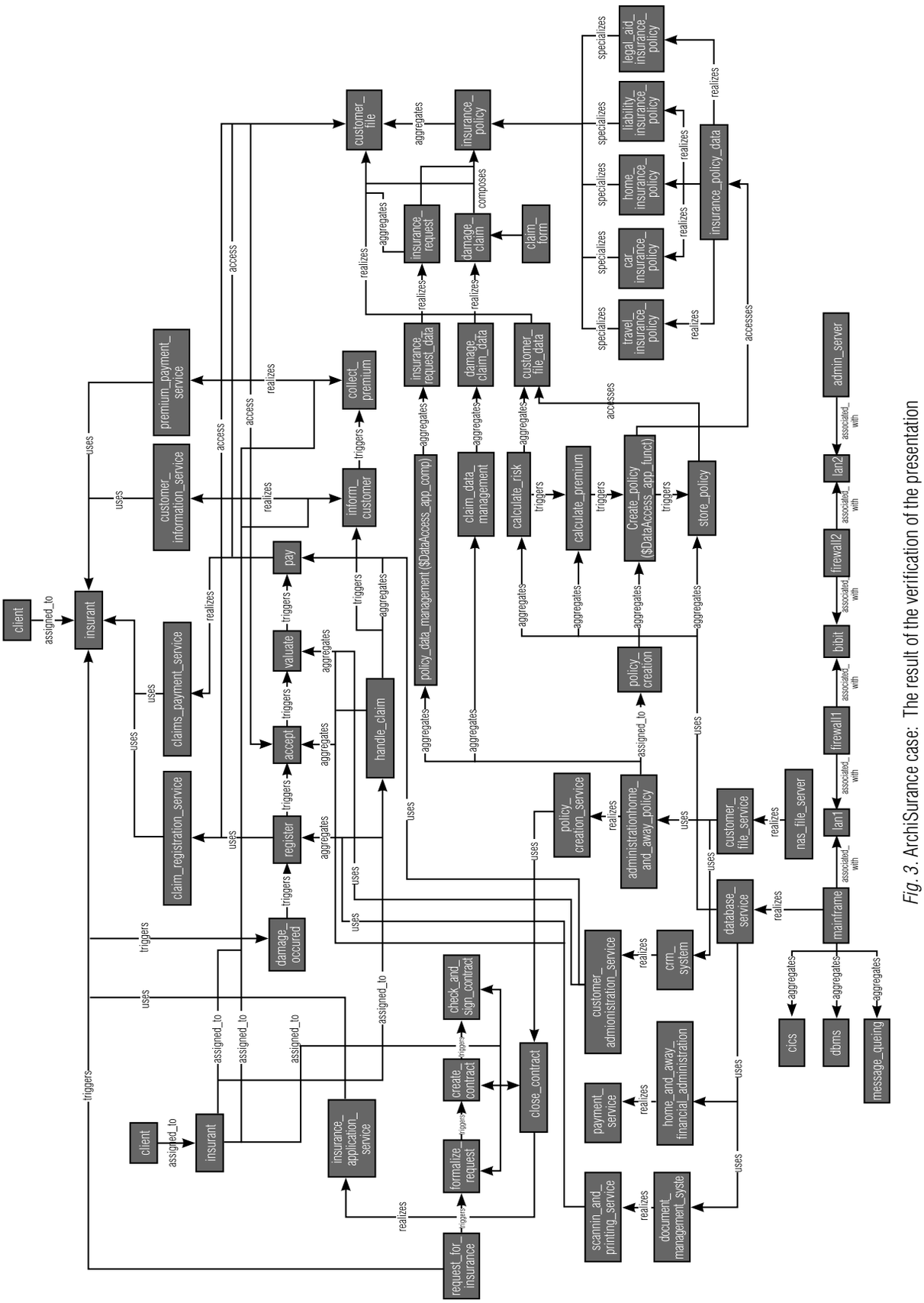

Fig. 3. ArchiSurance case: The result of the verification of the presentation

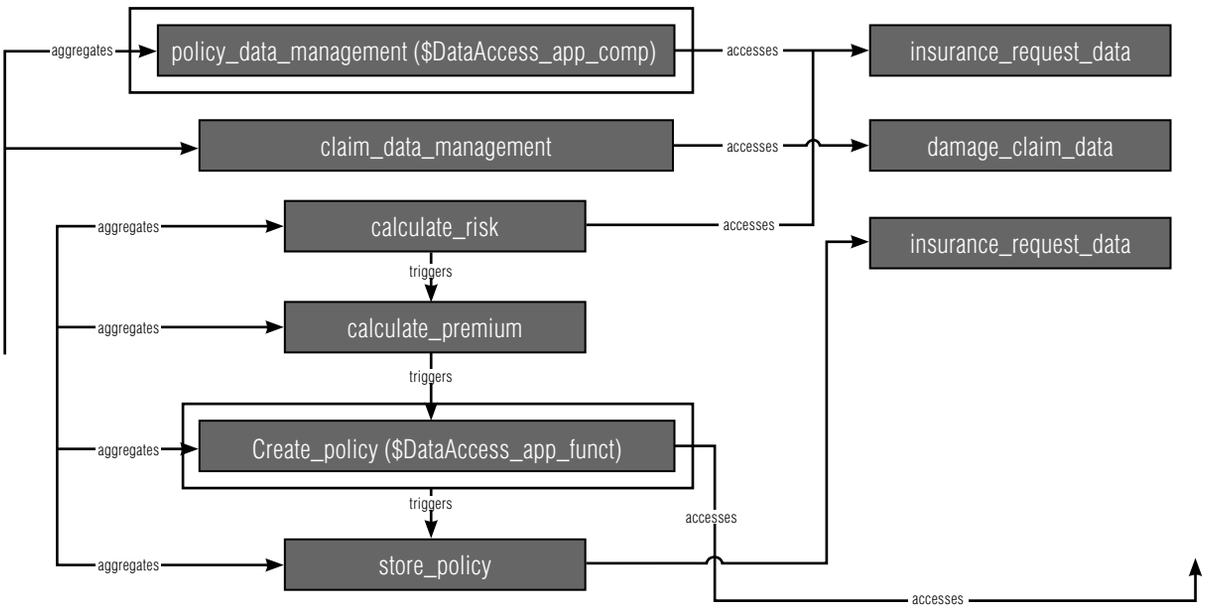

Fig. 4. ArchiSurance case: Found entities that contradict the given condition (in the box)

ArchiMate models and make the approach of testing models in the field of business modeling more accessible. As a software platform, the popular Archi editor has been selected. This is a free cross-platform open source tool for visual modeling and design of ArchiMate models developed on the Eclipse EMF platform and extensible with custom plug-ins.

In this paper, a plug-in was developed for the visual editor Archi which supports the automatic transformation of the enterprise architecture view into the Alloy language (*Figure 5*). The plugin is written in Java. The design of the solution contains three main classes: the logic of parsing and model transformation, the implementation of the user interface and auxiliary methods for converting strings. The transformation logic is encapsulated in the Alloy Exporter class, and the public transformModel method takes an object that implements IArchimateModel interface and creates a temporary CaseExample.als file into which the converted model is written. The input interface for Alloy commands is written using the Swing library. The amount of code due to the re-use of Alloy libraries is 546 lines (*https://github.com/nik-ponomarev/archimate2alloy*).

After designing a multilevel architecture of the ArchiSurance enterprise, the dialog of the verification window in Archi is opened by selecting "File → Export → Model to Alloy Format" from the context menu.

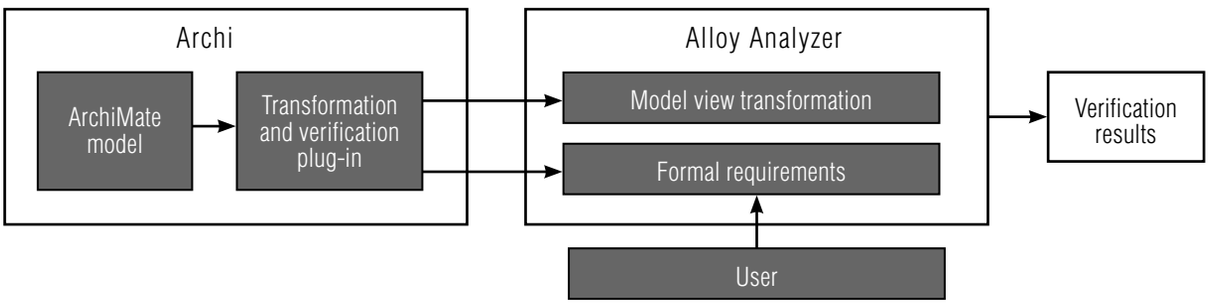

Fig. 5. Scheme of verification toolkit

Next, there appears an interface for entering formal requirements for the architecture in the language Alloy, as well as commands for launching verification, where you need to determine the coverage of the search. At this stage of implementation, it is possible to check only one statement per launch.

The "*Use full scope*" option means that all the entities of the model must participate in the verification, each in a single instance. If the user needs to set his own type of coverage, then there is the option "*Custom scope*". The "*Find solution*" button starts the verification mechanism.

In the case of a negative verification result, in the presence of skolemized structures, entities that do not satisfy the query condition will be painted red. In the absence of these variables, all elements of the model will be painted in red, which will be generated by Alloy Analyzer as a counter-example. If no counter-examples are found, then the original model satisfies formal requirements. In this case, the corresponding window with the message "*No example/counter-example found*" will be displayed.

In our example, the plugin found two application-level components that did not satisfy the constraint condition, and colored them in red. These are Policy Data Management and Create Policy components that use client data in the current architecture, but do not have access to their receipt from database services of the technological level. In this regard, it is necessary to revise the logic of the current architecture.

The proposed approach to model testing allows to identify errors that are not just related to the incorrect display of the specification, namely errors at the level of the architecture logic of the enterprise architecture and the behavior of business processes. It should be noted that it is not possible to carry out such verification manually with the increase in the size of models, which again confirms the practical value of this tool.

Conclusion

In this paper, the problem of automatic verification of enterprise architecture models in ArhiMate language is investigated. As a tool for the model checking method, the relational logic tool and the MIT Alloy Analyzer modeling system were used.

The idea of the work is to develop a solution that is tightly integrated with the actively used in practice Archi modeling tool to extract all the elements from the architecture view, and then completely automatically create a formal model in the Alloy language and conduct verification through MIT Alloy Analyzer. Verification is based on the projected meta-mode of the ArchiMate specification in the Alloy language, which describes the main entities, the relationships between them, and other language restrictions. Specification of formal requirements in the form of facts, assertions, predicate and functions can be entered by the user in a separate menu, specifying options for restricting the search. The work also discusses the methods for transforming the model and the requirements description rules used in the implemented software.

Finally, the applied nature of the method is presented in the ArchiSurance case – verification of its generalized multi-layered representation of the enterprise architecture. In addition, the work describes the mechanism for displaying verification results in the Archi modeling tool, which allows you to better display the principles of the enterprise's functioning.

Based on the work done, we can conclude that the mechanism of logical validation for the architecture models of the company ArchiMate is applicable. This method is most important for models with a large number of elements and connections between them, given that manual verification is not possible. The introduction of a formal description of the enterprise architecture, its business processes, and requirements

will allow us to build quality management systems of the company, to solve the problem of building an effective management structure, to optimize processes based on key indicators.

In the future, this work will be improved in the direction of implementation of verification on models with the help of LTL/CTL logics. Further development of the suggested approaches will allow us to use formal analysis methods for a broad class of the models, including the models of communication of autonomous agents in the framework of DEMO methodology. ■

References

1. James G.A., Handler R.A., Lapkin A., Gall N. (2005) *Gartner enterprise architecture framework: Evolution 2005*. October 25, 2005. Gartner ID: G00130855.
2. Clarke E., Grumberg O., Peled D. (1999) *Model checking*. MIT Press.
3. Korotkov A. (2013) *Arkhitektura predpriyatiya. Kak zastavit' IT rabotat' na vashu kompaniyu* [Enterprise architecture. How to force IT to work in favor of your company]. Available at: http://andrey-korotkov.ru/wp-content/uploads/2013/02/andrey-korotkov.ru_Enterprise_architecture.pdf (accessed 16 March 2017) (in Russian).
4. Clark T., Sammut P., Willans J. (2008) *Applied meta-modeling: A foundation for language driven development*. Ceteva.
5. Jonkers H., Band I., Quartel D. (2012) *The ArchiSurance case study. White paper*. The Open Group.
6. The Open Group (2013) *Archimate 2.1 Specificattion. Open Group Standard*. Zaltbommel: Van Haren Publishing. Available at: https://www.vanharen.net/Samplefiles/9789401800037SMPL.pdf (accessed 16 March 2017).
7. Jackson D. (2006) *Software abstractions: Logic, language and analysis*. MIT Press.
8. Kudryavtsev D.V., Arzumanyan M.Y., Grigoriev L.Y. (2014) *Tekhnologii biznes-inzhiniringa* [Technologies of business engineering]. St. Petersburg, Polytechnic University (in Russian).
9. Karpov Y.G. (2009) *Model checking. Verifikatsiya parallel'nykh i raspredelennykh programmnykh sistem* [Model checking. Verification of parallel and distributed program systems]. St. Petersburg, BHV-Petersburg (in Russian).
10. Szwed P. (2015) Verification of ArchiMate behavioral elements by model checking. Proceedings of the *14th IFIP International Conference on Computer Information Systems and Industrial Management (CISIM 2015)*. Warsaw, Poland, 24-26 September 2015, pp. 132–144.
11. Lano K. (2012) *The B language and method: A guide to practical formal development*. Springer Science & Business Media.
12. Warmer J.B., Kleppe A.G. (1998) *The object constraint language: Precise modeling with UML*. Addison-Wesley.
13. Fitzgerald J.S., Larsen P.G., Verhoef M. (2008) Vienna development method. *Wiley encyclopedia of computer science and engineering*. John Wiley & Sons.
14. Spivey J.M., Abrial J.R. (1992) *The Z notation*. Hemel Hempstead: Prentice Hall.