© A. G. IVASHKO, A. V. GRIGORYEV, A. A. KROPOTIN, E. O. OVSYANNIKOVA

ivashco@mail.ru, 107th@mail.ru, aakropotin@icloud.com, ek.ovsyannikova@gmail.com

UDC 510.5; 510.22; 519-6

USE OF TABLEAU ALGORITHM FOR BUSINESS PROCESS VERIFICATION

ABSTRACT. The aim of this work is to develop the verification method of a business-process model applying the description logic instrument. The task of verification is to determine the presence of inconsistencies in the models, which can lead to unreachable actions. The article considers five main types of inconsistencies that arise from a business process modeling due to incorrect business-process representation and modification of a formalization method of the business process diagrams in BPMN and UML Activity notations as most widely-used. Limitations of existing approaches for verification are given, in particular the instrument of Petri Nets. The new method of verification with the help of only description logic formalism is suggested. The special software forming the knowledge base in OWL was developed for testing the above method. The tableaux algorithm was used as the instrument for reasoning and model inconsistencies detection. Examples of test models of different classes depending on their dimension and results of method testing are presented.

KEY WORDS. Simulation modeling, description logic, verification, business-process, BPMN, UMLActivity.

Introduction

Verification of business process models helps to determine at hardware stages various syntax errors or irrelevances of model behavior to its business process. The successful verification of business process model makes it possible to proceed with the model analysis of the process.

The relevance of this work is determined by the fact that building business processes models is an integral task of information systems design and business reengineering [6]. The paper discusses the most popular notations and methodologies of business process modeling, such as BPMN [7], and UMLActivity [8].

The approach of business processes verification, proposed in this paper, helps to determine design flaws, unrelated to non-compliance with the syntax rules of design notation, and errors caused by incorrect display of business process in the construction of the model. In the first case it is sufficient enough to check the validity of design notation syntax rules, but in the second case it may not be enough, because the detection of such errors requires the simulation of business process to compare the behavior of the model and the process itself. It should be noted that the identification of the abovementioned errors without the application of information technologies is hampered by the growth of business process model size (number of elements), because notation and methodology, which this work is devoted to, contain more than a dozen possible syntactic constructions (having greater expressive power) that occur in real object-oriented models more than one hundred times. Thus, we can conclude that the simulation of models aimed at detecting irregular projection process by means of mental simulation (mental experiment) is practically impossible due to limited human abilities (short-term memory, attention, etc.).

Over the last years, different methods for the formalization of business process models have been worked out with the aim to apply logical apparatus to the verification of these models [14–27], which attests to the absence of a perfect and irrefutable method of business process model presentation in any formal system and algorithms for this model verification. All these facts make the studied subject of great current interest.

It may be added that at the moment there is a method of identifying syntactic design errors using description logic [23–27] and identifying errors related to the mismatch of the behavioral aspects of the modeled business process of its projection with the application of Petri nets algorithms. The idea of the method is to create a test chart for errors of any kind with the use of only formal description logics without Petri nets algorithms, thus dropping the need to construct models in several formal systems, taking into account the characteristics of each one.

The aim of this paper is to develop an alternative approach to the verification of business processes using the apparatus of description logics, which will promote the development of a flexible tool for models verification.

Statement of the problem and solutions review

The task of business processes verification is to identify the errors in the models, which may lead to inaccessibility of certain actions. W. Sadiq and M. Orlowska distinguish 5 types of possible errors in data streams models [5]:

1. incorrectusage—synchronization of object with only one input stream;
2. deadlock—synchronization of two mutually exclusive flows;
3. livelock—circling, without the possibility of interruption;
4. unintentional multiple execution—merger of two parallel streams;
5. active termination—parallel streams lead to more than one final task.

By now, the most widely used approach for the verification of the models is the tool of Petri nets [6, 3]. This approach is applicable in cases where the elements of the constructed model may be clearly displayed in Petri network elements, wherein the algorithm simulates the execution of tasks by moving the markers in the network, corresponding to the business process. Petri nets approach is used to verify the models described with the most popular notations: EPC [3], BPMN [7], and UMLActivity [8]. However, algorithms using Petri nets are characterized by several drawbacks, among which it is worth noting first of all that they belong to the NP class (having non-polynomial computational complexity), which can significantly reduce the time of algorithm execution at the increase of elements quantity in the models. Another disadvantage is inflexibility of data algorithms, which means that identification of a new error type will lead to the improvement (or total processing) of implemented algorithm.

Solution

BPMN notation is used as a notation for describing business process models, since it provides ample opportunities for the description of any possible cases in business processes [13]. Moreover, it is one of the most popular notations used in the development of information systems [29]. In order to identify the errors in the models the authors apply formal description logics [9] and tabular algorithm [10], which is used for logical induction and search of errors in the models.

Slight example, displayed in Fig. 1, shows the specifics of the design errors and ambiguity of their identification.

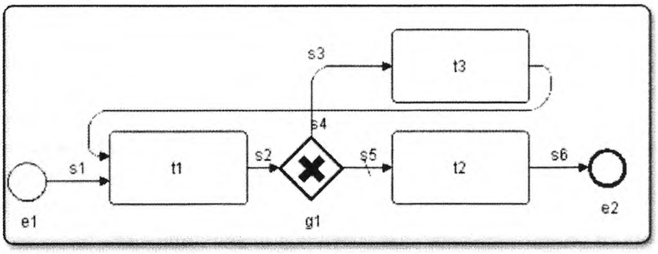

Fig. 1. Non-loopable process in BPMN notation

Figure 1 shows a diagram of a business process, which is a loop passing through the elements t1, g1 and t3. However, due to the fact that g1 is an exclusive gateway, the process can be completed during the passage through the elements e1, t1, g1, t2, and e2. Therefore, the process is not considered loopable if in any of its interpretations it reaches its end.

To use description logics [30] and table algorithm we formalize the elements of BPMN notation, using a modified approach of R. Dijkman, M. Dumas and C. Ouyang [7], but confine ourselves to the task entries, to incorporating and exclusive gateways, starting and terminal events, and to the control flows, considering the fact that most models can be constructed without other elements. We denote T as the task set, G—the set of all gateways, G_E—sets of exclusive and incorporating gateways (G_I), so that $G_E \cap G_I \equiv \emptyset$, $G_E \cup G_I \equiv G$, s being an initial event and e—the final event. The set of all chart objects we denote as $O \equiv G \cup T \cup \{s\} \cup \{e\}$. By ratio of transfer control R we denote the set of pairs $(o_1, o_2) \in (O$ ⍰ $O)$, featuring all the arrows in the BPMN diagram. To carry out the tasks of logical induction, we use SROIQ logic, which allows the use of standard operations logic ALC [9], broadened by individuals and reverse roles.

The test of models will result in the construction of ontology, the compatibility of which will determine the presence of errors in the model. To construct the ontology we define the function f, displaying the elements in the set O into the elements ABox

A ontology, described in description logic language, as well as the ratio of L, establishing connections between the elements A:

each element $t_i \in T$ will be displayed in individual $y_i \in A, f(t_i) = y_i$;

each element $g_i \in G_E$ will be displayed in individual $h_i \in A, f(g_i) = h_i$;

each element $p_i \in G_I$ will be displayed in individual $k_i \in A, f(p_i) = k_i$;

element $\{s\}$ will be displayed in an individual $d \in A$, $f(s) = d$;

element $\{e\}$ will be displayed in individual $r \in A$, $f(e) = r$;

$L\ (f(s), f(o_j)) = \{CF\}$, for any $o_j \in O$ such that $(s, o_j) \in R$;

$L\ f(o_j), f\ (e)) = \{CF\}$, for any $o_j \in O$ such that $(o_j, e) \in R$;

$L\ (f(t_i), f(o_j)) = \{CF\}$, for any $o_j \in O$ such that $(t_i, o_j) \in R$;

$L\ (f(g_i), f(o_j)) = \{CF\}$, for any $o_j \in O$, such that $(g_i, o_j) \in R$;

$L\ (f(o_j), f(o_j)) = \{\neg CF\}$, for any $o_j \in O$;

For the elements o_i we define axiom $f(o_i) \sqsubseteq (\forall RC.\{\neg(f(o_l))\})$, for $o_l \in O$ with $(g_i, o_l) \notin R$;

For the elements g_i we define C_{gi} concept, interpretation of which they belong to , while $C_{gi} \equiv (\sqcup(\exists CF.\{f(o_j)\} \sqcap \forall CF.\neg C_j))$ for any $o_j \in O$ with $(g_i, o_j) \in R$, $o_k \in O$ with $(o_k, g_i) \in R$, concept Cj is only of those individuals $f(o_h)$, for which $(g_i, o_h) \in R$ and $o_h \neq o_j$; the axiom $\{g_i\} \sqsubseteq C_{gi}$ will be added to a set of axioms TBox *T*;

For each of the g_i elements we add to T axiom $\{g_i\} \sqcap \exists RC.\{t_j\} \sqcap \exists RC.\{t_k\} \sqsubseteq \perp$, for all different t_j and t_k, with $(t_i, g_i) \in R$ and $(t_j, g_i) \in R$ or $(g_i, t_i) \in R$ and $(g_i, t_j) \in R$;

For each of the p_i elements, we define the axiom $\{f(p_i)\} \sqcap \exists CF.\{f(o_j)\} \sqsubseteq (\sqcap \exists RF.\{f(o_k)\})$, for any $o_j \in O$ with $(p_i, o_j) \in R$, $o_k \in O$ with $(o_k, g_i) \in R$;

To define the reachability of terminal process , we should add the axiom $f(s) \sqsubseteq (\sqcap \exists CF.\{f(o_j)\})$, for any $o_j \in O$;

To check livelock errors it is important to exclude step 11 of this formalization and to add the axiom of transitivity of CF function. To check the consistency of ontology we will use the standard tabular algorithm logic SROIQ, implemented in the system HermiT [11]. The verification procedure will return *false* value, if the model has no loopable actions, otherwise the procedure returns *true* value. Formalization of the algorithm is presented in Listing 1.

Listing 1. **Checking for livelock errors.**

```
There is a cycle procedure (Diagram B)
1. OWL ontology O = Convert diagram (B);
2. To remove axioms $f(o_i) \sqsubseteq (\forall RC.\{\neg(f(o_l))\})$ from ontology O;
3. Add axiom of CF transitivity role to ontology O.
4. Check the correlation of ontology O;
5. If O is correlated, then return false, otherwise return true;
```

The procedure *Convert chart* converts model in terms of OWL in accordance with the above rules.

To determine infeasible actions it is necessary to check the correlation of ontology built by these rules. If such knowledge base is nonsatisfiable, the end of the process will be unattainable. We can determine the reachability of any process by checking the availability of the element *f*(t_i) in any interpretation of the knowledge base. The procedure for determining infeasible actions is presented in Listing 2.

Listing 2. **Checking for deadlock errors.**

```
Infeasible Actions Procedure (Diagram **B**).
1. OWL ontology **O** = Convert diagram (**B**);
4 . Check the correlation of **O**;
5 . If **O** is uncorrelated, then return **true**;
6. Return **false**;
```

Fault finding procedures were divided into two different algorithms, so that the process of models verification could help to distinguish each type of error.

Despite the fact that originally the task of business process models verification was set in BPMN notation, this approach was found to be able to support the verification of models built in UMLActivity notation. In [12] we can see the templates for the translation of business process diagrams from one to another, built in BPMN and UMLActivity notations. Thus, many syntactic BPMN diagrams can be represented by syntactic constructions and UMLActivity charts, despite the fact that they are characterized by different graphical presentation, i.e. one and the same business process may be equally mapped both in BPMN notation, and UMLActivity notation. For example, the process shown in Fig. 1 as a BPMN diagram can be represented as UMLActivity chart displayed in Fig. 2.

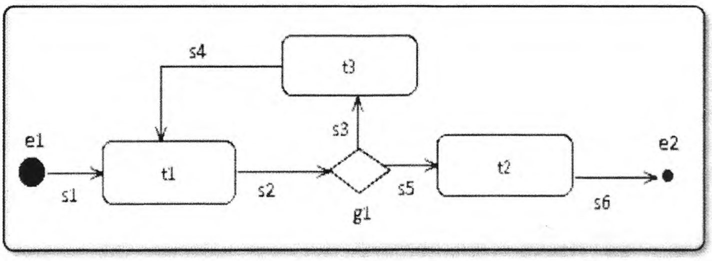

Fig. 2. Non-loopable process in UMLActivity notation

With the ability to provide consistent elements of the process in BPMN and UMLActivity notations the proposed verification approach to business process is applicable in both cases.

Results

Test of the proposed method required the development of BPMN2OWL software, which forms the knowledge base in OWL language from XML extensible markup language file describing the model that is designed in a software product Eclipse Modeling Tools, which uses the tools for constructing BPMN2-Modeler and the Papyrus [27]. To determine the fixation processes the system of logical induction HermiT [11] was used, which is provided together with the construction tool for Protégé ontologies.

Models of three classes, depending on their size, were applied (number of graphical elements in diagrams):

• Class A—from 10 to 35 elements;

• Class B—from 36 to 100 elements;

• Class C—from 101 to 300 elements.

In addition, diagrams of three options were designed for each of the model classes:

• containing no errors;

• containing livelock errors;

• containing errors associated with infeasibility of any action.

Thus, a positive assessment test of our approach is identification of all models containing errors and setting of all perfectly-built business process models with the use of tableau algorithm regardless of the applicable notation (BPMN or UMLActivity).

Due to the limited space in Figs. 3–6, the examples of test models of class A business processes are given.

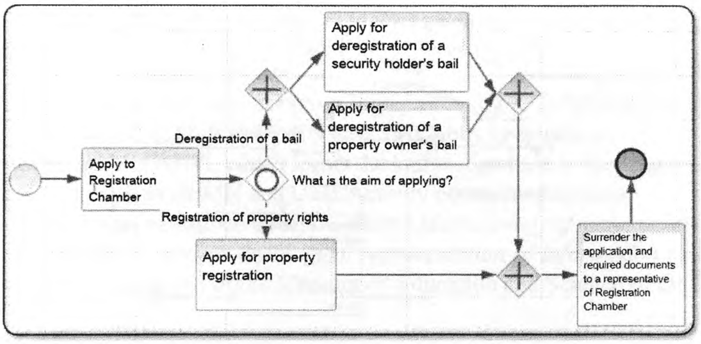
Apply for deregistration of a security holder's bail
Apply for deregistration of a property owner's bail
Deregistration of a bail
Apply to Registration Chamber
What is the aim of applying?
Registration of property rights
Apply for property registration
Surrender the application and required documents to a representative of Registration Chamber

Fig. 3. Class A test process containing no errors in BPMN notation

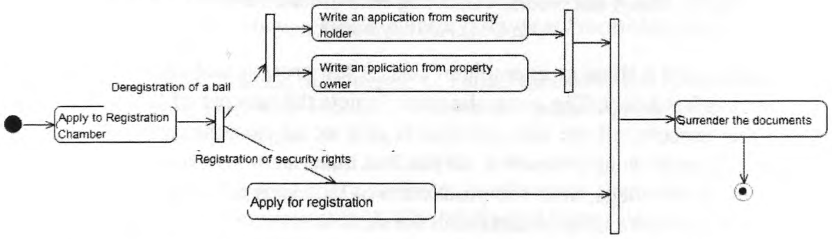
Write an application from security holder
Write an pplication from property owner
Deregistration of a bail
Apply to Registration Chamber
Surrender the documents
Registration of security rights
Apply for registration

Fig. 4. Class A test process containing no errors in UMLActivity notation

Figures 3 and 4 show an example of Class A test process, containing no design errors. The diagram displays the process of property registration of a mortgaged apartment, purchased after one reference to Registration Chamber. The main objectives of the process are applying for cancellation of registration of mortgage and applying for property rights registration. Moreover, these objectives cannot be executed at once, but they both have to be performed to achieve the goals. If you perform only one of them, then mortgage registration can be cancelled or property registration can be refused due to the fact that it is impossible to register ownership of the pledged property.

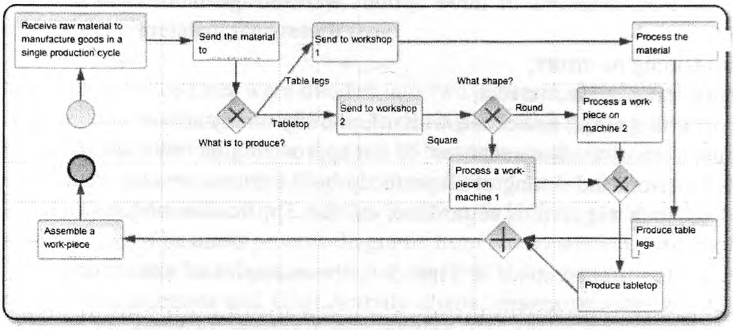

Fig. 5. Class A test process containing error of infeasibility of any action in BPMN notation

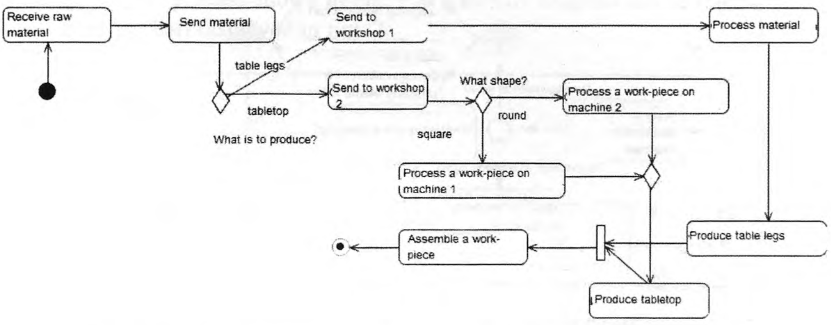

Fig. 6. Class A test process containing error of infeasibility of any action in UMLActivity notation

Figures 5 and 6 show an example of class A test process that contains an error of any action infeasibility. The given diagram depicts the process of table production. Within this process, all the raw material is sent to the manufacturing stage with a single production cycle. Moreover, all the raw material is sent to the production of either a set of table legs, or to the production of tabletops of one out of two shapes, but the production of a table requires both the manufacturing of table legs and tabletops at once. As a result, there comes the infeasibility of the action 'Assemble a work-piece' because it requires simultaneous completion of all paths in the logical operator 'and' which can never be reached.

Testing was carried out on a computer with the following configuration characteristics: CPU: 2.2 GHz, RAM: 1024 Mb with the operating system Windows

XP Professional 64bit. The results of models testing with the use of tableau algorithm are shown in the table below.

Table

Results of method testing

Model size	Containing no errors	Containing livelock errors	Containing infeasible actions
Class A	1.76	2.09	0.721
Class B	2.579	1.402	1.632
Class C	60.406	10.281	12.109

The results of system testing (Table) showed that the developed method detected the error in all models containing errors and found no errors in perfectly-built models, displayed in the form of BPMN and UMLActivity diagrams. Moreover, the results showed that error detection testing takes time, acceptable for models verification.

Conclusion

The given work analyzed syntactic constructions of BPMN and UMLActivity notations, and developed:

• method to identify OWL ontology from the business process diagram, built in BPMN and UMLActivity notations;

• axiomatic assertions of description logic, allowing to determine the errors in models of data streams, built in BPMN and UMLActivity notations;

• software BPMN2OWL which forms the knowledge base in the language OWL from XML file, built in BPMN and UMLActivity notations diagrams.

The research was performed under the project *Methodology of managed enterprise architecture design, based on ontological representation of information systems* as specified in the instructions of the Ministry of Education and Science of the Russian Federation for 2013.

REFERENCES

1. 1. Wynn, M.T., Verbeek, H.M.W., W.M.P. van der Aalst, A.H.M. ter Hofstede, Edmond, D. Business Process Verification — Finally a Reality! *Business Process Management Journal.* Vol. 15(1), Pp. 74-92.

2. Dijkman, R.M., Dumas, M., Ouyang, C. Formal Semantics and Analysis of BPMN Process Models using Petri Nets. Intelligent Information Technology Application. IITA '08. Second International Symposium. 2008. Pp. 70-74.

3. Lopez-Grao, J.P., Merseguer, J., Campos, J. From UML Activity Diagrams To Stochastic Petri Nets: Application To Software Performance Engineering*. Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium. 2008. Pp.70-74.

4. Object Management Group. Business Process Model and Notation (BPMN) Version 2.0 // http://www.omg.org/spec/BPMN/2.0/PDF (дата обращения: 18.05.2013)

5. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T. Querying business processes // In VLDB 06. 2006. Pp. 343-354.

6. Dimitrov, M., Simov, A., Stein, S., Konstantinov, M. A bpmo based semantic business process modeling environment. In Proc. of the Workshop on Semantic Business Process and Product Lifecycle Management at the ESWC. Vol. 251 of CEUR-WS, 2007.

7. Koschmider, A., Oberweis, A. Ontology based business process description // In Proceedings of the CAiSE-05 Workshops, LNCS. 2005. Pp. 321-333.

8. Markovic, I. Advanced querying and reasoning on business process models. In W. Abramowicz and D. Fensel, editors, BIS. Vol. 7 of Lecture Notes in Business Information Processing. 2008. Pp. 189-200.

9. De Nicola, A., Lezoche, M., Missikoff, M. An ontological approach to business process modeling // In Proceedings of the 3d Indian International Conference on Artificial Intelligence (IICAI). 2007. Pp 1794-1813.

10. Thomas, O., Fellmann, M. Semantic epc: Enhancing process modeling using ontology languages // In Proceedings of the Workshop on Semantic Business Process and Product-nLifecycle Management (SBPM). 2007. Pp. 64-75.

11. Weber, I., Hoffmann, J., Mendling, J. Semantic business process validation // In Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM). 2008.

12. Wong, P.Y.H., Gibbons, J. A relative timed semantics for BPMN // In Proceedings of 7th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA08). 2008.

13. Born, M., Dorr, F., Weber, I. User-friendly semantic annotation in business process modeling // In Hf-SDDM-07: Proc. of the Workshop on Human-friendly Service Description, Discovery and Matchmaking WISE-07. 2007.

14. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., P. Tonella. Reasoning on semantically annotated processes // In Proc. of the 6th Int. Conference on Service Oriented Computing (ICSOC'08), Vol. 5364 of LNCS. 2008. Pp. 132-146.

15. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P. Semanticallyaided business process modeling // In Proc. of 8th International Semantic Web Conference (ISWC 2009). 2009. Pp. 114-129.

16. Ghidini, C., Rospocher, M., Serani, L. A formalisation of BPMN in description logics // Technical Report TR 2008-06-004, FBK, 2008.

17. Bozzato, L., Ferrari, M., Trombetta, A. Building a domain ontology from glossaries: a general methodology // In Proceedings of 5th Workshop on Semantic Web Applications and Perspectives, 2008.

18. Van Dongen, B.F. and Verbeek, H.M.W. Verification of EPCs: Using Reduction Rules and Petri Nets // Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05), Vol. 3520 of Lecture Notes in Computer Science. 2005. Pp. 372-386;

19. White, S. Process Modeling Notations and Workflow Patterns // In Workflow Handbook, 2004. Pp. 265-294.

20. MDT Papyrus Group. Papyrus // URL: http://www.eclipse.org/papyrus/ (reference date: 20.05.2013).21. Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P.F. (editors).

21. The Description Logic Handbook: Theory, Implementation and Applications. CUP, 2003.

22. Schmidt-Schau, M. and Smolka, G. Attributive concept descriptions with complements // Artificial Intelligence, Vol. 48(1). 1991. Pp. 1-26.

23. Horrocks, I., Motik, B., Wang, Z. The HermiT OWL Reasoner // In Proc. ORE 2012. Pp. 136-141.