Session 3.

Formalization and Simulation of Business Systems

VERIFICATION OF BUSINESS RULES USING LOGIC PROGRAMMING MEANS

Henrikas Pranevicius, Regina Miseviciene

Kaunas University of Technology, Department of Business Informatics
Kaunas, LT-44029, Lithuania
E-mail: {hepran, regina.miseviciene}@ktu.lt

In this paper development of the analysis tool for verification and validation of business rules is presented. Formalism for expressing the business rules is the predicates logic language. The rules are developed on the framework of Piece–Linear Aggregate (PLA) formalization. By the presented approach the PLA based business process specification is transformed to the set of predicate logic rules describing both the specification and the properties under investigation. The article also shows verification and validation of the business rules determined by the clauses of the first order predicates logic. The convenient interface of the system is presented. The interface allows competent user to interfere, find and correct mistakes. The presented approach is illustrated by the example.

Keywords*: business rules, verification, validation, predicates logic language*

1. Introduction

Business rules and constraints have become a basic topic in systems development and enterprise modelling. The essence of the business rules is to describe and verify aspects of the business functions. Business rules can appear in many different forms, both formal and informal. Formal methods for describing rules are relatively new and still being refined.

Systems designers have long been able to describe business in terms of the structure of the data that enterprise uses and the functions it performs. Unfortunately, the constraints under which the enterprise operates, they have tended to neglect [1-3].

Moreover, discussions on business rules frequently concern on that has been done in knowledge-based systems in such area as knowledge verification. There are significant numbers of articles on verification and validation constructs in knowledge-based systems. Many authors who write about the verification and validation in knowledge-based systems have their own definitions of the concepts [4-7].

Business rules are better understood, when techniques and tools are developed. The techniques include formal methods for describing rules, with tools translating these formalisms directly into other implementation constructs.

This paper employs integrated business process modelling and simulation framework – Piece-Linear Aggregate (PLA) – for formalization of business process and analysis of rules written for the process [8-9].

Special tools are created for execution of the aggregate models. Our approach is based on using the first order predicate logic and programming language Prolog for constructing of the executable specification [10-12]. By the presented approach the PLA based business process specification is transforming to the set of predicate logic rules describing both the specification and the properties under investigation. The resolution method implemented in Prolog is applied to analyse the rules.

The article also shows how the system executes verification and validation of business rules determined by the clauses of the first order predicates' logic. The convenient interface of the system is presented. The interface allows competent user of the problem domain to interfere, find and correct mistakes. The presented approach is illustrated by the example.

2. Business Process Modelling in the PLA Notation

This paper employs integrated modelling and simulation framework Piec-Linear Aggregate (PLA) – for analysis of business processes. On the ground of the developed PLA principles automatic analysis tools are created (see Figure 1).

MoSIS (Modelling, simulation, analysis and implementation suite) for business systems permits to perform main stages of analysis and design using formal methods. The main feature of this system is all stages of analysis, and design of business processes are performed based on single PLA mathematical scheme. The system consists of such sub-systems performing the following tasks: transformation of system models specified with

SDL, ESTELLE, UML languages to PLA model; validation and verification of PLA formal specifications; simulation; test creation; automated software implementation.

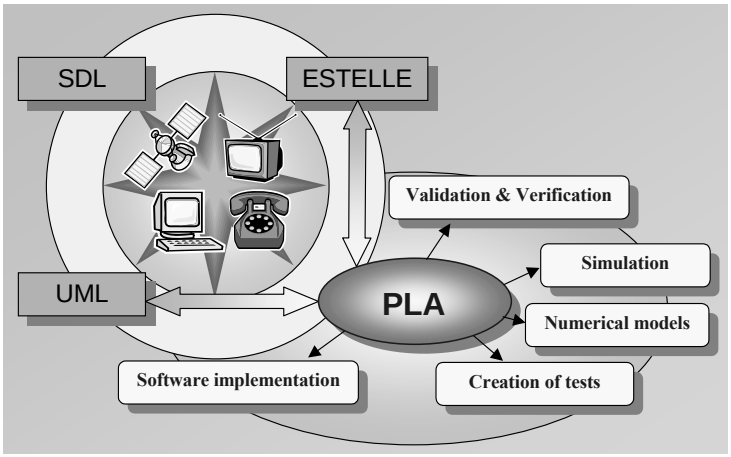

Fig. 1. *MoSIS – the verification and validation tool*

2.1. PLA definition

PLA is a special case of automaton models. These system described in any formalisms, one can associate a transition system of a set of states, a set of initial states, a set of discrete actions, a set of discrete steps $s' \xrightarrow{a} s$. Furthermore, from state s the system can move to state s' during a positive amount of time t in which no discrete action occurs. PLA model is extended automaton model. In the formalism business processes are modelled as compositions of aggregates. Each aggregate specifies interactions among its partners and each aggregate service a unique business process (e.g. processing a payment or shipping).

By the PLA approach each aggregate A is viewed a system $A = \langle X, Y, E, Z, H, G \rangle$. In the definition two sets (X and Y) present input and output signals that arrive or exit of the aggregate. The state Z of the aggregate is specified by discrete ($D = \{d_1, d_2, \ldots, d_p\}$) and continuous ($W = \{w_1, w_2, \ldots, w_f\}$) components. It is asserted that from state s the aggregate can instantaneously move to new state s' via the occurrence of event e_i (see Figure 2).

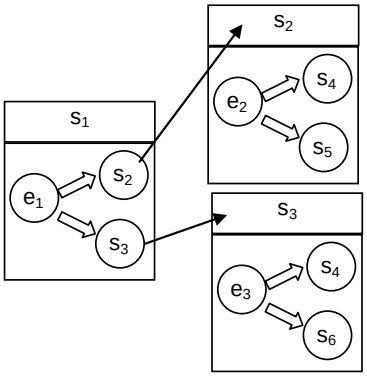

Fig. 2. *State transition using two events*

The events can occur when an input signal arrives at the aggregate or when a continuous component acquires a definite value. The events $E = E' \cup E''$ are defined as the sets of internal (E'') and external (E') events. Transition ($H : E \times Z \to Z$) and output ($G : E \times Z \to Y$) operators reflect one state to other. The aggregates of the system (of n elements) are connected by channels. Each channel connects a pair of contacts:

$$S = \langle A_1, A_2, \ldots, A_n, R \rangle,$$

where $R : X \to Y$ – is a finite set of channels, which represent connections between aggregates.

2.2. Example of PLA specification

Throughout this paper we will demonstrate an example which includes a part of on-line e-Books Sales process business model. By the model *Company* sells books to the public customers using an electronic

catalogue. *Customer* orders books mentioned in the catalogue. If the order is acceptable, *Order clerk* of the *Company* ships the books and *Invoice* to *Customer*.

PLA specification of the example consists of the system $S=(A1, A2, R)$, where $A1$ (*Customer*), $A2$ (*Company*) are aggregates of the business system; $R=\{r1, r2\}$ presents two channels r1 (the channel for sending documents or resources from *Customer* to *Company*) and r2 (the channel for sending documents or resources from *Company* to *Customer*). Figure 3 shows the structure scheme of the PLA model.

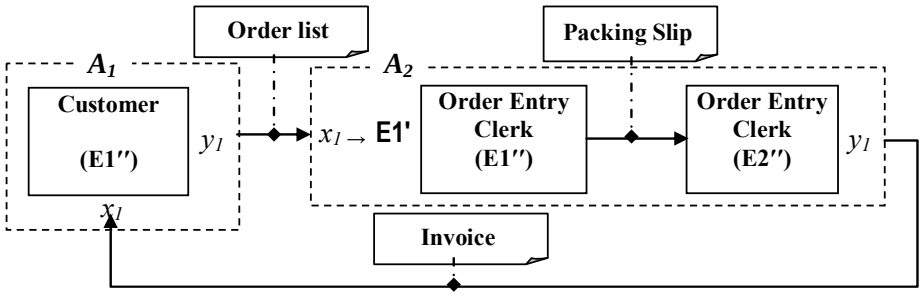

Fig. 3. *Structure scheme of the example*

Description of aggregate $A_1=\langle X, Y, E, Z, H, G\rangle$:

1. A set of input signals $X=\{x_1\}$, where x_1 is "*Invoice*" document as the input signal.

2. A set of output signals: $Y=\{y_1\}$, where y_1 is document "*Order list*" as output signal.

3. A set of external events $E'=\{e'_1\}$, where e'_1 – event describes the arrival of the "*Invoice"*.

4. A set of internal events $E''=\{e''_1\}$, where: e''_1 – event describes the forming of the "Order".

5. Discrete component of the state $v=\{k\}$, where k is amount of ordered books.

6. Continuous component of state: $z_v=\{w(e''_1)\}$, where: $w(e''_1)$ is moment when starts the event.

7. The initial state presents the forming of the new "*Order*" and amount of ordered books is 0. $w(e''_1)=1$; $k=0$.

8. Transition operator $H(e''_1)$ describes, that the internal event "*Order*" is formed and amount of ordered books is $k=1+random(5)$. The amount of the ordered books $y_1=k$ is sent to *Company* (in output operator $G(e''_1)$).

$H(e''_1)$:
$k=1+random(5);$
$w(e''_1)=1$

$G(e''_1)$:

$y_1=k$

Specification of other aggregates is presented in [10].

3. Rule Based Process Model Based on PLA

3.1. Rule sets as units of logic

Business rules can appear in many forms, both formal and informal. Last provides natural-language statements within a limited range of patterns. The formal expression of business rules needs either graphical or formal (logic-based) language. Special tools are created for execution of the formal models.

By our approach the rules are developed on the framework of Piece-Linear Aggregate (PLA) formalization. The PLA based business process specification is transforming to the set of predicate logic rules describing the specification. By our approach two parts of rules are constructing. One of the part corresponds to the business process specification, and the second part describes statements about the underlying business rules. The consistency of the rules is verified by the resolution method using logic programming language Prolog.

The logic programming language Prolog presents some advantages. The language based specification allows clear separation of logic and control parts and permits the specification to be more easily improved. Correspondingly, the logic component specifies rules, which are used in solving problems and control component determines the problem – solving strategy by means in which that rules are used adapted to new problems.

The suggested approach uses business process rules, based on PLA model. The most convenient way of creating the rules is to select an appropriate pattern from a short list of available patterns. The patterns we cover in the form of the following predicates:

The predicate $QX\_name\left(input\_p, input\_v, w_1,\ldots,w_f, d_1,\ldots,d_z\right)$ acquires True value when arrive input signal with value $input\_v$ at the contact $input\_p$ of an aggregate with continuous and discrete coordinates of the aggregate state.

Similarly predicate $QY\_name\left(output\_p, output\_v, w_1,\ldots,w_f, d_1,\ldots,d_z\right)$ acquires *True* value when a signal with value $output\_v$ is generated at the pole $output\_p$ of aggregate.

The following predicate $QW\_name\left(w_1,\ldots,w_f, d_1,\ldots,d_z\right)$ acquires *True* value when an internal event occurred.

$P(w_1, \ldots, w_f, d_1, \ldots, d_s)$. The predicate is *True*, when are satisfied conditions for the state coordinates.

These predicates are used in rules of transition and output operators. They depend on occurrence of external and internal events. The following rules are possible:

1. The external event changes the state coordinates and the output signal is sending:

$$QX\_name\ (x1,\ldots,xi,\ldots,xm, w1,\ldots,wf, d1,\ldots,ds) \wedge P(w1,\ldots,wi,\ldots,wf, d1,\ldots,ds) \rightarrow$$
$$QY\_name\ (y1',\ldots,yi',\ldots,yn', w1',\ldots,wf', d1',\ldots,ds').$$

2. The external event changes the state coordinates without the output signal:

$$QX\_name\ (x1,\ldots,xi,\ldots,xm, w1,\ldots,wf, d1,\ldots,ds) \wedge P(w1,\ldots,wi,\ldots,wf, d1,\ldots,ds) \rightarrow$$
$$QW\_name\ (w1',\ldots,wi',\ldots,wf', d1',\ldots,ds').$$

3. The internal event changes the state coordinates and the output signal is sending:

$$QW\_name\ (w1,\ldots,wi,\ldots,wf, d1,\ldots,ds) \wedge P(w1,\ldots,wi,\ldots,wf, d1,\ldots,ds) \rightarrow$$
$$QY\_name\ (y1',\ldots,yi',\ldots,yn', w1',\ldots,wf', d1',\ldots,ds').$$

4. The internal event changes the state coordinates without the output signal:

$$QW\_name\ (w1,\ldots,wi,\ldots,wf, d1,\ldots,ds) \wedge P(w1,\ldots,wi,\ldots,wf, d1,\ldots,ds) \rightarrow$$
$$QW\_name\ (w1',\ldots,wi',\ldots,wf', d1',\ldots,ds').$$

In the aggregate system the aggregates are connected by communication channels. The channels communication of system is describing using the following predicates: the first predicate OY_name_1 describes the aggregate (that marked as $name\_1$) which generates an output signal on the $output\_p$ contact, the next predicate of the aggregate (that marked $name\_2$) on which arrives an input signal on the $input\_p$ contact.

Communication of output pole $output\_p$ of aggregate with name $name\_1$ and input pole of aggregate with $name\_2$ is described by a logic formula:

$$QY\_name\_1\left(output\_p,\ output\_v,\ w_1^1,\ldots,w_f^1, d_1^1,\ldots,d_z^1\right)\rightarrow$$
$$QX\_name\_2\left(input\_p,\ input\_v,\ w_1^2,\ldots,w_f^2, d_1^2,\ldots,d_z^2\right).$$

Here $w_1^1,\ldots,w_f^1, d_1^1,\ldots,d_z^1,\ w_1^2,\ldots,w_f^2,\ d_1^2,\ldots,d_z^2$ are continuous and discrete coordinates of the aggregates states.

3.2. Example of rule based specification

In the example of aggregate specification presented above transition operator $H(e_1'')$ describes, that the internal event "*Order*" is forming and amount of ordered books is equal $1+random(5)$.

$H(e_1''):$
$k = 1 + random(5);$
$w(e_1'') = 1$

$G(e_1''):$

$y_1 = k$

By the presented approach the operators above are transformed to the rule:

$QW\_A1(w1, k) \wedge w1 \neq 0 \rightarrow$
$next\_k = random(5) + 1 \wedge next\_w1 = 1 \wedge QY\_A1(next\_k, next\_w1, next\_k)$

The rule declares: if aggregate's with name "*A1*" coordinates are (*w1* and *k*, where *w1* is internal event "Order" is formed and *k* is amount of ordered books) and the internal event is not equal 0, then the new amount of ordered books is generating and new order of books is preparing.

4. Rules for Verification of Business Rules

Verification and validation of the business process consists of verifying that a model satisfies the properties by exploring all its reachable states. This requires that this state space be finite and tractable. The reachable states analysis is based on the concept of global state, which is a composition of local states of aggregates. By the given approach reachable states tree is generated and then analysed. Analysing the reachable states tree, the following properties can be observed: 1) completeness, 2) deadlock freeness, 3) tempo blocking freeness, 4) liveliness, 5) termination or cyclic behaviour, 6) boundedness, etc. Invariant approach is used to analyse individual properties of the aggregate models. The invariant is an assertion describing correct system state and remaining true in spite of events taking place and of transitions from one state to another.

By our approach the set of rules of general and individual properties under investigation are the following:

1. Initial state statement presents what concrete initial values must to be of all aggregates (global) coordinates:

$Q\_global\ (\ w0_1, \ldots, w0_{f_1}, d0_1, \ldots, d0_{s1}, \ldots, w0_1, \ldots, w0_{f_n}, d0_1, \ldots, d0_{s_n}\ )$.

2. Final state statement also presents what concrete terminal (*T*) values must to be of all aggregates (global) coordinates:

$Q\_global\ (\ wT_1, \ldots, wT_{f_1}, dT_1, \ldots, dT_{s1}, \ldots, wT_1, \ldots, wT_{f_n}, dT_1, \ldots, dT_{s_n}\ )$.

3. Boundedness rule demonstrates boundaries for discrete coordinates of the all aggregate state:

$Q\_global\ (\ w^{1_1}, \ldots, w^{1_{f1}}, d^{1_1}, \ldots, d^{1_{s1}}, \ldots, w^{n_1}, \ldots, w^{n_{fn}}, d^{n_1}, \ldots, d^{n_{sn}}\ ) \rightarrow$
$D = \{d_1, \ldots, d_j, \ldots, d_s\}$

there predicate $D = \{d_1, \ldots, d_j, \ldots, d_s\}$ demonstrates boundaries for the discrete coordinates.

4. Deadlock statement shows, that when all the continuous coordinates in global state are equal 0, then the execution is False.

$Q\_global\ (\ 0, \ldots, 0, d^{1_1}, \ldots, d^{1_{s1}}, \ldots, 0, \ldots, 0, d^{n_1}, \ldots, d^{n_{sn}}\ )$

5. Invariant rules demonstrate general (or individual) conditions:

$Q\_global\ (\ w^{1_1}, \ldots, w^{1_{f1}}, d^{1_1}, \ldots, d^{1_{s1}}, \ldots, w^{n_1}, \ldots, w^{n_{fn}}, d^{n_1}, \ldots, d^{n_{sn}}\ ) \rightarrow$
$I = \{d_1, \ldots, d_j, \ldots, d_s\}.$

5. Verification of Business Rules

5.1. Verification and validation tool

Verification and validation tool is created on the means of logic programming language Prolog. The tool provides convenient user interface for automatic analysis of the business rules.

The verification and validation tool consists of the following components: translator, knowledge base and user interface (see Figure 4).

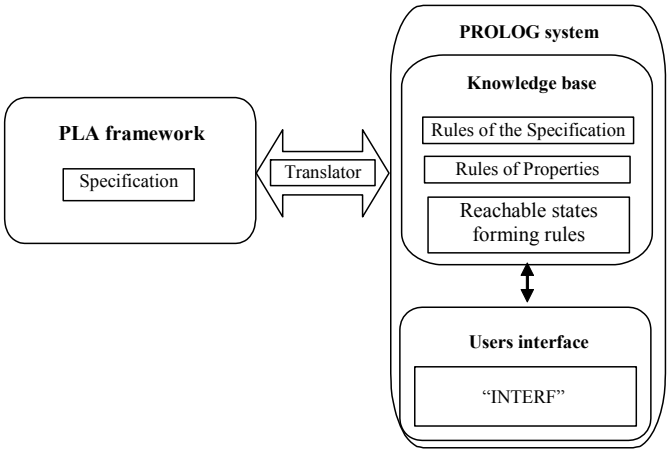

Fig. 4. *Structure of the Verification and Validation tool*

The translator automatically generates Prolog rules from the aggregate specification. The translator also provides intelligent editor. Predicate logic system Prolog is used for analysis of the rules base. The considerable attention is paid to user interface and user dialog. The user of the verification and validation tool have ability not only to derive conclusion from existing data in the knowledge base, but also to give explanations *how* and *why* corresponding solutions have been made.

5.2. User interface of the tool

In addition to other features the presented tool has to provide ability for user to interfere in certain stage of PLA specification analysis.

For example, in certain stage of analysis user can ask the system to explain how one or another presented solution of the problem was derived, if it is not clear (therefore, question *how* is implemented). While asking question *how* expert system will present the list of rules used in goal satisfying process. The latter list is presented in direct order.

During problem solution process system may request additional information, which is needed to make conclusion. If it is not clear to user why the system gave the corresponding request, the user may also ask the system question *why*. In this case system will present inference chain stipulated by question *why*, in other words, the user will receive the list of rules used to find the goal. In *how* question's case expert system presents the list of rules in direct order but in *why* question case the list of rules will be presented in indirect order. The rule, which satisfies final goal, is at the bottom of the presented list.

The difference between mentioned questions is presented on Figure 5.

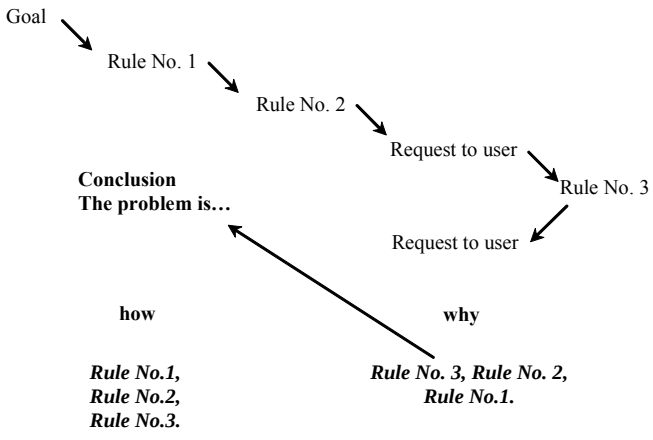

Fig. 5. ***How*** *and* ***why*** *questions*

5.3. Example of user interface

Example of the user interface shows dialog between system and user during the specification analysis. The example (see Figure 6) of the user interface shows dialog between system and user during analysis. By the interface user can ask the system to explain *how* or *why* presented solution of the problem is derived. In question case *why* the system lists predicates with discrete and continuous coordinate values in indirect order of inference.

```
INTERF is a simple PROLOG shell.
Please, enter one of the commands: help. load. analyze. how. or why.
> load.
Please, enter the name of knowledge base in single commas
 (for example: 'books_order.pro'.):
>' books_order.pro'.
> analyze.
-->1+1 - [] - [] - [] - 0 - 35
| |
| W1
| |
? (yes/why/end)  >> yes.
| -->2+1 - 1 - [] - 10 - 1 - 25
| | |
| | W1
| | |
? (yes/why/end)  >> why.
   r(1,1,[],10,1,25,2)
   qw(1,1,[],10,1,25)
   qt(1,[],[],[],0,35)
   r(1,[],[],[],0,35,1)
| | -->3+1 - [1 | 1] - [] - [10 | 10] - 2 - 15
| | | |
| | | W1
| | | |
```

Fig. 6. *Example of the user interface*

6. Conclusions

In this paper we set out for using a business rule model to represent control flow of business processes. In particular we demonstrate how the business rule based model defines business model and the state transitions of the business model. The formalism for expressing the business rules is the predicates logic language. The rules are developed on the framework of Piece-Linear Aggregate formalization. By the presented approach the PLA based business process specification is transformed to the set of predicate logic rules describing both the specification and the properties under investigation. The article also shows verification and validation of business rules. The convenient interface of the system is presented as well.

Our presented approach has some advantages:

- This allows clear separation of business process logic and rules that constrain the business process.
- Furthermore application of PLA approach allows automated analysis of general properties (completeness, deadlock freeness, termination, boundedness) and individual properties of the rule based specifications.
- The user of the verification and validation tool has ability not only to derive conclusion from the existing data, but also to give explanations how and why corresponding solutions have been made.
- The approach permits the rule based specification to be more easily improved modifying rules when changing business policies without the need to change the process definitions.

References

1. Vanthienen, J., Goedertier, S. and Mues, C. Experiences with Modelling and Verification of Regulations. In: *International Workshop on Regulations Modelling and their Validation & Verification (REMO2V '06), in conjunction with the 18th Conference on Advanced Information System Engineering (CAiSE '06), Luxembourg, 2006.* Luxembourg, 2006, pp. 793-799.

2. Goedertier, S., Vanthienen, J. Rule-based business process modelling and execution. In: *Proceedings of the IEEE EDOC Workshop on Vocabularies Anthologies and Rules for the Enterprise (VORTE 2005). CTIT Workshop Proceeding Series (ISSN 0929-0672), 2005–* http://perswww.kuleuven.be/~u0041863/articles/goedertierVORTE2005.pdf.
3. *Defining Business Rules. What are they really? The Business Rules Group: Final Report revision 1.3, July, 2000* – www.businessrulesgroup.org/first_paper/br01c1.htm.
4. Preece, A. Foundation and Application of Knowledge Base Verification, *International Journal of Intelligent Systems,* Vol. 9, 1994, pp. 683-701.
5. Plant, R. T. *Methodologies for the development of knowledge-based systems, 1982-2002: The Knowledge Engineering Review, Cambridge University Press 18.* Cambridge, 2003, pp.47-81.
6. Tsai, W.T., Vishnuvajjala, R., Zhang, D. Verification and validation of knowledge-based systems. *IEEE transactions on knowledge and data engineering*, Vol. 11, No 1, 1999, pp. 202-212.
7. Tsai, J.P., Liu, A., Juan, E., Sahay, A. Knowledge-Based Software. Architectures: Acquisition, Specification and Verification, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No 1, 1999, pp. 187-201.
8. Pranevicius, H. *Aggregate approach for specification, validation, simulation and implementation of computer network protocols: Lecture notes in Computer Science No502.* Springer-Verlag, 1991, pp. 433-477.
9. Pranevicius, H. Formalization and simulation of business process. In: *Modelling and Simulation of Business Systems. International Conference.* 2003, pp. 198-202.
10. Pranevicius, H., Misevicius, P.V., Miseviciene, R. Transformation of aggregate specifications to the predicate logic models. In: *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling and Simulation.* 2003, pp. 378-384.
11. Pranevicius, H, Miseviciene, R, Janusauskaite, Z. Using piece-linear aggregate model for resource-event-agent business process analysis. In: *Information technologies 2006, International conference.* 2002, pp. 484-490.
12. Pranevicius, H, Miseviciene, R, Milciute, V. Use of aggregate specification and logic programming for knowledge base validation and verification. In: *International Conference on Modelling and Simulation of Business Systems.* 2003, pp. 203-208.

MODELLING OF RAILWAY SCHEDULE IN TEMPORAL DATABASES

Eugene Kopytov[1], Vasilijs Demidovs[1,2], Natalia Petukhova[2]

[1] Transport and Telecommunication Institute
Lomonosova 1, Riga, LV-1019, Latvia
Fax: +371 67100660. E-mail: kopitov@tsi.lv

[2] State Join-Stock Company "Latvian Railway"
Turgeneva 21, Riga, LV-1547, Latvia
Fax: +371 67234366. E-mail: dem@ldz.lv

The given paper deals with the issues of building a temporal database for the railway schedule. For the mathematical description of the railway schedule system, we suggest to use set theory and algebra of logics. We define the periodicity parameters and temporal elements and formulate the rules of their calculation. We also give practical examples on defining the dates on which the set version of the train schedule becomes actual.

Keywords*: railway transportation, schedule, temporal database, statement of comparison, periodicity*

1. Introduction

The given paper considers the task of supporting the railway schedule in information systems (IS). The main problems of holding the schedule in the database are connected with the great number of its variants conditioned by season changes of the railway changes, different days of the week, planned and unplanned repair works, re-arranging holidays and weekdays and other factors. The schedule-supporting task can be solved by means of a calendar stating the ply of each train on each day of the year. However, the number of entries describing on what day, at what time and at what station each train will be having a stop, will approximately equal the product of the number of trains by the average number of stops and by the number of days covered by the schedule. With this approach we will need about 2 million records to hold the local Latvian railway schedule in the relational database and for larger railway companies the volume of the corresponding database will increase by about ten times. Moreover, holding the schedule for several years will increase the number of records up to hundreds of millions. However, the main problem of the above approach lies no so much in the volume of the stored data but in the trustworthiness of these data and in the operativeness of entering changes in the schedule. When these changes are frequent, the task becomes much more complicated.

The above circumstances have served the basis for carrying out the given research, in which we suggest the principle of realizing the schedule system on the basis of the temporal database principles [1, 2]. In the given paper, the authors consider the issues of building a temporal database (TDB) of the railway schedule with the use of the periodicity parameters and temporal elements. The suggested approach is applied in designing the informational system of the inland Latvian railway train schedule.

2. Designing a Temporal Database Model in the System of the Railway Schedule

Principles of designing a temporal database assume that the database holds all objects versions and access to all of them is available for the user [1]. The TDB peculiarities are first of all connected with using the stored objects of the temporal database for the VIEW. The main concept in the temporal model is lifespan. It means the duration of time period associated with the existence of the object in a concrete state. The example of temporal relation "Train schedule" constructed for different time periods is shown in Fig. 1.

The authors also use the relational environment for developing the system of the railway schedule. To realize temporal principles in the frame of the relational model, the authors of the paper suggest an open model with an Abstract Object Identifier (AOID) [1]. The given model provides the support of the multi-version of the object and minimizing expenses for "imitation" of DELETE and UPDATE operations with the transition of the old version to "shadow area". In this model lifespan of the object is described through life spans of all its properties, defined in different relations and having time attributes *DATE_START* (time of lifespan start) and *DATE_STOP* (time of termination). Realization of the temporal logics using some elements of active databases in analytical applications and real time tasks has been performed on the basis of triggers and Java Stored Procedures [1, 2].

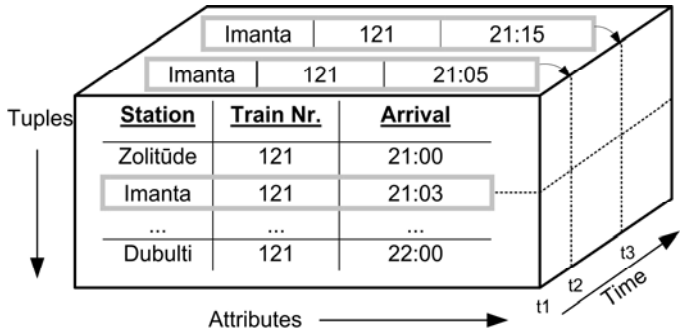

Fig. 1. *The three-dimensional temporal relation "Train Schedule"*

Let us consider a mathematical description of the suggested model. We have the totality of m objects (trains), which is described by the relation $RO(A_1, A_{2,}, ..., A_m)$, where $A_1, A_2, ..., A_m$ are the relation attributes characterizing the object properties, e.g. data about the train stops. A classic relational approach assumes that the relation RO has m of tuples— one for each object. In the period of the database lifespan the stored objects (train schedules) are subject to constant changes, correspondingly there change the tuples of the relation RO, which correspond to the changed objects. The outdate information about the changed objects is no longer kept in the database (DB). To keep the complete historic information about the DB objects we need to support several object versions in the relation, which may be achieved by including some additional attributes in the relation RO. To keep the complete historic information about the objects, let us use the relation of the type:

$$R(A_1, A_2, ..., A_m, AOID, t_s, t_e), \tag{1}$$

where $AOID$ is an abstract object identifier; t_s and t_e are correspondingly *DATE_START* and *DATE_STOP* of the stored object versions lifespan.

One of the serious problems occurring in the considered IS of the railway schedule is connected with the overlapping of the objects' lifespans, when one schedule overlaps another (see Fig. 2). The peculiarity of the given situation is that, the object when changed cannot lose the actuality of its state either in the past or in the future – for a certain period only it is substituted by another version. As a result, more than one actual tuple describing different versions of the same object's property can exist at one time. The latter contradicts the very idea of TDB, therefore it is necessary to trace and settle such situations by making special data queries.

To explain the considered case we refer to example shown in Fig. 2. We assume that the variable t_n means the moment of reviewing. The season railway schedule is fixed in February for the period from the moment t_1 to t_6. Later on in March, the schedule is changed for the period from the moment t_2 to t_4, and then in April, we make one more change for the period from t_3 to t_5. For the inquiry of the train traffic at the moment t_{n1}, we'll get the schedule fixed in February for $[t_{1,} t_6]$, at the moment t_{n2} – the schedule fixed in April for $[t_3, t_5]$, and at the moment t_{n3} we again have the schedule for $[t_{1,} t_6]$.

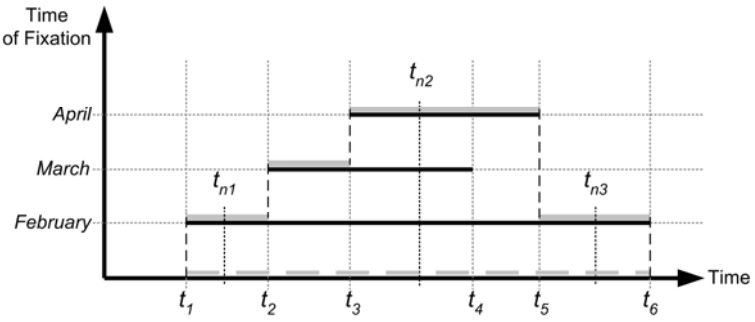

Fig. 2. *Crossing of Object Lifespans in DB of the Railway Schedule System*

3. Using the periodicity principle in the railway schedule

In practice, the train may have simultaneously several valid schedules for a long period. In this situation, every schedule is fixed with the account of the days of traffic and with the use of different periodicity

characteristics, for example: on the weekdays, on the rest-days, on the particular rest-days, on the first weekday, on the last day of the weekend (which sometimes may be Monday), on the even, on the odd days, etc.

Let us take as an example a multi-variant train schedule presented in Fig. 3. A we can see, for the train numbered *n* there are two basic schedules, one for the weekdays – *wd*, the second one for the rest-days – *rd*. Both schedules are actual in the time period $[t_1, t_4]$, but in different weekdays. Then, for repair works we make a change in the schedule of the given train on Tuesdays and Thursdays for the time period from t_2 to t_3. Thus, in the time period $[t_2, t_3]$ we already have three actual schedule versions valid for the days, which correspond to the characteristics *wd*, *rd* and *Tue,Thu*, with only one schedule version *wd* or *rd*, or *Tue,Thu* valid for one day.

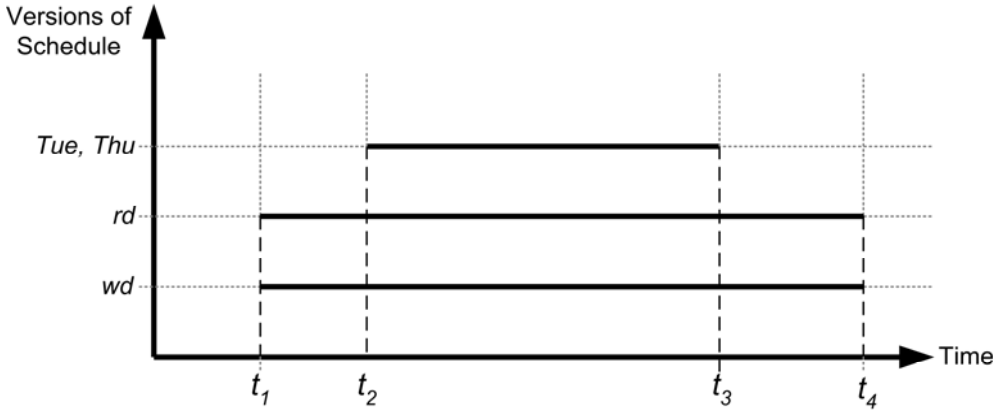

Fig. 3. *Variety of the train schedule valid versions: wd, rd and Tue, Thu*

We shall illustrate the former statement with the diagram in Fig. 4, in which the axis "Day of Week" shows the weekdays in the following sequence: 1 – Monday, 2 – Tuesday, etc. The granularity level of this schedule equals one day; therefore, we can state that time in our system has a discrete character and the versions possess the property of periodicity. As we can see in the figure, on different weekdays, one version from the three is the actual train schedule, and the other two are in the "shadow zone" at the moment [1]. Here, the train schedule for Tuesdays and Thursdays (characteristic *Tue,Thu*) overlaps the weekday schedule with the characteristic *rd* (see Fig. 4).

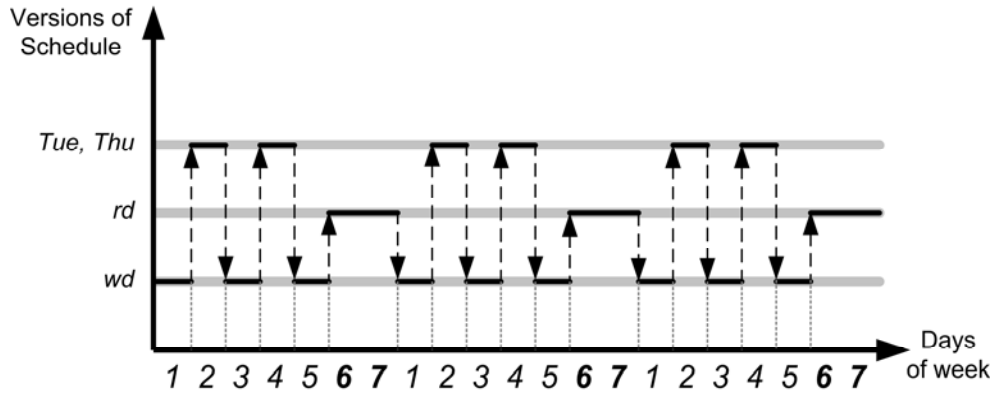

Fig. 4. *Periodicity in the train schedule*

However, except periodical versions of the train schedule, there may be one-time schedule changes. These are referred to holidays and additionally fixed holidays or cancelled holidays. Point out such case of the schedule change when a weekday changes place with a rest-day. Thus, for example, in 2007, in Latvia, Monday April 30 (weekday) changed place with Saturday April 14 (rest-day). As a result, the trains of the April 30 schedule plied according to the April 14 (rest-day) schedule and visa versa. Such changes or special fixations sometimes cause paradoxes. For example, in 2007 Monday November 19 was fixed as an additional rest-day and all trains on this day plied according to the November 18 (rest-day) schedule, which caused the periodicity failure as it is shown in Fig. 5. It should be added that some trains with the periodicity of the Sunday day were cancelled on November 18 due to the fact that a regular Sunday day is also the last rest-day of the week and additional Sunday day trains are fixed to carry passengers from traditional places of rest on these days.

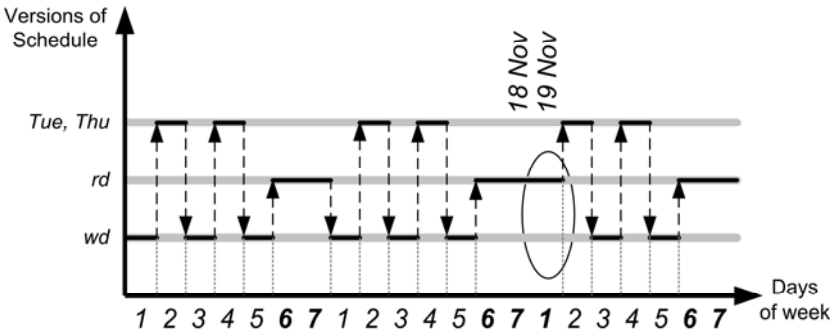

Fig. 5. *Failure in the periodicity of the train schedule*

For the correct work of the schedule system with the account of all the above non-trivial rules, we need a clear formalization of the task setting and describing the schedule temporal model. For this, we will take a group of sets characterizing the rules of forming the trains' traffic periodicity, the railway objects and the schedule system as a whole.

4. Formalization of the Schedule Model

4.1. Mathematical description of the railway schedule system

To describe the objects and the rules we will take a number of basic sets:

$S = \{s_1, s_2, \ldots, s_k\}$ is a set of all railway stations, where k is the power of set *S*;

$N = \{n_1, n_2, \ldots, n_m\}$ is a set of the railway trains where m is the power of set N;

We will define the train with number $n \in N$ by the vector of the following type:

$v = \left\{< s_1^{(n)}, T_1 >, < s_2^{(n)}, T_2 >, \ldots, < s_\alpha^{(n)}, T_\alpha >\right\}$, where the couple $< s_j^{(n)}, T_j >$ defines the *j*-th train stop, in particular, the name of the station $s_j^{(n)} \in S$ and the time of the train departure T_j; α is the number of the train stops on the schedule v.

For the train numbered n, we can keep several schedule versions characterized by the validity period, time of fixation and periodicity. Correspondingly, we take the following schedule parameters:

t_s is start time when the schedule version comes into action;

t_e is end time when the schedule version expires;

t_{fix} is fixation time, i.e. the moment of entering the given version of the train schedule in the database;

C is the periodicity characteristic used to define the days for which the given schedule will be valid.

Characteristic C presents a logical expression consisting of one or more statements of comparison, which are connected by the logical operations $\vee, \wedge$ and $\neg$ (see [4]). Each statement is an elementary parameter of the p_i periodicity and defines belonging of the day to a particular group, for example, "the day is an even day of the week"; it may be true or false. Elementary periodicity parameters form such a set of attributes on which base we can build logical expressions setting any more complicated periodicity.

We are going to consider different variants of the periodicity parameter given by the set $P = \{p_1, p_2, \ldots, p_\pi\}$ where π is the set power; the set elements are the following statements: *ed* is "any weekday" (used in case of daily train ply); *wd* is "work day" (used in case of work days ply); *rd* is "rest-day"; *Mon, Tue, Wed, Thu, Fri, Sat, Sun* are "the corresponding weekday"; *pd* is "the even day of the week"; *nd* is "the odd day of the week"; *lh* is "the last rest-day of the week"; *fw* is "the first work day of the week"; etc.

To identify a concrete version of the schedule for the train with number $n \in N$ we will use the tuple $\left\langle n, C, t_{fix}, t_s, t_e \right\rangle$. Then, the *i*-th version $v_i^{(n)}$ of the schedule for the train number $n \in N$ will be defined by the tuple:

$$v_i^{(n)} = \left\{\left\langle n, C, t_{fix}, t_s, t_e \right\rangle, \left\langle s_1^{(n)}, T_1 \right\rangle, \left\langle s_2^{(n)}, T_2 \right\rangle, \ldots, \left\langle s_\alpha^{(n)}, T_\alpha \right\rangle\right\}. \quad (2)$$

Let $V^{(n)} = \{v_1^{(n)}, v_2^{(n)}, ..., v_g^{(n)}\}$ be the set of all versions of the schedule for train with number $n \in N$, where g is the set power equal to the number of versions of the*n*-th train schedule;

then $V = \{V^{(n_1)}, V^{(n_2)}, ..., V^{(n_m)}\}$ is a set of all versions of all trains' schedules.

Then, we introduce the sets characterizing exceptions in the periodicity fixed directly.

$EX(C) = \{ex_1^{(C)}, ex_2^{(C)}, \ldots, ex_\theta^{(C)}\}$ is a set of days-exceptions $ex_j^{(C)}$, which are given the C characteristic, where θ is the power of set $EX(C)$. Special example of the set $EX(C)$ are given by the set $EX(wd) = \{ex_1^{(wd)}, ex_2^{(wd)}, ..., ex_w^{(wd)}\}$ setting additionally fixed work days by the periodicity characteristic $C = wd$ and the set $EX(rd) = \{ex_1^{(rd)}, ex_2^{(rd)}, ..., ex_q^{(rd)}\}$ determining additionally fixed rest-days with the periodicity characteristic $C = rd$;

$EX = \{\langle C_1, EX(C_1)\rangle, \langle C_2, EX(C_2)\rangle, \ldots, \langle C_l, EX(C_l)\rangle\}$ is the set of all exceptions for similar periodicity characteristics, where *l* is the power of the set EX; the elements of set EX re-define the periodicity attributes for the dates indicated in $EX(C)$.

$Z = \{z_1, z_2, ..., z_\mu\}$ is the set of dates transferred (changed) in the trains' schedule, where z_i is a couple of days $\langle d_1, d_2 \rangle$ determining that the schedule fixed for d_1 date is now fixed for the d_2 date; μ is the power of set Z.

4.2. Formalization of the periodicity model

To define the time in which the version of a particular train schedule becomes actual for the whole period of its lifespan, we will use the notion of temporal elements *TE* [3]. Element $TE(C)$ for $C = wd$ is a finite union of intervals *t* during *wd* is active: $t_1^{(wd)} \cup ... \cup t_\lambda^{(wd)}$ and $TE(wd)$ also can be denoted by the set $\{t_1^{(wd)}, ..., t_\lambda^{(wd)}\}$. In this way $TE(C)$ for $C = wd$, $C = rd$ and $C = Tue \vee Thu$ (see Fig. 4 and Fig. 5) can be defined as follows:

$$TE(wd) = \{t_s, ..., [64,68], [71,75], ..., t_e\}, \quad (3)$$

$$TE(rd) = \{t_s, ..., [69,70], [76,77], [83,84], ..., t_e\}, \quad (4)$$

$$TE(Tue \vee Thu) = \{t_s, ..., [65,65], [67,67], ..., t_e\}, \quad (5)$$

where [64,68] is the time interval expressed in days of the year (64 – Monday, 68 – Friday, t_s and t_e are denoted as the start and end points of the interval $t = [t_s, t_e]$.

As it is seen from the formulae (3)–(5), temporal elements *TE* of the schedule versions possess periodicity, therefore, to get the whole set of the *TE* values, we introduce the extension function $Ext(C)$ [3]. E.g., if $C = Mon \vee Wed \vee Fri$ is the definition of Monday, Wednesday and Friday, $Ext(C)$ gives as output the set of all Mondays, Wednesdays and Fridays in the time interval of the following schedule version. The periodic event in the time range $[t_1, t_4]$ will be presented as a pair $\langle [t_1, t_4], C \rangle$, and the set of the $TE(C)$ values in the given time range as the extension function Bounded Extension in the format $BExt([t_1, t_4], C)$. With the employment of the used definitions and functions, we define the temporal element for the periodicity characteristic C in the time range $[t_s, t_e]$ as follows:

$$TE(C) = BExt([t_s, t_e], C) \cup EX([t_s, t_e], C) - EX([t_s, t_e], \neg C), \quad (6)$$

where $EX([t_s, t_e], C) \subseteq EX(C)$ is the set of exceptions of the periodicity characteristic C acting in the period of the schedule version validity: $EX([t_s, t_e], C) = \{ex^{(C)} \mid ex^{(C)} \in [t_s, t_e]\}$;

$\neg C$ is the set of days which are subject to the change of the periodicity attributes except the C characteristic.

Thus, the expressions (3), (4) and (5) will look as follows:

$$TE(wd) = BExt([t_1, t_4], Mon \vee Tue \vee Wed \vee Thu \vee Fri) \cup EX([t_1, t_4], wd) - EX([t_1, t_4], rd); \quad (7)$$

$$TE(rd) = BExt([t_1, t_4], Sat \vee Sun) \cup EX([t_1, t_4], rd) - EX([t_1, t_4], wd); \quad (8)$$

$$TE(Tue \vee Thu) = BExt([t_2, t_3], Tue \vee Thu') - EX([t_2, t_3], Mon \vee Wed \vee Fri \vee Sat \vee Sun). \quad (9)$$

Example. Calculation of the temporal element for even days of the month, except Tuesday, can be presented as follows:

$$TE(pd \wedge \neg Tue) = BExt([t_2, t_3], pd \wedge \neg Tue) - EX([t_2, t_3], nd \vee Tue). \quad (10)$$

For periodicity characteristics we have formulated the given essential ***rule***: coverings of the temporal scale, which correspond to the periodicity characteristics C and $\neg C$ never overlap, in other words $Ext(C) \cap Ext(\neg C) = \varnothing$. Examples of calculating the periodicity characteristic $\neg C$ are illustrated in table 1.

Table 1. Examples of performing a complement operation on $\neg$ on the *C* statement

Statement C	Statement $\neg C$
wd	rd
rd	wd
Mon	$Tue \vee Wed \vee Thu \vee Fri \vee Sat \vee Sun$
$Tue \vee Thu$	$Mon \vee Wed \vee Fri \vee Sat \vee Sun$
$pd \wedge \neg Tue$	$nd \vee Tue$

It is significant that we will be using temporal elements in order to define actual versions of the trains' schedule. Thus, the version of schedule v_i for train number $n \in N$, defined by expression (2), may be specified using a temporal element:

$$v_i = \left\{ \left\langle n, TE(C, t_{fix}) \right\rangle, \left\langle s_1^{(n)}, T_1 \right\rangle, \left\langle s_2^{(n)}, T_2 \right\rangle, ..., \left\langle s_\alpha^{(n)}, T_\alpha \right\rangle \right\}, \quad (11)$$

where temporal element $TE(C, t_{fix})$ is formed with the account of the fixation time t_{fix} of schedule v_i and defines all the dates for which the given version is actual.

Temporal element $TE(C_i, t_{fix}^{(i)})$, besides periodicity attribute characteristics and date changes, takes into account the cases of the schedules' overlap. When spotting such conflicts, similar dates are excluded in that of the two temporal elements, which finds correspondence with the earlier fixation time t_{fix}. Then, the definition $TE(C_i, t_{fix}^{(i)})$ presents a recursive function. Recursive essence is also found in the fact that in calculating of one temporal element there are used the temporal elements of later versions, which elements, in turn, are defined in the same way:

$$TE(C_i, t_{fix}^{(i)}) = TE(C_i) - \bigcup_{v_j \in V(v_i)} TE(C_j, t_{fix}^{(j)}), \quad (12)$$

where $V(v_i)$ is the set of versions of the schedule for *n*-th train, which time periods cover partly the period of version v_i and elements of $V(v_i)$ are fixed later than version v_i; the set $V(v_i)$ can be found by formula:

$$V(v_i) = \left\{ v_j \mid v_j \in V^{(n)}; [t_s^{(i)}, t_e^{(i)}] \cap [t_s^{(j)}, t_e^{(j)}] \neq \varnothing \wedge t_{fix}^{(j)} > t_{fix}^{(i)} \right\}; \quad (13)$$

$\bigcup_{v_j \in V(v_i)} TE(C_j, t_{fix}^{(j)})$ defines the set of dates, for which versions of the schedule $v_j \in V(v_i)$ are actual and which overlap the schedule v_i and have higher priority.

The condition of recursion termination is fulfilled when such a version of the schedule $v_{last} \in V^{(n)}$ for train with number *n* is reached for which we will not spot any other version of this train schedule $v_j \in V^{(n)}$, satisfying the condition: $[t_s^{(last)}, t_e^{(last)}] \cap [t_s^{(j)}, t_e^{(j)}] \neq \varnothing \wedge t_{fix}^{(j)} > t_{fix}^{(last)}$, here $j \neq last$. Then, from (13) we have $V(v_{last}) = \varnothing$.

Let us note that after the correction of $TE(C)$ according to the formula (12), temporal elements of different schedule versions for one train never overlap, i.e. we satisfy the rule:

$$TE(C_i, t_{fix}^{(i)}) \cap TE(C_j, t_{fix}^{(j)}) = \varnothing \text{ for } i, j = 1,2,...g; \quad i \neq j.$$

To take into account all date changes, the temporal elements of versions for schedule $v_i \in V^{(n)}$ should be transformed as follows:

$$TE(C_i, t_{fix}^{(i)}) = TE(C_i, t_{fix}^{(i)}) \cup z_i^- - z_i^+; \qquad i = 1,2,..., g, \tag{14}$$

where

$z_i^- = \{d_1 \mid z =< d_1, d_2 >, z \in Z, d_2 \in TE(C_i, t_{fix}^{(i)})\}$ is the set of dates, which should be excluded from the temporal element of version v_i since they subject to change;

$z_i^+ = \{d_1 \mid z =< d_1, d_2 >, z \in Z, d_1 \in TE(C_i, t_{fix}^{(i)})\}$ is the set of dates, which should be included in the temporal element of version v_i since its temporal element $TE(C_i, t_{fix}^{(i)})$ comprises the date which schedule is standard for the date to be updated.

Fig. 6 illustrates the schedule change from Friday April 13 to Saturday April 14, which means excluding April 14 from the temporal element of schedule #2 for the rest-days (*rd*) and including it in the temporal element of schedule #1 for the weekdays (*wd*).

Function $BExt([t_s, t_e], C)$ defines the dots of the time period within the diapason $[t_s, t_e]$, responding the periodicity characteristic C. The basis of the function is checking every dot $x \in [t_s, t_e]$ for correspondence to the logical expression C. This check means including the reviewed date x in every statement incorporated into C; as a result, we get a certain complicated statement $C(x)$ built on the base *C* for day *x*.

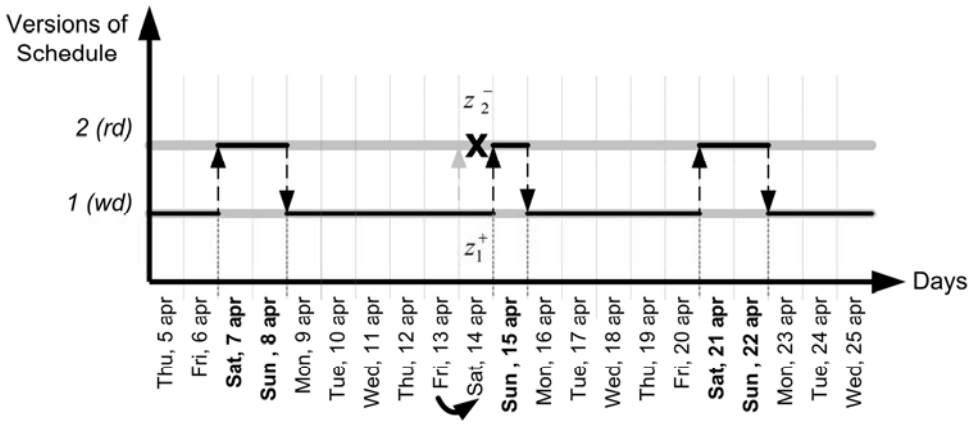

Fig. 6. *Schedule change of one day for to another*

The result $C(x) = True$ means that day x satisfies the periodicity characteristic of schedule C, while $C(x) = False$ means that for day x another version of the schedule is actual. Then, the set of dots of the time axis within the diapason $[t_s, t_e]$ corresponding to the periodicity attribute C is calculated in the following way:

$$(\forall x \in [t_s, t_e])C(x) \to x \in BExt([t_s, t_e], C). \tag{15}$$

For an example of calculating according to formulae (15), we consider the definition of dates in the diapason $[t_s, t_e]$, which are the workdays and the even days of the month. We may put down the solution of this task as follows:

$$(\forall x \in [t_s, t_e])\left[(x \text{ is } pd) \wedge (x \text{ is } wd)\right] \to x \in BExt([t_s, t_e], pd \wedge wd). \tag{16}$$

Generally, in calculating the expression $C(x)$, we have, at first, to work out all the elementary expressions $p(x)$ constituting it, and then to subject the received results (1 – *True* or 0 – *False*) to the

operations $\vee, \wedge$ and $\neg$ forming the expression C. Let us note that $p(x)$ means the expression built on the basis of the periodicity parameter p for day x.

For calculating the expressions $p(x)$ for the majority of possible elements of the set P, it is enough to know the coordinates of the reviewed dot on the time axis. For example, if we know the date, we can calculate out the day of the week and thus, to check it for the correspondence to the periodicity attributes *Mon, Tue, Wed, Thu, Fri, Sat* and *Sun*, and using the rule of correlation of the weekdays to defining the rest days, we can check it for the correspondence to the attributes wd and rd.

Still, there are such attributes for which check up it is not enough to know the coordinates of the reviewed dot, and we need to know periodicity attributes of the other days. The examples of such "complicated" periodicity are the first and the last days of the rest-days' group. To define them, we need to know the periodicity attributes of the adjacent days: the day before the reviewed date in case of the day of the rest-days' group and the day after the reviewed date for the last day of that group. Difficulty is added by the fact that for these adjacent days there might have been fixed changes or substitutions, and we need to take into account the periodicity attributes of all these changes.

For each element $p \in P$, there is a special rule of calculating $f_p(x)$. Examples of this calculations' rules are shown in table 2. In the given expressions we use the functions $WeekDay(x)$ and $DayOfMonth(x)$ returning for the date x, correspondingly, the day of the week and the number of the day in the month, as well as the function $Mod(a, 2)$ returning remainder after number *a* is divided by 2.

Table 2. Examples of rules for calculating belonging of the date *x* to the elements of the set *P*

Periodicity attributes p	**Rules of calculating the attribute** $f_p(x)$
Day of week: $p = dw \in \{Mon, Tue, Wed, Thu, Fri, Sat, Sun\}$	$f_{dw}(x) = WeekDay(x)$
Attribute type of day (of either a rest-day or a workday): $p = td \in \{wd, rd\}$	$f_{td}(x) = \begin{cases} wd, \text{ if } WeekDay(x) \in \{Mon, Tue, Wed, Thu, Fri\}; \\ rd, \quad \text{otherwise} \end{cases}$
Day's even and odd: $p = pn \in \{pd, nd\}$	$f_{pn}(x) = \begin{cases} pd, \text{ if } Mod(DayOfMonth(x)), 2) = 0; \\ nd, \quad \text{otherwise} \end{cases}$
The last holiday: $p = lh$	$f_{lh}(x) = \begin{cases} lh, \quad \text{if } WeekDay(x) \in \{Sat, Sun\} \wedge \\ \qquad WeekDay(x + 1\,day) \in \{Mon, Tue, Wed, Thu, Fri\}; \\ \varnothing, \quad \text{otherwise} \end{cases}$ where $x + 1\,day$ – the day following the day *x*
Every day: $p = ed$	$f_{ed}(x) = ed$

Example. The train plies on even days of the month except Tuesday. To work with the schedule of the given train, we need to define the dates from the diapason $[18.11.2007;\ \ 20.11.2007]$, which satisfy the periodicity characteristic "is an even day of the month and not Tuesday". For the given periodicity attribute we have $C = pd \wedge \neg Tue$, then the check up of the dates' correspondence to the given periodicity characteristic comes down to calculating the expression $C(x)$ of the type $C(x) = (f_{pn}(x) = pd) \wedge (f_{dw}(x) = \neg Tue)$:

$$C('18.11.2007') = [(f_{pn}('18.11.2007') = pd) \wedge (f_{dw}('18.11.2007') = \neg Tue)] = 1 \wedge 1 = True;$$

$$C('19.11.2007') = [(f_{pn}('19.11.2007') = pd) \wedge (f_{dw}('19.11.2007') = \neg Tue)] = 0 \wedge 1 = False;$$

$$C('20.11.2007') = [(f_{pn}('20.11.2007') = pd) \wedge (f_{dw}('20.11.2007') = \neg Tue)] = 1 \wedge 0 = False.$$

We cannot but notice that in the given time diapason only one date $x = '18.11.2007'$ satisfies the considered periodicity characteristic. The resulting is as follows:

$$x = '18.11.2007' \in BExt([18.11.2007;\ \ 20.11.2007], pd \wedge \neg Tue).$$

5. Examples of Calculating Temporal Elements

Let us consider an example of calculating temporal elements for different versions of the schedule for the train numbered n. The set of versions of train schedule with different periodicity is schematically presented in Fig.3 and Fig.5. As we can see in Fig. 3, the train has three schedules v_1, v_2 and v_3. Let us fix two versions of the schedule v_1 and v_2 with the corresponding periodicity attributes *wd* and *rd* on the period from November 1 to November 30, 2007; the time of fixation of these versions is October 10, 2007 correspondingly at 12:00 and 14:00, and the third version v_3 with the attribute *Tue,Thu* for the period from November 5 to 25, 2007 is fixed on November 2, 2007. Then we have: $t_1 = '01.11.07'$, $t_2 = '05.11.07'$, $t_3 = '25.11.07'$ and $t_4 = '30.11.07'$.

Fig. 5 shows the time period November 5 – 25, 2007, when three schedules v_1, v_2 and v_3 were in use; here: "1" on the axis "Days of week" corresponds to Monday November 5, 2007; "2" corresponds to Tuesday November 6, 2007, etc. According to formulae (2) we can describe each schedule version:

$$v_1 = \{\langle n, wd, '10.10.07\ \ 12:00', '01.11.07', '30.11.07\rangle, \langle s_1, '11:05'\rangle, \ldots, \langle s_{10}, '12:15'\rangle\};$$

$$v_2 = \{\langle n, rd, '10.10.07\ \ 14:00', '01.11.07', '30.11.07\rangle, \langle s_1, '11:00'\rangle, \ldots, \langle s_{10}, '12:12'\rangle\};$$

$$v_3 = \{\langle n, (Tue \vee Thu), '02.11.07\ \ 11:00', '05.11.07', '25.11.07\rangle, \langle s_1, '11:10'\rangle, \ldots, \langle s_{10}, '12:21'\rangle\}.$$

Note that in given examples we show only the first and the last stops of the train schedules to simplify the data presentation.

Since November19, 2007 was announced a rest-day in Latvia, on that day the trains were plying in conformity with the rest-day schedule. Therefore, in our case we have a non-empty set of exceptions for rest-days; in other words we have $EX(rd) = \{19.11\}$. Note that here and further on, we do not mention the year 2007 to simplify the data presentation. Calculation of the temporal element for the schedule version v_3, fixed after the first two versions, is made pretty simply according to formula (5), since we do not need the temporal elements of the other schedule versions here:

$$TE(C_3, t_{fix}^{(3)}) = TE(C_3) = BExt([t_s^{(3)}), t_e^{(3)}], C_3) \cup EX([t_s^{(3)}, t_e^{(3)}], C_3) - EX([t_s^{(3)}, t_e^{(3)}], \neg C_3) =$$
$$BExt([5.11, 29.11], Tue \vee Thu) \cup EX([5.11, 29.11], Tue \vee Thu) -$$
$$EX([5.11, 29.11], Mon \vee Wed \vee Fri \vee Sat \vee Sun \vee rd) =$$
$$\{6.11, 8.11, 13.11, 15.11, 20.11, 22.11\} \cup \varnothing - \varnothing = \{6.11, 8.11, 13.11, 15.11, 20.11, 22.11\}.$$

To calculate the temporal element of the schedule v_2, we need to know the temporal element of version v_3 – the only version of the schedule, which was fixed later:

$$TE(C_2, '10.10.07\ \ 14:00') = TE(C_2) - TE(C_3, t_{fix}^{(3)}) =$$
$$BExt([t_s^{(2)}, t_e^{(2)}], C_2) \cup EX([t_s^{(2)}, t_e^{(2)}], C_2) - EX([t_s^{(2)}, t_e^{(2)}], \neg C_2) - TE(C_3) =$$
$$BExt([01.11, 30.11], rd) \cup EX([01.11, 30.11], rd) - EX([01.11, 30.11], wd) - TE(C_3) =$$
$$\{3.11, 4.11, 10.11, 11.11, 17.11, 18.11, 24.11, 25.11\} \cup \{19.11\} - \varnothing -$$
$$\{6.11, 8.11, 13.11, 15.11, 20.11, 22.11\} = \{3.11, 4.11, 10.11, 11.11, 17.11, 18.11, 19.11, 24.11, 25.11\}.$$

The schedule v_1 is the earliest one, and to calculate its temporal element we need the temporal elements of the schedules v_2 and v_3:

$$TE(C_1, t_{fix}^{(1)}) = TE(C_1) - (TE(C_2, t_{fix}^{(2)}) \cup TE(C_3, t_{fix}^{(3)})) =$$
$$BExt([t_s^{(1)}, t_e^{(1)}], C_1) \cup EX([t_s^{(1)}, t_e^{(1)}], C_1) - EX([t_s^{(1)}, t_e^{(1)}], \neg C_1) - (TE(C_2, t_{fix}^{(2)}) \cup TE(C_3)) =$$
$$BExt([01.11, 30.11], wd) \cup EX([01.11, 30.11], wd) - EX([01.11, 30.11], rd) -$$
$$(TE(C_2, '01.04.2007\ \ 14:00') \cup TE(C_3)) =$$

$$\{01.11, 2.11,\ 5.11-9.11,\ 12.11-16.11,\ 19.11-23.11, 26.11-30.11\} \cup \varnothing - \{19.11\} -$$
$$(\{3.11, 4.11, 10.11, 11.11, 17.11, 18.11, 19.11, 24.11, 25.11\} \cup$$
$$\{6.11, 8.11, 13.11, 15.11, 20.11, 22.11, 27.11, 29.11\}) =$$
$$\{01.11, 2.11, 5.11, 7.11, 9.11, 12.11, 14.11, 16.11, 19.11, 21.11, 23.11, 26.11, 28.11, 30.11\}$$

6. Defining an Actual Schedule Version

Presence of the temporal element allows simplifying significantly many of the temporal functions, such as defining an actual object version, defining the dots on the valid time axis when a particular object version comes out of the "shadow zone".

Thus, to define the actual schedule version for the train with number n we suggest using function $ActVer()$ of the type:

$$ActVer(x, V^{(n)}) = \begin{cases} v_i & if\ x \in TE(C_i, t_{fix}^{(i)}); \\ \varnothing & otherwise \end{cases} \tag{17}$$

where x is the date which schedule you have to know.

The resulting of the function $ActVer(x, V^{(n)})$ is the element of the set $v_i \in V^{(n)}$ presenting the sought for actual version. If the train does not ply on the given date x, the function will give back the empty set. The guarantee of only one actual version in action is the condition of non-overlapping *TE* of all versions of the train schedule (see rule in the section 4.2.).

7. Conclusions

In the paper we have shown the possibility of using the theory of sets and the algebra of logics for the description of the railway schedule system. Realization of the mathematical model of railway schedule is performed in the environment of the temporal database, for which we have defined the periodicity parameters and temporal elements and formulated the rules of their calculation.

The suggested schedule's support system may serve not only as an informational schedule system but also as a part of the management system making it possible to model the railway schedule by automatically generating changes options in case of announcing an additional rest-day and as a part of an analytical system for analyzing and predicting passengers' flows.

Further step of the present research will be aimed at reaching efficient solutions for realizing the given system with the use of modern database technologies.

References

1. Kopytov E., Demidovs V., Petoukhova N. Method of Temporal Databases Design Using Relational Environment. In: *Scientific Proceedings of Riga Technical University – Computer Science: Applied Computer Systems*. Riga: Riga Technical University, 2002 Series 5, Vol. 13, pp. 236-246.
2. Kopytov E., Demidovs V., Petoukhova N. Use of Temporal Databases Principles in Information Systems of the Latvian Railway. In: *Proceedings of VII International Conference "TransBaltica 2002"*. Riga: RMS Forum, 2002. pp. 183-190. (In Russian)
3. Novikov F. Discrete Mathematics for Programmers/ Sankt.Pt. Piter, 2000, 304 p. (In Russian)
4. Terenziani P. Symbolic User-defined Periodicity in Temporal Relational Databases. *IEEE Transactions on Knowledge and Data Engineering.* Vol. 15, Issue 2, 2003, pp. 489-509.

SIMULATION OF BUSINESS PROCESS USING PLA FORMALISM

Henrikas Pranevicius, Vytautas Pilkauskas

Kaunas University of Technology, Business Informatics Department
Studentu 50, Kaunas LT-51368. Lithuania
E-mail: hepran@if.ktu.lt, vytpilk@ktu.lt

While creating business process simulations models, BPMN descriptions of those models are used. Simulation model creation means assigning parameters to BPMN model. BP-PLA graphic language is developed for creating simulation models. The summarizing aggregates simulating BPMN model elements were created while developing metamodel of that language. Two types of aggregates, such as: BP generator and business process behaviour imitator were distinguished in the implemented system version of simulation modelling. The named aggregates were formalized using PLA method and were implemented in PLA modelling system. BP simulation model PLA description is obtained from BPMN model in BP-PLA language. This description is automatically transformed into program code of simulation model. Provided method for simulation model creation is illustrated with oil-filled wagon outpour in to the oil terminal simulation model.

Keywords: *business process modelling, simulation, piece-linear aggregate, domain specific language*

1. Introduction

According to [1-2] a business process is a set of partially ordered activities that are aimed at reaching a goal. The main concepts of the business process are:

- *Goal* or objective of the process. It means that after the process has completed it is possible to know if the process has achieved its goal.
- *Activity or task*. It changes the state of the process in some way.
- *Time* – an axis which the partial order refers to. The time can be regarded as being absolute or relative, discrete or continuous, internal or external.
- *State*. It is a characteristic of the process. It can be final (the goal has been reached) or intermediate (the goal has not been reached yet).
- *Event*. A moment of time at which the state of the process changes.
- *Process type*. Processes that are similar by their goals and activities are united in such types.

These concepts are similar to concepts that are used to describe a dynamic system which changes states on the way to reaching its goal.

Simulation models can provide the most accurate and insightful means to analyze and predict the performance of business processes [3]. A simulation is an imitation of a system. A simulation of a department, responsible for paying accounts, would imitate the behaviour of such department and portray its dynamics. In the past few years, several new software tools have been developed specifically for modelling business processes. Most of these tools define business processes using graphical symbols or objects, with individual process activities depicted as a series of boxes and arrows. Special characteristics of each process or activity may then be attached as attributes of the process. Business process simulation software tools can be divided into three major categories [3]:

- Flow Diagram Based simulation tools;
- System Dynamics Based simulation tools;
- Discrete Event Based simulation tools.

The most capable and powerful tools for business process simulation are the Discrete Event driven simulation products. These tools provide modelling of entity flows with animation capabilities which allow user to see how objects are routed through the system. Some of these tools even provide object-oriented and hierarchical modelling which simplifies development of large business process models.

Most of the existing semantic models, languages and logics dedicated for describing and reasoning about time-based systems, implicitly view execution as an alternating sequence of instantaneous 'discrete' actions and 'continuous' phases during which time advances. To each system described in any of these formalisms, one can associate a transition system or automaton consisting of a set of states, a set of initial states, a set of discrete actions, a set of discrete steps $s' \xrightarrow{a} s$, and a set of time-passage steps $s' \xrightarrow{d} s$. It is asserted that from state s' the system can instantaneously move to state s via the occurrence of the discrete action a'. Furthermore, from state s' the system can move to state s during a positive amount of time d in which no discrete action occurs. These transition systems provide a very abstract view of the original system behaviour, in which many aspects, such as the number of parallel components, the communication between these components, the way in which a system evolves during the continuous phases, etc., are no longer represented.

In this paper Piece-Linear Aggregate formalism (PLA) [4] will be used for creating dynamical models of business process. Motivation for using formal methods is:

- Permit an unambiguous formal description of a system being analyzed;

- Model properties can be analyzed using mathematical proof techniques;
- Formal description approach is a theoretical background for developing computerized formal specifications' analysis software tools (validation, verification, simulation).

PLA is a special case of automaton models. The state of the aggregate can change in two cases only: when an input signal arrives at the aggregate or when a continuous component acquires a definite value. The Piece-Linear Aggregates are based on Piece-Linear Markoff representation.

The scheme for creating a model for simulating business process (BP) is given in Figure 1.

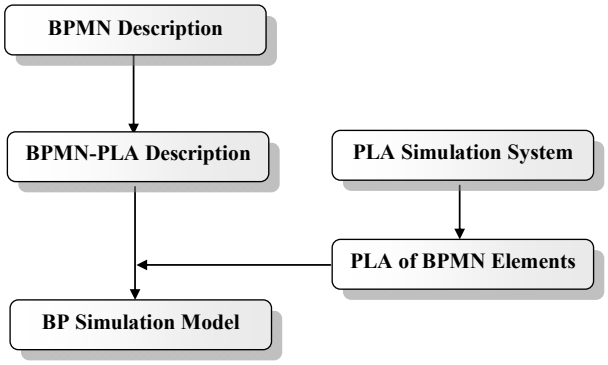

Fig. 1. *The scheme for creating BP simulation model*

BP-PLA graphic language was created for the expansion of BPMN model. Language was developed by deploying Microsoft Tools for Domain Specific Languages [6].

While creating metamodel of the graphic language, the summarizing aggregates, simulating BPMN model elements, were created. In the implemented version of simulations system two types of aggregates: *Business Process Generator* and *Business Process Behaviour Imitator* were distinguished. The first aggregate simulates BPMN *Start Event* element. The second aggregate simulates the following BPMN elements: *Activity, Sequence flow, Message flow, AND Gateway, Exclusive Gateway* and *End Event*.

The named aggregates were formalized using PLA method and implemented in PLA modelling system. PLA description of BP simulation model was obtained by describing BPMN model in BP-PLA language. This description was automatically transformed into program code of the simulation model. The BP simulation model was obtained by applying PLA simulation modelling framework.

The rest of the paper is structured as follows: Section 2 describes PLA formalization approach, Section 3 presents object oriented model of PLA that had been used for developing Business Process Simulation Modelling System. Section 4 presents how the simulation model is obtained by deploying our created simulation system BP-PLA and using BP BPMN descriptions. Section 5 presents an example of methodic for creating simulation model of business processes in transhipping oil terminal.

2. PLA Simulation System

The aggregate approach for a system specification is applied by representing a system as a set of interacting Piece-Linear Aggregates (PLA) [4]. The PLA is understood as an object defined by a set of states Z, input signals X and output signals Y. The aggregate function is defined in a set of moments in time $t \in T$. The states $z \in Z$, the input signals $x \in X$, and the output signals $y \in Y$ are considered to be time functions**.** Along with these sets, transition H and output G operators are also used.

Simulation modelling system was implemented in Microsoft .NET 2.0 environment in C# program language [5]. The system has a library composed of basic classes for aggregate simulation.

For the implementation of aggregate connection scheme there were three dedicated classes used (Fig. 2):

SimulationModel – aggregate simulation model basic class;

Aggregete – aggregate basic class;

OutputPoint – aggregate interaction point class.

The aggregate connection in the aggregate system is implemented by the *SimulationModel* class which contains an aggregate list – *modelAggregates*. In this list aggregates belonging to the *Aggregate* are kept. Aggregation objects and *SimulationModel* classes are created during the creation of aggregate connection, which is formed in the *AddAggregate* method. These are then placed into the *modelAggregates* list. After that, the aggregate objects are connected to the closed system by deploying *OutputPoint* class (i.e. aggregate interaction point). The object method inside this class *ConnectTo()* is connected to the closed system using the connection channels. The aggregate interaction while communicating through aggregate connection channels is implemented by the interaction point object method *Output()***.**

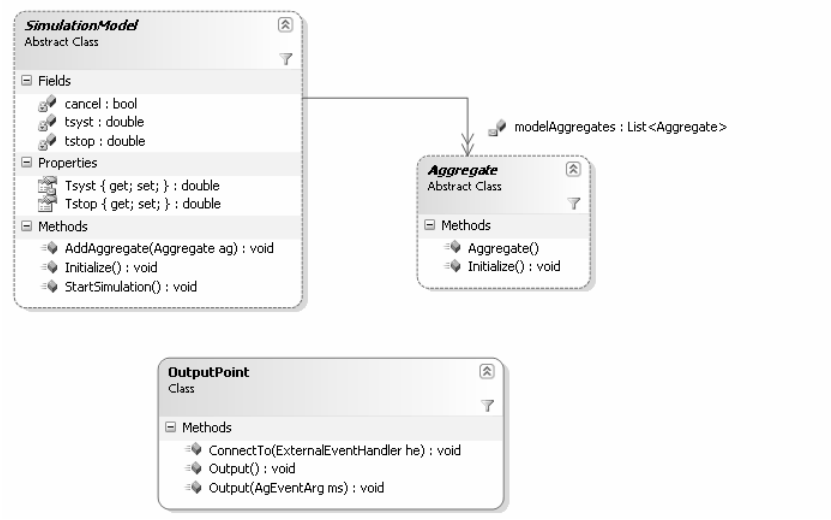

Fig. 2. *The diagram of aggregate connection scheme implementation classes*

SimulationModel class method *initialize()* is created for the aggregate system initialization (i.e. initial aggregate system state) setting. The implementation of this method calls *initialize()* methods of each aggregate object in the *modelAggregates* list.

The following classes for the aggregate external event generation and it's processing in the library are described (Fig. 3):

ExternalEvent – external event class;

ExternalEventHandler – delegate of aggregate external event processing method;

AgEventArg – the base class of message transfer between aggregates.

The generation of aggregate external events is initiated by the aggregate message transformer in the method *Send()* which moves events to the appropriate interaction point object. The outgoing message is the object within *AgEventArg*. The structure of this message in the interaction point object method *output()* is placed into the newly generated outer event *ExternalEvent* type object. Within this object there are as many external events generated as there are saved references in interaction point object attribute *targetEventHandlers*. The references to the delegate *ExternalEvenHandler* of the appropriate outer event processing method are saved in this attribute. All generated external events are placed and saved in the queue *SimulationMoel.externalEventQueue* till they are selecting for processing with *NextExternalEvent()* method from *SimulationModel* class.

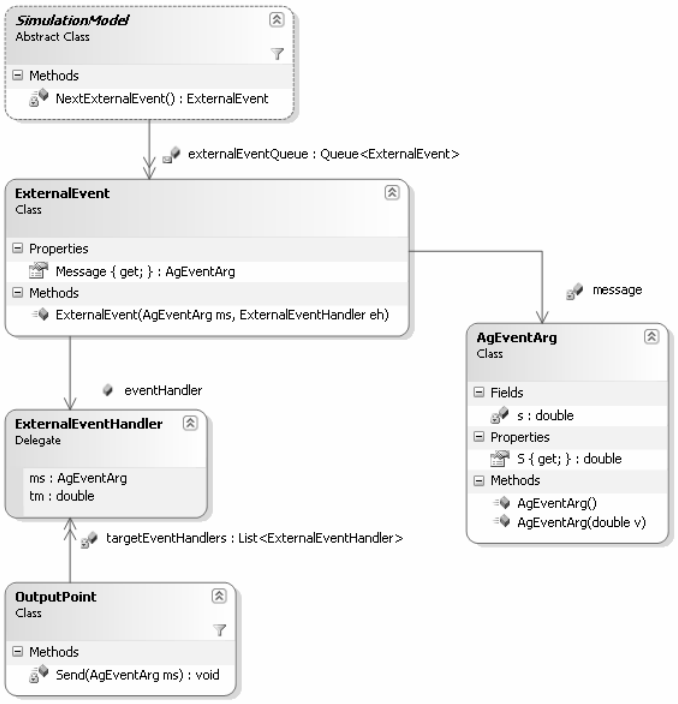

Fig. 3. *The diagram of external events implementation classes*

The following classes for the generating and processing aggregate's internal events are distinguished (Fig. 3):

InternalEvent – internal events class;

ContiousCoordinate – abstract continuous coordinates class;

InternalEventHandler – delegate of aggregate internal event processing method;

ControlSum – class of control sum;

ControlSequence – class of control sequence.

Aggregate internal event generation is initiated by aggregate controlling sum *ControlSum* method *CreateInternalEvent(w)*. The time moment of the expected internal event is set by this method's parameter *w*. One of the further mentioned methods can be used for setting parameter *w* of control sequence *ControlSequence* class object:

Next() – generates random number equally distributed in the interval (*0, 1*);

Next(double a, double b) **–** generates random number equally distributed in the interval (*a, b*);

Next(double b) – generates random number equally distributed in the interval (*0, b*);

NextExp(duoble m) – generates random number equally distributed according the exponential law with the mean *m*.

Generated internal event is put into the sorted list of internal events *internalEventQuque***.** The sorting is made according the internal event attribute *W*. Internal event for the processing is selected by *SimulationModel* class method *NextInternalEvent()*.

3. BP-PLA Simulation Systems

While creating metamodel of the grapic language, the summarizing aggregates, simulating BPMN model elements, were created. In the implemented version of simulations system two types of aggregates: *Business Process Generator* and *Business Process Behaviour Imitator* were distinguished. The first aggregate simulates BPMN *Start Event* element. The second aggregate simulates the following BPMN elements: *Activity, Sequence flow, Message flow, AND Gateway, Exclusive Gateway* and *End Event*.

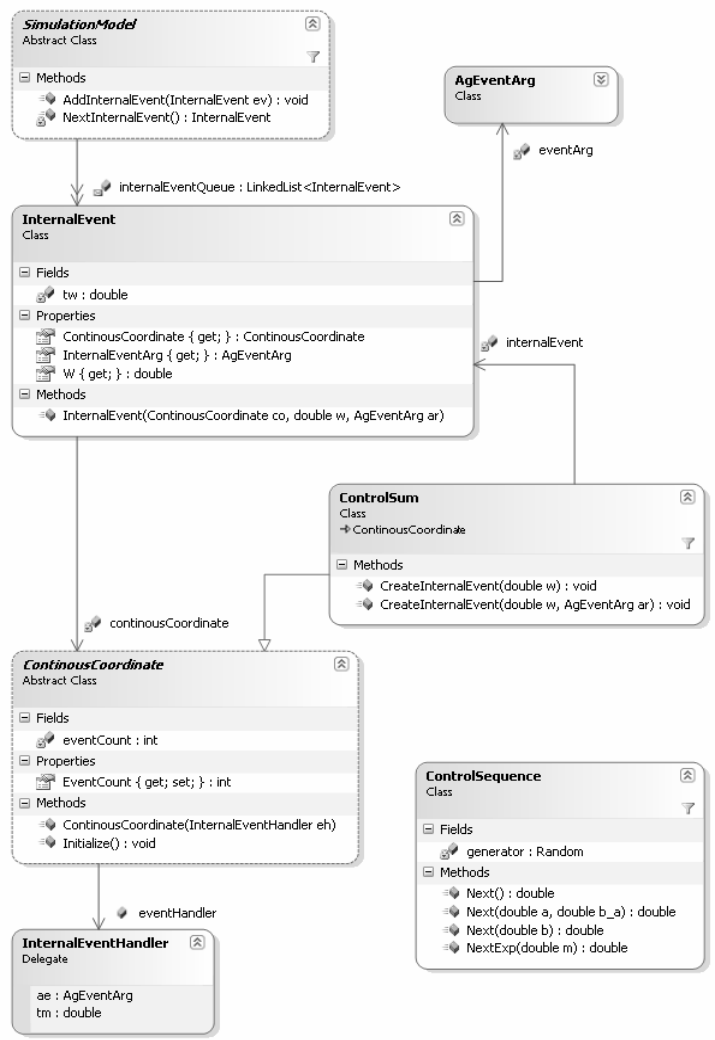

Fig. 4. *The diagram of internal event implementation class*

3.1. Aggregate for BP generation

Aggregate generator implements BPMN start event, i.e. creates a new BP instance that is provided by the number of i-th queue (unique identifier). This number identifies appropriate BP process instance in the simulation model (BPi). The BPi performance trajectory is defined by (Fig. 5):

$$PB_i = \left\langle a_1^i, a_2^i, a_3^i, \ldots a_{N_i}^i \right\rangle,$$

where: a_j^i is the identification of *j*-th activity in BP_i trajectory; N_i is the number of activities in BP_i trajectory.

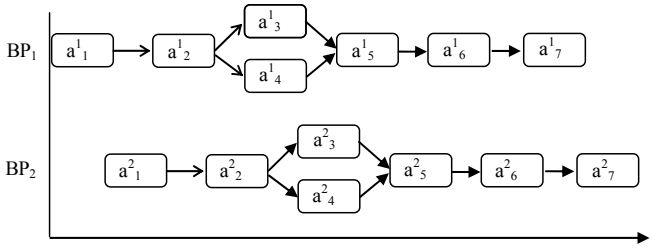

Fig. 5. *BP performance trajectory*

The following performance durations are defined for each activity element of that performance trajectory:

$$PB_i^{\tau} = \left\langle \tau_1^i, \tau_2^i, \tau_3^i, \ldots \tau_{N_i}^i \right\rangle,$$

where τ_j^i is the duration of a_j^i activity.

BP_i process is defined by two sequences

$$BP_i = \left\langle BP_i^a, BP_i^{\tau} \right\rangle.$$

Characterized sequences are generated according to the schedule formed in advance or according to the distribution law that defines the time moments when new processes appear.

Generated sequences are output signals of Generator aggregate ($y = BP_i$).

3.2. Aggregate for imitation of BP behaviour

The structural scheme for business process behaviour aggregate is given in Figure 6.

The presented scheme pictures n-channel service system that is composed of:

- Service devices R_i, $\mathrm{i} = \overline{1,n}$;
- Priority queues Q_j, $j = \overline{1,m}$;
- Buffer groups $B_j = \{B_{j1}, B_{j2}, B_{jk_j}\}$, $j = \overline{1,m}$.

Service devices R_i are used for BPMN activity performance modelling. Activity can be performed in any free service device. If all the devices are busy, BP is put into queue. If the service device is busy, relative priorities can be assigned to queued activities. The priority queues were introduced for implementation of priority service. Buffer groups are used for the simulation of BPMN element's *AND Gateway.*

3.3. BP-PLA graphic language

BP-PLA graphic language was created for describing BPMN model expansion. Microsoft Tools for Domain Specific Languages (DSL) were used for the composition of this language [6]. Using DSL, users can design graphical languages and generate code and other text output from the domain-specific languages. DSL tools consist of:

- A project wizard creating a fully configured solution. There user defines a domain model;
- A graphical designer for defining and editing domain models;
- Designer definitions in XML. The code for implementing designers is generated from these definitions so that one can define a graphical designer hosted in Microsoft Visual Studio without manually writing any code.
- A set of code generators that take a domain model definition and a designer definition as input and produce code implementing both of the components as output. The code generators also validate the domain model and the designer definition.

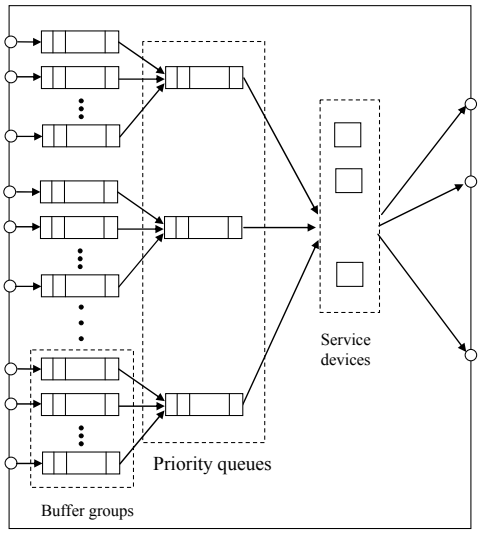

Fig. 6. *The structural scheme of BP behaviour aggregate*

BP-PLA graphic language domain model definition is given below. BPPLA model consists of two types of aggregates (Fig. 7):

- *AgrStart* is used to simulate BPMN *Start Event* element;
- *AgrFlow* is used to simulate BPMN flow elements.

Aggregate AgrStart has an output OutPort, while AgrFlow has both input and output points InPort.

Aggregate's input and output points are connected with *Connection* channels, which are used for message transmission between aggregates (Fig. 8). Message received via *InPort* is transmitted to BPMN element *Activity*. This BPMN element can transmit the message further to another aggregate via *OutPort.*

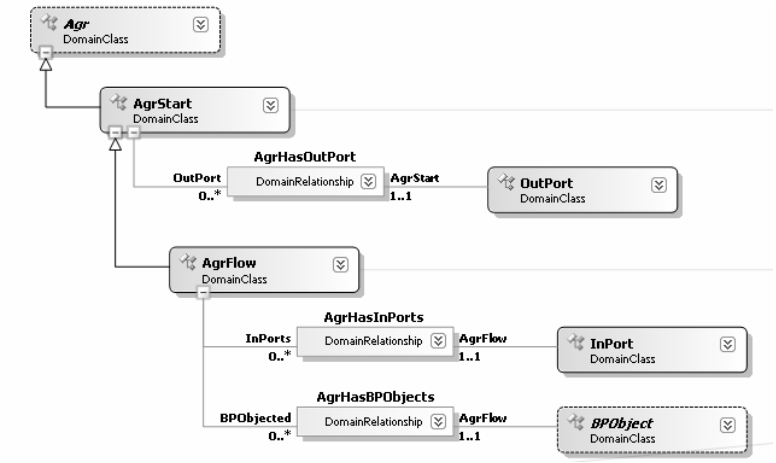

Fig. 7. *Diagram of BP-PLA graphical language for aggregates definition*

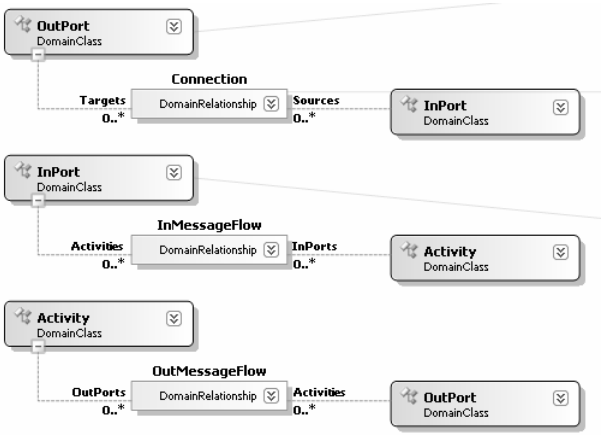

Fig. 8. *Aggregate and Activity interconnection diagram*

A diagram of BPMN elements (*EndEvent, Actuvity, Gateway*) is presented in Figure 9.

4. Simulation Model of Oil Terminal

Oil products are transported to the oil terminal by trains. When a train arrives, the representative of the railway gives the load documents to the terminal operator. After that the data of load arrival and delivery notes are entered into informational system of the terminal. At the same time customs procedures are

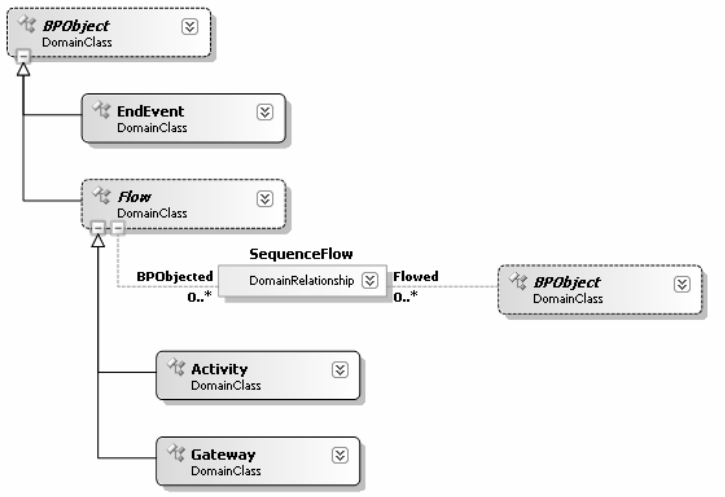

***Fig. 9**. Diagram of BPMN elements*

being completed. The worker of the strategic management department groups the delivered wagons into groups with no more than 32 wagons in each. In this way all the wagons are emptied in two turns. After the message about assigned groups reaches railway station service, wagons are delivered to the overhead road for the outpour. The needed tanks are prepared before outpour, their level is fixed, and the route for product pumping by the pipes is set, needed bolts are opened, the outpour devices in the overhead are connected, the steam is passed, and the pumps are switched on. The wagons are checked, steam pass is switched off, bolts are closed, transhipment devices are disconnected, and the van dispatch documents are formalized. The railway station service takes the wagons from the terminal operators and draws the documents for further wagon transportation.

A business process BPMN diagram of oil pouring out of the wagons into reservoirs is presented in Figure 10.

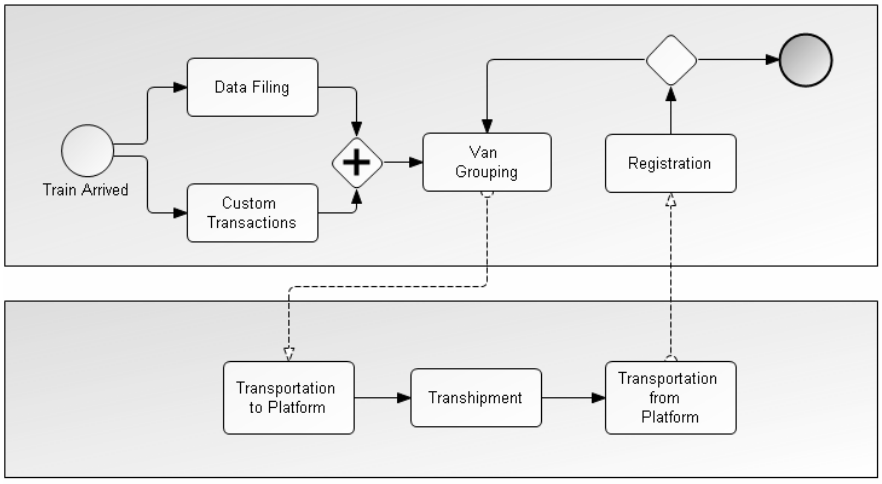

***Fig. 10.** BPMN diagram for oil outpour from wagon to terminal tank*

While creating the simulation model, the following oil terminal resource requirements had to be evaluated:

- Two working places for the document processing in the strategic management department exist: arrival data filing (aggregate *ManDep1*), and wagon grouping into groups (aggregate *ManDep2*);
- One working place for document processing in customs is foreseen (aggregate *Custom*);
- Two platforms are used for transhipment of oil in terminal (aggregate *Platform*);

- One locomotive for wagon transportation between railway station and transhipment platforms is used (aggregate *Locomotive*).

A BP-PLA model diagram was created using these constraints (Fig. 11).

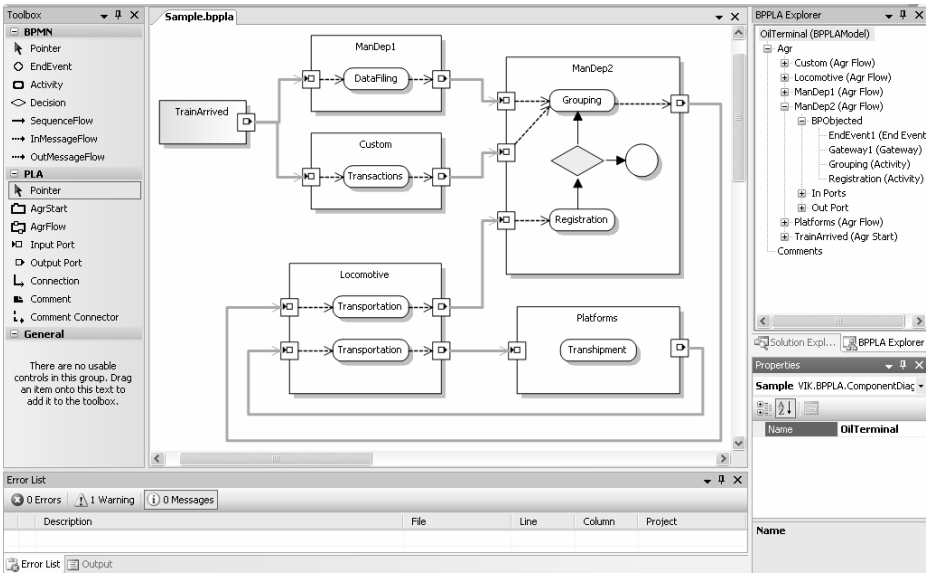

Fig. 11. *BP-PLA model diagram*

Using the diagram a software code of the simulation model was generated.

Simulation results An income from wagon unloading, i.e., oil pouring out from wagons into terminal reservoirs, was calculated during simulation. The income depends on unloaded amount of oil. Income dependencies of pier loading coefficient (*K*) in different environment temperature conditions (*T*) are presented in Figure 12.

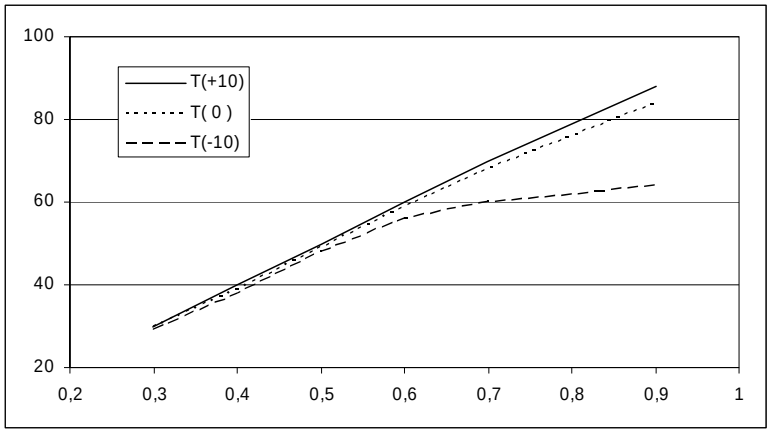

Fig. 12. *Simulation results – income*

The income dependencies are presented using relative percentage values. It is considered that 100 percent income is at 20 Celsius environment temperature (there is no need to warm wagons before pouring out) and while pouring out piers are fully loaded, i.e., when K=1.

Dependencies of formed oil pouring expenses from pier load coefficient (K) in the different environment temperature conditions (T) are presented in Fig. 13. Pouring expenses depend on the amount of the poured oil and expenditures for wagon warming till the required temperature before the pouring begins. Warming expenses arise when oil in wagons is getting cold while waiting for the pouring to begin. These expenses are presented using the same relative values as for income in Fig. 12.

Dependencies of the difference between received income and pouring expenses on pier loading coefficient (K) in the different environment temperature conditions (T) are presented in Fig. 14. This difference is presented using the same relative values as for income in Figure 12.

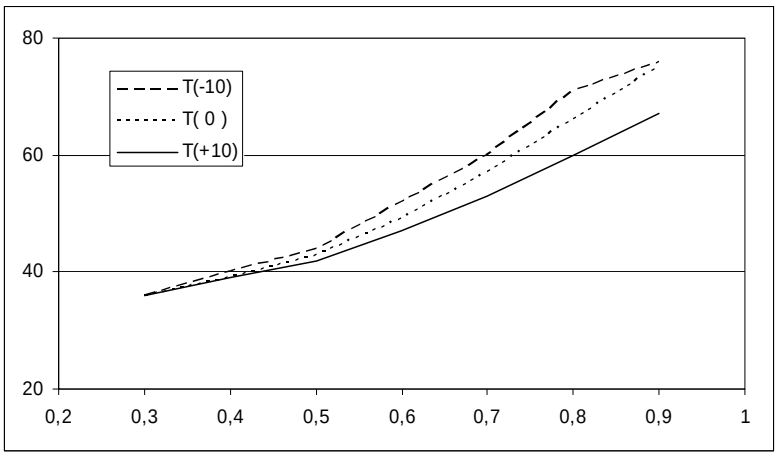

Fig. 13. *Simulation results – expense*

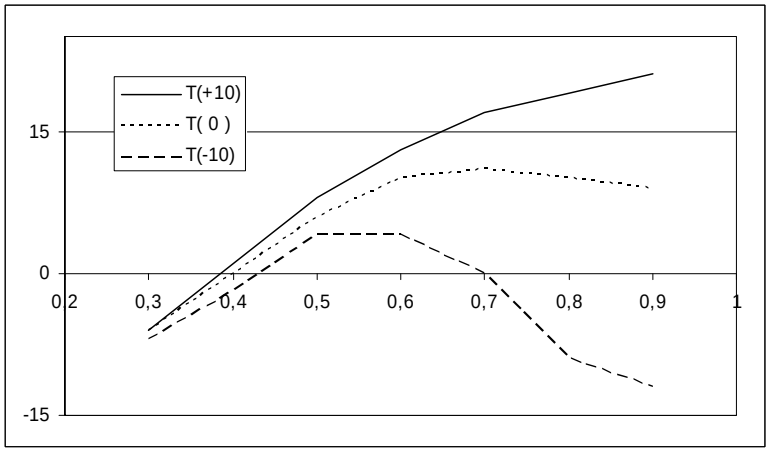

Fig. 14. *Simulation results – difference*

5. Conclusions

The BP-PLA language created by authors for design of simulation models is presented in the paper. The language uses PLA modelling concepts. It permits to expend business process descriptions in BPMN. This extension permits to automate the generation of simulation model programs.

References

1. Kirikova, M., Makna, J. Renaissance of Business Process Modelling. In: *Proceedings of 13th International Conference on Information Systems Development – Advances in Theory, Practice, and Education (ISD 2004)* Vilnius, 2004, pp. 403-414.
2. F-R.Lin, M-Ch. Yang, and Y-H. Pai. A generic Structure for business process modelling. *Business Process Management Journal* Vol. 8(1), 2002, pp. 19-41.
3. Hlupic, V., Vreede G-J., Orsoni A. Modelling and Simulation Techniques for Business Process Analysis and Re-Engineering, *International Journal of Simulation Systems Science and Technology*, Vol. 7 (4-5), 2006, pp. 1-8.
4. Pranevicius H. Aggregate Approach for specification, Validation, Simulation and Implementation of Computer Network Protocols. *Lectures Notes in Computer Science* No 502, Springer, 1992, pp. 433–477.
5. Pranevicius, H., Pilkauskas V., Guginis G. Creating Simulation Models Specified by PLA Using UML. In: *International Conference on Operational Research: Simulation and Optimisation in business and Industry.* Kaunas: Technologija, 2006, pp. 87–92.
6. Greenfield J., Short K., Cook S., Kent S.: *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. Wiley Publishing, 2004, 215 p.

MODELLING AND ANALYSIS OF BUSINESS RULES USING KNOWLEDGE BASED AND FORMAL APPROACHES

Henrikas Pranevicius, Germanas Budnikas

Business Informatics Department, Kaunas University of Technology
Studentu 56-301, Kaunas, Lithuania
Phone: +370 37 451654
E-mail: {henrikas.pranevicius, germanas.budnikas} @ktu.lt

The paper presents an approach that applies knowledge based and piece-linear aggregate (PLA) formal approaches for modelling and analysis business rules. In our technique, business rules are represented by production rules using concepts of PLA model. A knowledge base of the business rules is analysed by checking its consistency (static properties) and dynamic constraints (dynamic properties). The analysis is performed by applying decision table verification method and reachable state validation method and supporting tools – PROLOGA and CLIPS. The proposed approach is illustrated by an example.

Keywords: *business rules, PLA, knowledge base, decision table, consistency and dynamic constraints*

1. Introduction

Business rules are significant for enterprises because they define how they are doing business. They are also popular in the information system community as they are able to make applications flexible. One of the most important parts in the development of information systems is problem representation and analysis of domain descriptions, which, in our case, are expressed in a form of business rules. Possible inconsistencies like redundancies, contradictions or missing rules as well as dynamic failures like inability to reach a defined goal should be corrected as early as possible to minimise cost of the development. Checking for consistency as well as dynamic constraints is emphasised in formal methods and formal specifications. The most popular formal specification languages being used for description of the systems are SDL, Lotos, Estelle, PLA [1], [2], [3]. Checking of business rules consistency is highly important. Examples of related works could be [4], [5], [6], [7], [8]. The use of models (especially formal ones) while representing and analysing business rules is propagated in [9], [10] as it simplifies the rule formalisation and ensures higher clarity and consistency of the rule descriptions.

PLA is a formal state-based model. It is successfully used for formal specification, validation and simulation of complex system including computer network protocols, transport systems, medical applications, etc. for several decades. Use of PLA model in representing and analysing business rules should give us an advantage of a use of formal model as well as accompanying analysis techniques. In this paper we will explore the possibilities to use PLA and related techniques for business rules.

Business experts find difficult to understand formal specification languages. Many authors (e.g. Bruynooghe *et al*. [11]) believe that the declarative style of description that is used in knowledge bases is more understandable and acceptable than the procedural style (the latter is used in PLA formal specification languages). This is because a problem is described at the knowledge level "at which the knowledge engineer specifies expertise" [12]. Representation of knowledge used in knowledge-based systems (KBS) is close to the way of thinking by a human. Conditional and sequential parts of the most business rules correspond to a structure of a production rule representation. In this way, representation of business rules by production rules is natural.

State structure, conditions for state changes due to incoming requests and internal events are strictly defined in the PLA model. Adding the PLA model to the representation process assigns extra value in terms of easier to represent a structure of business processes expressed using business rules, their interaction; easier to perform checking.

Our approach for representing business rules is based on production rule representation and PLA model elements. This approach is explained in section 2. Business rules consistency is expressed using general properties — non-redundancy, non-conflictness, non-deficiency. The consistency of business rules is checked by transforming PLA KB productions to decision tables (DT) and applying tabular static verification technique that is implemented in Prologa system. The consistency checking technique is described in section 3. Dynamic constraints of business rules are expressed using general dynamic properties — absence of static deadlocks, final state reachability, boundedness, and completeness. The constraints are checked using expert system that is built by combining PLA KB with knowledge base of dynamic properties and validation method. The technique is presented in section 4. Section 5th illustrates the proposed technique with the example of business rules describing goods ordering regulations used by a firm manager. Relation of the proposed approach to other works is discussed in section 6. Conclusion sums up the paper.

2. Business Rules Representation Using Concepts of PLA Model

According to the review made in [5] classification categories of BR include the following: rules describing situations (e.g. state changes due to business events), constraints, facts asserting structure, terms corresponding to actors, events, parameter values, actions, etc. These concepts and rules already present in the state-based formal piece linear aggregate PLA model [3]. The model defines notions that can be successfully applied for business rule representation and analysis. Input and output requests, their structure, internal and external (business) events along with their cause, activities, computational parameters, state structure and rules describing how state is changed, actors and their interconnection are all defined in the formal PLA model. As was stated early, representation of business rules in a declarative way – by using productions is natural and will be used in our approach.

Therefore, we propose for each rule category to use an appropriate rule template, which can be seen as a sentence pattern that tells how to describe the rules that belong to a particular category. Further we present examples of business rule categories and how they are represented by rule templates expressed in a form of knowledge base based on PLA model (PLA KB) predicates and productions.

Business rule category	*PLA KB predicate or production*
Incoming request	IncomingRequest (a_i, x_j^1, …, $x_j^{c_{ic}^i}$), where a_i – symbolic name of the ith actor; x_j^1,…, $x_j^{c_{ic}^j}$ – components of the request.
Actor's state	State (a_i, w_i^1, …, $w_i^{c_{cc}^i}$, d_i^1, …, $d_i^{c_{dc}^i}$), where w_i^1,..., $w_i^{c_{cc}^i}$ and d_i^1,..., $d_i^{c_{dc}^i}$ are actions and computation parameters describing the state.
State change and output of the request after occurrence of an external event	IF IncomingRequest (a_i, x_j^1, …, $x_j^{c_{ic}^i}$) and State (a_i, w_i^1, …, $w_i^{c_{cc}^i}$, d_i^1, …, $d_i^{c_{dc}^i}$) and Aux (a_i, w_i^1, …, $w_i^{c_{cc}^i}$, d_i^1, …, $d_i^{c_{dc}^i}$) THEN State (a_i, w_i^1*, …, $w_i^{c_{cc}^i}$*, d_i^1*, …, $d_i^{c_{dc}^i}$*) and OutcomingRequest (a_i, y_j^1, …, $y_j^{c_{oc}^j}$) where predicate Aux describes auxiliary conditions that check state values.

The representation using PLA KB is performed by putting knowledge of business rules to corresponding elements of PLA KB which form and structure are defined with respect to formal PLA model. Such a way of representing the business rules by putting their knowledge to the constructs of the defined structure is suggested in [9], [10], as it simplifies the rule formalisation and ensures higher clarity and consistency of the rule descriptions.

After creation of the PLA KB, it is checked for consistency and dynamic failures by analysing its general static and dynamic properties correspondingly.

3. Checking of Business Rules Consistency

Business rules consistency means that the business rules are expressed in the right way. Checking this property is known as verification [4]. Consistency of business rules usually is expressed in terms of non-redundancy, non-conflictness, non-deficiency [5]. These constraints by their nature correspond to the ones that are analysed while checking static properties (i.e. performing static verification) of rule bases.

As it was mentioned earlier, business rules in our approach are represented by production rules of PLA KB. Since most of inconsistencies in rule-based systems can be solved using decision tables (DT) [13] and the tabular verification technique is computerised in Prologa system [14], our approach employs these facilities. Therefore, in order to perform checking of consistency of business rules represented in PLA KB, its production rules have to be transformed to Prologa DTs. As stated in [13], a DT is equivalent to a set of production rules. Consistency constraints for DTs have direct correspondences to the ones, defined for rules, which make the PLA KB.

In our technique, consistency checking is performed in Prologa system. The system uses single hit DT representation. In a single hit table each possible combination of a condition can be found in exactly one and only one column.

The transformation to Prologa DTs is specific with respect to the PLA model and employs some of its concepts that have been mentioned in the beginning of this section. Production rules are transformed to the DTs of certain groups thus enabling to fully exploit advantages of tabular representation to perform consistency checking.

Static verification technique for analysis of PLA KB is based on [13], [14] works. Further we present a portion of this technique. A *subsumed column pair* in a single hit DT is represented by single column that corresponds to several PLA KB productions. Thus, the subsumed column pair anomaly is detected if several PLA KB productions correspond to the same DT column. Full description of the PLA KB static verification technique as well as transformation steps of PLA KB constructs to single hit DTs are presented in [15], [16].

4. Checking Dynamic Constraints of Business Rules

The dynamic constraints (or properties) are those characteristics of a rule-based system that can be evaluated only by examining how the system operates at a run time. The most common techniques of validation and verification that have been developed for use on KBS are identified in [17]. A detailed review of specific methods and supporting tools can be found in [18].

Functional validation of the PLA KB is carried out using the expert system in CLIPS (C Language Integrated Production System). CLIPS is a tool for productive development and delivery of expert systems [19]. The expert system is built by combining statically verified PLA KB with a KB of dynamic properties and validation method (KB DPVM) [16].

Inference in the expert system for functional validation is performed using forward-chaining strategy joined with a reachable states method [15]. The idea behind this method is a construction of a corresponding graph of global states of the model (i.e. the investigated business process) under analysis. Construction is started with the predefined initial global state. All possible transitions from the given state are analysed. Repeatedly generated (according the model functioning) states that have been already analysed are excluded from the analysis. The reachable states graph is constructed till the final state is reached or some of the validated dynamic properties are not satisfied. While making the inference, rules of PLA KB, which describe both business processes and dynamic properties, are activated and executed. The scheme below illustrates how functional validation in the expert system is performed.

The defined KB DPVM, which is implemented in CLIPS, may be used for various kinds of applications. The instances of dynamic properties are the absence of static deadlocks, final state reachability, boundedness of countable state parameters, and completeness. Validation method, in which all reachable states are analysed, is implemented in the KB DPVM. In a view of analysis of the execution paths, this method is similar to functional validation method suggested by Preece *et al*. [20] that analyses the sequences of rules that must fire to achieve a goal.

When performing the join of PLA KB with KB DPVM, the needed adaptation changes are minor—they are an adaptation of description of dynamic constraints for a specific business process. For instance, in order to check the boundedness property, individual bounds have to be defined. Having combined the KB DPVM with the PLA KB, the expert system (ES) in CLIPS is built. The ES uses CLIPS inference engine that is based on the forward chaining strategy. If the validated dynamic constraints are violated, the expert system generates a corresponding report and a designer corrects the KB accordingly.

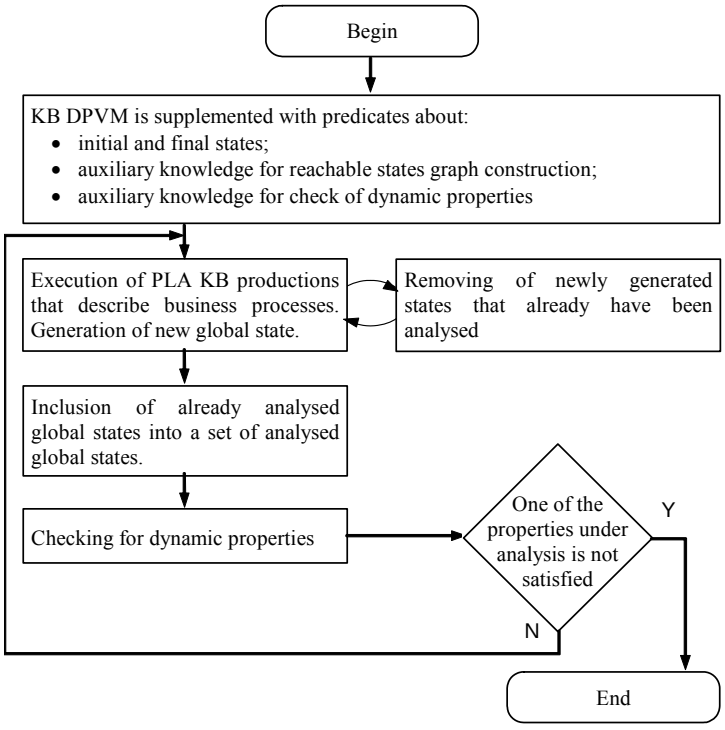

Fig. 1. *Schematic illustration of the analysis of dynamic properties in the validation expert system*

5. Illustration of the Proposed Technique

For illustration of the proposed technique we employ business rules describing automobile spare parts ordering regulations used by a firm manager. The rules are the following. A customer orders an item (or items). At the first, the manager searches for it in a local store. If the search fails, the item is ordered from a warehouse; otherwise it is ordered from the local store. If an amount of the requested items is greater than the local store has, partial ordering (from the local store and then from the warehouse) is requested. Local store may have up to 100 items. A fragment of the business rules represented by PLA KB is given in Table 1.

Table 1. PLA KB fragment of goods ordering regulations

Business rule	**PLA KB predicates and rules**
State of the aggregate *LocalStore* is characterised by predicate State:	
Item search in a local store, order from a warehouse, order from a local store, partial order from the warehouse and local store, search result and amount of the requested items.	State(*LocalStore, search, order_ws, order_ls, porder_ws, porder_ls, search_res, item_amount*)
For the briefness, afore-defined predicate will be denoted as State_C.	
A customer orders an item (or items). At first, the manager searches for it in a local store.	IF IncomingReqest (*LocalStore,item_number, amount*) and State_C THEN *search** = True and State_C
If the search fails, the item is supplied from a warehouse.	IF EndOfOperation (*LocalStore,Search*) and State_C and *search_res* = False THEN *order_ws** = True and State_C
If the search fails, …; otherwise it is supplied from the local store. If an amount of the requested items is greater than the local store has, partial order (from the local store and then from the warehouse) is requested.	IF EndOfOperation (*LocalStore,Search*) and State_C and *search_res* = True and *item_amount* > 100 THEN *porder_ls** = True and State_C IF EndOfOperation (*LocalStore,POrderFromLocalStore*) and State_C THEN *porder_ws** = True and State_C

Two PLA KB productions describing *end of search in a local store* are represented by a decision table (see Table 2).

Table 2. Decision table describing end of search in a local store

1. EndOfOperation (*LocalStore,Search*)	True				False
2. ^State_C	True			False	-
3. *search_res*	True		False	-	-
4. *item_amount*	≤ 100	> 100	.	-	-
1. *porder_ws** = True	.	.	x	.	.
2. *porder_ls** = True	.	x	.	.	.
3. State_C	.	x	x	.	.
	1	2	3	4	5

The decision table shows that no rules corresponding to the first row. Thus, a deficiency case – missed rule presents in the KB. Such verification report presents Prologa system. Therefore, the KB has to be supplemented with a corresponding rule.

Next we present a production rule of KB of dynamic properties and validation method that checks dynamic constraint — absence of deadlocks:

IF State(*parrent_state, current_state, LocalStore, search, order_ws, order_ls, porder_ws, porder_ls, search_res, item_amount*)
AND *search* = Inactive
AND *order_ws* = Inactive
AND *order_ls* = Inactive
AND *porder_ws* = Inactive
AND *porder_ls* = Inactive
THEN
Validation_Result ('Deadlock presents in state ' & *current_state* & ' which is originated from' & *parrent_state*).

6. Related Works

General structures of the predicates and rules for KBs of application and validation are defined in both techniques too. Our approach is similar to [21] since they all use the decision table (DT) for representation of state-based systems. Mappings between rules and DTs in order to elaborate advantages of tabular representation are used in our and [22] approaches. Our approach as well as many others, example of which is [23], exploits advantages of tabular representation in order to perform verification by transforming certain representation to DTs. While performing static verification of PLA KB, we use results of Vanthienen *et al*. [14], whereas when analysing the dynamic constraints we use the reachable state method that is similar to the functional validation method suggested by Preece *et al*. [20] in a view of analysis of execution paths. Rule manager [24] checks for the same set of anomalies as our proposed technique does. The following anomalies are being explored: contradictions, redundancies, and incompleteness. Our approach is similar to that of [25] since they both offer the use of declarative languages for a description of the user needs. We use PLA model concepts for representation of the object structure, while Specht [25] use "object as theory" model. In our approach, in contrast to the compared one, we emphasise verification of the created declarative description. Bergeron *et al.* [26] state that static and dynamic analyses are complementary. They propose to use static analysis first. In our work at the beginning of the checking we use static verification technique that complements the functional validation. Our approach, as another ones (e.g. [27], [28], [29]), uses static and dynamic analyses in a pair in order to comprehensively analyse the application being developed.

7. Conclusions

In this paper we have presented a technique for representation and analysis of business rules. Using the illustrative example we showed as follows:

- PLA model can be successfully used as a background for business rules representation; It gives us the following benefits:
  - PLA related validation method for analysis of dynamic properties using reachable states graph can be successfully applied while checking general dynamic constraints of business rules, i.e. — absence of static deadlocks, final state reachability, boundedness of countable state parameters, and completeness;
  - Tabular verification method can be successfully applied for analysis of static constraints of business rules expressed in PLA knowledge base. These constraints are non-redundancy, non-conflict/ness, non-deficiency

Application area of the approach proposed are business rules describing interactions between structural parts of a system or organisation. This statement is based on a fact that PLA model is primarily suited for specification and analysis of complex systems interacting between themselves. The scalability of the proposed technique is limited by software tools that are used in creating the specification. The limitation requirements for these tools are defined in [13], [19].

References

1. Facchi, C., Haubner, M., Hinkel, U. *The SDL Specification of the Sliding Window Protocol Revisited*. *Technical Report TUM-I9614*. Munchen: Technical University, 1996.
2. Spirakis, P., Tampakas, B., Antonis, K. Hatzis, and K.Pentaris, G. *Specification Languages of Distributed and Communication Systems: State of the Art. ESPRIT Long Term Research project Nr.20244 report.* 1996.
3. Pranevicius, H. *Formal specification and analysis of computer network protocols*: *Monograph.* Vilnius: Technologija, 2005.

4. Spreeuwenberg, S. Using Verification and Validation Techniques for High-quality Business Rules, *Business Rules Journal*, Vol. 4, No 2, 2003.
5. Halle, B. *Business Rules Applied: Building Better Systems Using the Business Rules Approach.* John Wiley, 2001.
6. Date, C.J. *What not how: The business rules approach to application development,* Addison-Wesley Longman, 2000.
7. Baisley, D. A Metamodel for Business Vocabulary and Rules: Object-Oriented Meets Fact-Oriented, *Business Rules Journal,* Vol. 5, No 7, 2004.
8. Gerrits, R. Verification of Business Rules Utilization, *Business Rules Journal*, Vol. 4, No 12, 2003.
9. Bajec, M., Krisper, M. A methodology and tool support for managing business rules in organisations, *Information Systems*, Vol. 30, No 6, 2005, pp. 423-443.
10. Martin, J., Odell, J. *Object-Oriented Methods, a Foundation*. NJ: Prentice-Hall, Englewood Cliffs, 1998.
11. Bruynooghe, M., Pelov, N., Denecker, M. Towards a more declarative language for solving finite domain problems. In: *ERCIM/ COMPULOG Workshop on Constraints*. 1999, pp. 1-14.
12. Velde, W.V., Aamodt, A. *Machine learning issues in CommonKADS, KADS-II/ TII.4.3/ TR/ VUB/ 002/ 3.0.* 1994.
13. Vanthienen, J. *Prologa v.5 User's manual*. Katholieke Universiteit Leuven, 2000.
14. Vanthienen, J., Mues, C., Wets, G. Inter-tabular Verification in an Interactive Environment. In: *4th European Symposium on the Validation and Verification of Knowledge Based Systems*. Katholieke Universiteit Leuven, 1997, pp. 155-165.
15. Budnikas, G. *Development and analysis of aggregate specifications using knowledge bases: PhD dissertation.* Kaunas: Kaunas University of Technology, 2004.
16. Pranevicius, H., Budnikas, G. Creation of Estelle/Ag Specifications Using Knowledge Bases, *Informatica*, Vol. 14, No 1, 2003, pp. 63-74.
17. Preece, A. Evaluating Verification and Validation Methods in Knowledge Engineering, *Micro-Level Knowledge Management*, 2001, pp. 123-145.
18. Preece, A., Shinghal, R. Foundation and Application of Knowledge Base Verification, *International Journal of Intelligent Systems*, No. 9, 1994, pp. 683-702.
19. *CLIPS Reference Manual, Basic Programming Guide*. 2003.
20. Preece, A., Grossner, C., Radhakrishnan, T. Validating Dynamic Properties of Rule-Based Systems, *International Journal of Human-Computer Studies*, No. 44, 1996, pp. 145-169.
21. Arentze, T., Borgers, A.W.J., and Timmermans, H.J.F. The integration of expert knowledge in decision support systems for facility location planning, *Computers Environment and Urban Systems*, Vol. 19, No 4, 1995, pp. 227-247.
22. Chambers, T.L., Parkinson, A.R. Knowledge representation and conversion for hybrid expert systems, *Journal of Mechanical Design*, Vol. 120, No 3, 1998, pp. 468-474.
23. Degelder, J., Steenhuis, M. A knowledge-based system approach for code-checking of steel structures according to Eurocode 3, *Computers and Structures*, Vol. 67, No 5, 1998, pp. 347-355.
24. *Rule Manager, User manual*. 2007. Available – http://www.acumenbusiness.com/
25. Specht, G. O!-LOLA - Extending the Deductive Database System LOLA by Object-Oriented Logic Programming, *Informatica*, Vol. 9, No 1, 1998, pp. 107-118.
26. Bergeron, J., Debbabi, M., Desharnais, J., Erhioui, M., Lavoie, Y., Tawbi, N. Static detection of malicious code in executable programs*, International Journal of Requirement Engineering,* 2004, pp. 1-9.
27. Mi, P., Scacchi, W. A Knowledge-based Environment for Modelling and Simulating Software Engineering Processes, *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No 3, 1995, pp. 283-294.
28. Regnell, B, Runeson, P. Combining Scenario-based Requirements with Static Verification and Dynamic Testing, In: *4th International Workshop on Requirements Engineering: Foundation for Software Quality*. Pisa, Italy, 1998, pp. 1-12.
29. Wang, T.-H., Edsall, T. Practical FSM Analysis for Verilog, In: *IEEE Int'l Verilog HDL Conf. and VHDL Users Forum*. Los Alamitos, California: IEEE Computer Soc. Press, 1998, pp. 52-58.

VARIOUS ASPECTS OF SIMULATION'S USAGE TO CREATE REAL SOLUTIONS WITH „DARSIR" CONCEPTION AS AN EXAMPLE

Vadim Zuravlyov[1], Eleonora Latisheva[2]

Riga Technical University
[1] Pernavas 1-2, Riga, LV-1012, Latvia
Phone: +371 28346517. E-mail: zuravlyov@yahoo.com
[2] Meza 1/3, Riga, LV-1048, Latvia
Phone: +371 7089576. E-mail: elat@inbox.lv

Nowadays concepts of designing data management systems and technologies exist, and they can be used to create new real solutions without using typical programming. In this work authors present one of the ways to create real solutions – the usage of simulations.

Different technologies can be used to enable necessary functionality, but the authors have chosen Radio Frequency Identification (RFID), as the most perspective direction nowadays.

One of their concepts of designing data management systems, that authors want to present, is a "Data-And-Rules-Save-In-Resource" ("DARSIR") [1] conception, that can be used with RFID technology. Main case of this concept is to achieve universality, flexibility, not using other sources (such as database) to accept the decision that the information (attributes and rules) is saved in RFID tag.

In this work authors present various aspects of simulation's usage to create a solution. Authors also present the main steps from simulation creation to real solution.

Keywords: "*DARSIR" conception, Radio Frequency Identification, XML, Resource Physical Mark-up Language, simulation*

1. Introduction

Nowadays one of the most important tasks that many specialists are trying to decide in information technologies is founding an easy way for creating new real solutions. Various concepts of designing data management systems and technologies exist, and they can be used to create new real solutions without using typical programming. In this work authors present one of the ways to create real solution – usage of simulations.

Different technologies can be used to enable necessary functionality, but authors have chosen Radio Frequency Identification (RFID), as the most perspective direction nowadays. This technology is chosen in [1]. RFID is an automatic identification method, which relies on storing and remote data retrieving with the use of devices called RFID tags. RFID tag is possible to place in physical object, embedding in physical object or to construct interaction with other physical objects (for example sensors). The approach gives a variety of architectural solutions.

One of their concepts of designing data management systems, that authors want to present, is a "Data-And-Rules-Save-In-Resource" ("DARSIR") [1] conception, that can be used with RFID technology. Main case of this concept is to achieve universality, flexibility, not using other sources (such as database) to accept the decision that the information (attributes and rules) is saved in RFID tag. This concept data storage type provides relieve transformation from simulation scheme to real solutions.

In this work authors present various aspects of simulation's usage to create a solution Authors present their opinion for main steps from simulation creation to real solution. In this paper the main elements of "DARSIR" conception, data storage type, attributes, rules etc. are described, working principles of the real solution and simulation are shown. For better understanding of practical usage, last step is the description of a concrete problem: it is necessary to organize regulation of temperature inside. Authors are describing aspects of simulation's creation of this task, as well as the real solution of it in RPML, which is a final result.

2. Description of the Conception

The first step of simulation's usage for solution creation is defining key elements of conception. The key elements of the "DARSIR" concept are resources. A resource is any object of the living or lifeless nature, which is involved in working process of information system. The information of a concrete resource and its interrelationship with other objects (or types of objects) must be stored in this resource.

Different technologies can be used to enable necessary functionality, but authors have chosen Radio Frequency Identification (RFID), as this technology is chosen in [1].

RFID (see more in [2]) is an automatic identification method, relying on storing and remotely retrieving data using devices the so-called tags. Basic element of this technology is a tag, it is an object that can be attached to or incorporated into a product, animal, or person for the purpose of identification using radio waves. It is possible to store up to 1 Mb data in each tag. Such size of data allows uniting the list of attributes (data) with functionality (rules) in resource.

3. Working Principles of the Real Solution and Simulation

Next step of our process is defining working principles of conception and ways of simulation usage in creation of new real solution. The "DARSIR" concept working process key element is RFID reader. RFID reader has a built-in embedded system that loads to and carries out rules from RFID tags. To load rules from RFID tags, they must be located in RFID reader working zone. If RFID readers are connected in network of RFID readers, that RFID readers realize RFID tag information (attributes and rules) exchange.

Resources (RFID tags) are important for real solution and simulation, because for creation of working process is necessary to change information (attributes and rules) only in resources (in RFID tags). Other elements of working process (such as RFID reader) usually do not need any modifications.

4. The Structure of Stored Information

Further it will be necessary to define the structure of stored information in simulation. From the previous chapters it is known, that all information that can be changed, is stored in resources (RFID tags). And all information that is needed in real solution customer must edit in simulation scheme. It is not possible to create finished real solution if this condition is not executed

Next step is to describe format of storing information in conception. In the "DARSIR" conception information is stored in resource in PML (Physical Mark-up Language) – the format modification of XML. PML is described in [3] as "a simple, general language for describing physical objects for use in monitoring and control of a physical environment – particularly through the Internet. Applications include inventory tracking, automatic transaction, supply chain management, machine control and object-to-object communication."

In the "DARSIR" concept description article [1] is represents a new modification of PML, that is been developed for this concept, that calls is RPML (Resource Physical Mark-up Language). Further in this document the authors gives examples in the language RPML, but the places, where syntax coincides with the official version of the language PML, will not be separately described (it is possible to find them on official page [3]).

To create real solution it is necessary to create RPML document for each element from the finished simulation model and load it in appropriate RFID tag. RPML documents have the attributes and rules, main types for them and kinds of stored information will be defined in the following chapters.

4.1. Attributes

Each resource has own unique features, but it is possible to emphasize some standard kinds of such features:

- Physical – any resource possesses physical properties, it is possible to fix from what material the resource is made, the length of the resource, etc.
- Sensors – the resource can have built-in sensor (such as thermometer, hydrometer) or many sensors; the current value of sensor is stored in resource as attribute.
- Entity – the resource can have its owner (it can be the person, the organization, the country, etc.), so here such parameters, as a name of the owner, the name of the organization, the address, etc., are registered.
- Location – the coordinates of a resource concerning the reading out device, it can be in 2D (X and Y), also in 3D (X, Y and Z) coordinates.
- Configuration – resource can consist of another resources (e.g. pallet shipment), here the unique numbers of other resources can be listed.
- History – parameters of any kind, that has timing component, when these parameters have been fixed. It is very important to remember, that memory of a resource is limited and consequently it is necessary to limit final number of possible records
- Variables – any program can be stored in a resource, and for its work variables are necessary. There are standard types of variables (such as Integer, String, Date, etc.), and also non-standard such as arrays (defined in PML).

So far the authors have described the basic kinds of features, but there are also other, not listed kinds of features. All of this depends on defined tasks and on variants of their solution. It would also be desirable to note, that exist tags that communicate with the carrier. The typical example of such tag is sensors. For example, the thermal sensor control can write down a current condition of temperature in the tag. Also any mechanism (being carrier of tags) can write down on it the information. For example, the printer can write down on the tags the working condition of itself (such as printing, on, off, waiting, error etc.).

Some standard kinds are possible to use in simulation as static data, such as: physical, entity. But some standard kinds are possible to change in simulation process, such as: sensors, location, configuration, history and variables.

4.2. Rules

In simulation, rules can be created by simulation's languages. In this chapter authors are defining format, how simulation rules must to be converted for creation of real solution.

The ideology is similar to usual programming languages, and analogical examples in programming language C ++ are specified for better understanding of the chapter. In a resource the rules are presented as the program, in the form of XML.

The executable block of the program, called "task", can be defined as procedure. The very first executable task in a resource is called "main". In the own task there is an opportunity to apply some kinds of operations:

1. Executable – actions which can execute resources. Typical actions: to include, to switch off, to start, etc.
2. Boolean – the Boolean operators, consisting of condition and task that will be executed (or on the contrary not to be executed) if condition is true.
3. Predefined – the built in operators who are predetermined by the system (for example, for definition of distance between two resources, operation "Locate" is used).

The syntax of the task:

```
<task label = "procedure" variable="value">
    ..
    <return type = "variable">value</return>
</task>
```

By the tag "task" it is defined the working block, where the name of the task (procedure) is required to be defined. Then the names of variables with their values are listed. The authors want to emphasize that it is necessary to define those variables which are transferred into "task". Inside of the procedure there are operations. If it is necessary to return any values, then the type of procedure (type) or the name (variable) or the returned value (value) are defined. If this task is used as a condition in verifying operation, then the last value in the last tag "return" is considered.

The syntax of the checking operation:

```
<if label = "name" condition="condition">
    <true>..</true>
    <false>..</false>
</if>
```

By means of tag "if" it is possible to define a condition. Conditions of comparison are set by an alphabetic combination: EQ (=), GT (>), LT (<), GE (=>), LE (<=), NE (<>). If the condition is true, then the working block, that is located in the tag "true", is to be executed. Otherwise, the block of the tag "false" must be executed. Verification operation has a name on which it is possible to identify verification operation (name). If the tags "true" and "false" don't exist, tag "true" will be considered.

5. The Simulation Creation from Real Solution

Sometimes customers have situations, when they have real solution ready and need to do some changes for this solution with simulation usage. For this functionality with simulation usage it is possible to have the following situations and possible decisions:

- Real solution is created by "DARSIR" conception – the best way in this situation, is possible to transfer all attributes and rules from resources to simulation scheme;
- Real solution with all elements with built-in RFID tags – in this situation it is possible to load some part of attributes in simulation scheme, as physical (if present RFID identification owner type description), sensors (if present), entity (if present in RFID tag), location (possible define by RFID readers), configuration (possible define by RFID readers), etc.;
- Real solution all elements without built-in RFID tags – in this situation first step is built-in RFID readers, it is possible to load some part of attributes in simulation scheme (but usually less than in previous situation), as physical (if present RFID identification owner type description), sensors (if present), location (possible define by RFID readers), configuration (possible define by RFID readers), etc.

6. The Example

So far, the theoretical basis of the various aspects of simulation's usage for creation of real solution with "DARSIR" conception was briefly described in this document. For better understanding of the practical usage of the simulation for creating real solution with "DARSIR" concept authors described a concrete problem, and its solution in RPML with remarks of simulation creation.

Condition of the problem: it is necessary to organize regulation of temperature inside. Initial data are the conditioner (downturn of temperature), a heater (rise of temperature) and thermo sensor (capable to take temperature).

First of all the authors need to define attributes that can be used to solve the problem. In thermo sensor there is already been one parameter "Temperature", that changes on demand of attributes from the environment. There are also 2 parameters: "Min" (admissible minimum) and "Max" (admissible maximum).

Example:

```
<mat label = "thermosensor">
    <extrude type = "Temperature"> 0 </extrude>
    <integer label = "Min"> 20 </integer>
    <integer label = "Max"> 21 </integer>
</mat>
```

Simulation creation process is dependent on simulation language. It is reason why authors can only define attributes, which must be used in simulation. There is defined one sensor type attribute and two attributes of variable type.

The next step is definition of accessible executable operations. The conditioner and a heater have two executable operations "On" and "Off" that allow switching on and off these devices.

There will be a control block (or in other words – working algorithm) in thermo sensor:

1. If value of the parameter "Temperature" is greater then value of the parameter "Max", then switch off the heater and switch on the conditioner.
2. If value of the parameter "Temperature" is less then value of the parameter "Min", then switch off the conditioner and switch on the heater.

In RPML it looks as:

```
<task label = "main">
    <if label = "Hot" condition = "Temperature GT Max">
            <msr label = "Heater.Off"></msr>
            <msr label = "Conditioner.On"></msr>
    </if>
    <if label = "Cold" condition = "Temperature LT Min">
            <msr label = "Conditioner.Off"></msr>
            <msr label = "Heater.On"></msr>
    </if>
</task>
```

In simulation it is important to define the correct executable actions, so they can execute resources they need. If simulation have some errors in names that mean that in transformation process from simulation to real solution this error must be fixed. After that it is necessary to realize mechanism for name correction checking.

7. Conclusions

Various aspects of simulation's usage for creation of real solutions of "DARSIR" conception as example are described in this article. It is one of the possible easy ways for creating new solutions without using typical programming.

Authors have chosen one of the concepts of designing data management systems. The choice is "Data-And-Rules-Save-In-Resource" ("DARSIR") [1] conception, that can be used with RFID technology. This concept data storage type provides relieve transformation from simulation scheme to real solutions.

In this article authors present their opinion for main steps from simulation creation to real solution. Main elements of "DARSIR" conception, data storage type, attributes, rules etc. are described here. Working principles of the real solution and simulation are shown. For better understanding of practical usage, last step is described as concrete problem: it is necessary to organize regulation of temperature inside.

Various aspects discussed in this article can be used as a practical guide for the implementation of "Data-And-Rules-Save-In-Resource" as the easy way of creation new real solution. Such simulation task can be easily created with user-friendly graphical interface.

Acknowledgements

This work has been partly supported by the European Social Fund within the National Programme "Support for the Carrying out Doctoral Study Programme's and Post-Doctoral Researches" project "Support for the Development of Doctoral Studies at Riga Technical University".

References

1. Zuravlyov, V. Main principles of a new concept of designing data management systems. In: *The 7th international thematic issue Computer Science of the RTU scientific proceedings series Applied Computer Systems; 47th RTU International Scientific Conference, Riga, Latvia, October 12-14, 2006.* Riga: Riga Technical University, 2006
2. Sandip Lahiri. RFID sourcebook. N.J., IBM, London: Upper Saddle River, 2005.
3. *The Physical Markup Language* – http://web.mit.edu/mecheng/pml/

FORMAL DESCRIPTION OF DYNAMIC FEATURES IN PLA

Henrikas Pranevicius[1], Dalius Makackas[2], Agne Paulauskaite-Taraseviciene[3]

Department of Business Informatics, Faculty of Informatics
Kaunas University of Technology
Studentu 56, Kaunas, Lithuania
Tel. +370 681 52842, fax +370 37 451654
[1] E-mail: hepran@vik.ktu.lt, [2] damaka@vik.ktu.lt, [3] agne@ifko.ktu.lt

Dynamic structure systems are systems with capabilities to change not only their behaviour in time but all structure as well. For expanding range of specified systems formal notation has to be extended with opportunities to describe all futures of such systems. We present extended PLA formalism – DPLA, which is able to specify systems that can change their structure dynamically. Operators, for defining the actions of structural changes in DPLA, are presented. A formal description of structural changes in DPLA specification is defined using Z specification language.

Keywords: *multi-agent systems, formalization, DPLA, dynamic structure, Z*

1. Introduction

Dynamic structure systems are systems with capabilities to change not only their behaviour state in time but all structure as well. More often we need to specify systems which are interactive in dynamic environment changing type and number of its attributes, where objects are changing their structure in time and can perform autonomously (i.e., multi agent systems)[1]. A deep understanding of dynamic data structures is particularly important for modern, object-oriented languages as well [2], [3]. Whereas such systems mostly are large, complex and critical, there exists motivation for formal specification. Most of formal methods are able to specify static structure systems. Dynamic structure is an important capability, related to hierarchical structure, for simulating complex systems. For expanding range of specified systems, formal notation has to be extended with opportunities to describe all futures of such systems.

A piece-linear aggregate (PLA) is timed automata based specification formalism, which is widely used for creating simulation models (often simulation of computer network protocols) and their validation. PLA is very important for design complex real time systems, but only static structure. In this paper extended PLA formalism DPLA is presented, which is able to specify systems that can change their structure dynamically. For dynamics in DPLA, we create four protocols for defining the sequence of actions of structural changes: the creation of relationships (CR), the deletion of relationships (DR), the creation of new aggregates (CA), and the deletion of aggregates (DA). A formal description of structural changes in DPLA specification is defined using Z specification language. Finally, an example of the use of these extensions is included.

2. Formalization Possibilities of Dynamic Structure Systems

Most of formal methods enable the specification of static structure systems, which mean that dynamic features of systems are complex to describe or can't be described at all. For expanding the range of specified systems, formal notations should be extended with capabilities to describe all features of such systems. For example, to describe dynamic structure systems, the classical notation of DEVS (discrete event system specification) was extended by adding special functions (4). In dynDEVS structural changes are invoked by a model transition function ρ_α, which generates "incarnations" of dynamic DEVS and a network transition function ρ_N, which keeps the state and structure of models that belongs to the composition of network. Using the dynamic DEVS notation the existing models can be deleted or new created, couplings between models added and removed. Such capabilities of dynDEVS are also useful for modelling MAS [5] – which, besides their specific features, such systems are primarily dynamic systems with capabilities to change not only their state in time, but all the structure as well. For various experiments, the system JAMES (A Java-Based Agent Modelling Environment for Simulation) was created [6] in which agents are modelled as time triggered state automata. The virtual environment allows creating different scenarios and testing the behaviour of agents.

Another popular DEVS variation – parallel dynamic structure of discrete events specification network (DSDEN) [7] where dynamic structure system network is defined with special component (network executiveχ) and structure function (which maps the network structure states set S_χ and set of network structures $\sum^*$)

Dynamic structure systems are also modelled with different Petri Nets (e.g. dynamic Petri Nets, Coloured Petri Nets, high level Petri Nets). In DPN, the structure of the net can be changed dynamically, by inserting or deleting a transitions or places [8]. For example, modelling MAS with high level Petri Nets it can be concluded that "individual agents are modelled as tokens so that they migrate from one component to another by transition firing at runtime" [9]. It is difficult to determine which methods are better than others, because all these formal methods specifying dynamic structure systems highlight and test different features of such systems. As a result of the overview, we can make a consumption that automaton based formal methods are really suitable for describing dynamic structure systems, including multi-agent systems. That is the reason for deciding to use the formal method PLA for dynamic structure systems formalization.

3. Dynamic Approach of PLA

A piece-linear aggregate (PLA) is timed automata based specification formalism, which is widely used for creating simulation models (for simulation of computer network protocols often) and their validation [10]. In the application of the aggregate approach a system is represented as a set of interacting piece – linear aggregates. The PLA is taken as an object defined by a set of states Z, input signals X, and output signals Y (Fig. 1). The aggregate functioning is considered in a set of time moments $t \in T$. The state $z \in Z$, input signals $x \in X$, and output signals $y \in Y$ are considered to be time functions. Apart from these sets, transition H and output G operators must be known as well.

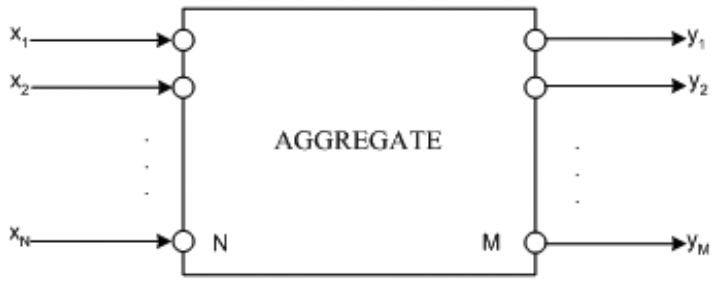

Fig. 1. *Scheme of aggregate*

3.1. DPLA: Dynamic PLA

The dynamic structure is an important capability, related to the hierarchical structure, for simulating complex systems. PLA is very useful for designing complex real time systems, but only for static structure systems. In dynamic structure systems the environment is varying in time changing the number of its attributes (e.g. new objects are added or removed). For specifying dynamic structure systems we modify some components of the original PLA and add some extra operators. The difference between PLA and DPLA (dynamic PLA) is presented below on Figure 2.

Differently from original PLA, in DPLA the number of events, input, output signals, controlling sequences, discrete and continuous component may vary in time. Each aggregate can have children aggregates in its content. dynDEVS notation is similar to DPLA, but differently from dynDEVS coupled model (which is like a frame of its component), a parent aggregate is an ordinary aggregate with its own internal events, external event, outputs, inputs and so on [11].

In DPLA there are two types of dynamic structural changes:

The level of a single aggregate: Structural changes of a single aggregate (i.e. a new initial state can be added $E''(t_{m+1}) = E''(t_m) \cup e_i''$ or deleted $E''(t_{m+1}) = E''(t_m) / e_i''$).

The level of aggregate system: Structural changes between two or more aggregates (i.e., a new link between aggregates can be added, or an existing link deleted; also new aggregates can be created $A(t_m) = A(t_{m-0}) \cup (A_{new} : type_i)$ or eliminated $A(t_m) = A(t_{m-0}) / (A_{old} : type_i)$).

PLA		DPLA
Input signals $X = \{x_1, x_2, ..., x_N\}$	→	**Input signals** $X(t_m) = \{x_1, x_2, ..., x_N\}$
Output signals $Y = \{y_1, y_2, ..., y_N\}$	→	**Output signals** $Y(t_m) = \{y_1, y_2, ..., y_N\}$
External events $E' = \{e_1', e_2', ..., e_N'\}$	→	**External events** $E'(t_m) = \{e_1', e_2', ..., e_N'\}$
Internal events $E'' = \{e_1'', e_2'', ..., e_f''\}$	→	**Internal events** $E''(t_m) = \{e_1'', e_2'', ..., e_f''\}$
Discrete component $\nu(t_m) = \{\nu_1(t_m), \nu_1(t_m) ..., \nu_p(t_m)\}$	→	**Discrete component** $\nu(t_m) = \{A(t_m), R(t_m) ..., \nu_p(t_m)\}$
Number of system aggregates K	→	**Set of children aggregates** $A(t_m) = (Ag\_name)$
Matrix of input signals $R = \{r_{ji}\}, i = \overline{1, L}, j = \overline{1,2}$	→	**Set of relationships** $R(t_m) = (name : source\_Ag \mapsto t\arg et\_Ag)$
Matrix of output signals $H = \{h_{ij}\}, i = \overline{1, K}, j = 1, \overline{\max_{1 \le i \le K}\{M_K\}}$		$R : Y^* \to X^*, where$ $Y^* = \bigcup_{A_i \in A(t_m)} Y_{A_i}, \quad X^* = \bigcup_{A_j \in A(t_m)} X_{A_j}$
Continuous component $z_\nu(t_m) = \{w(e_f'', t_m)\}$	→	**Continuous component** $z_\nu(t_m) = \{w(e_f'', t_m)\}$
Controlling sequences $e_i'' \mapsto \{\xi_j^{(i)}\}$	→	**Controlling sequences** $V(t_m) = \{e_i'' \mapsto \{\xi_j^{(i)}\}\}$
Initial state $z(t_m)$	→	**Initial state** $z(t_m)$
Transition operator $H[z(t_m, e_i)]$	→	**Transition operator** $H[z(t_m, e_i)]$
Output operator $G[z(t_m, e_i)]$	→	**Output operator** $G[z(t_m, e_i)]$

Fig. 2*. Comparison of PLA and DPLA*

3.2. Protocols and Z schemas of structural changes

For external structural changes in DPLA, we create four protocols, defining the sequence of such actions: the creation of relationships –CR, the deletion of relationships – DR, the creation of new aggregates – CA, and the deletion of aggregates – DA [12]. Some constrains are applied to avoid conflict. For example, an aggregate whose $A(t_m) = \varnothing$ can't delete or add other aggregates. Only coordinator can create and delete aggregates from its own set of child aggregates $A(t_m)$.

If one aggregate, which belongs to one coordinator, decided to create a new link with the aggregate, in its turn which belongs to the second coordinator, it has to be removed from the content of the first coordinator and after that to be added to the second one (Fig. 3).

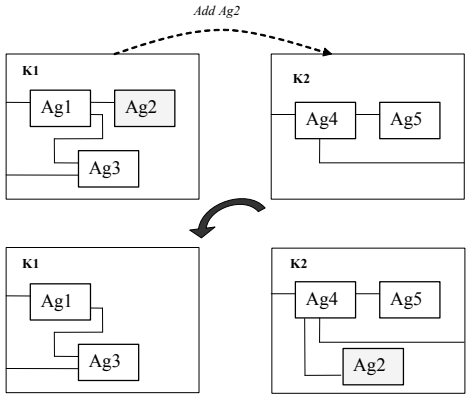

Fig. 3. *Structural changes in coordinator level*

These protocols are usable for realization of DPLA, but writing specification using this notation is more practical to include operators for such actions of system structural changes. For this purpose we decided – Z formal language [13] for formal description of operators DR, CR, DA and CA, which corresponded to the protocols. These common operators significantly reduce specification size, because we have to be predefined just

one time. Performing structural changes by these operators, an aggregate can evolve in the process and differently from dynDEVS, we don't have to predefine all possible models variations of each aggregate.

CR operator is presented on Figure 4. Operator keeps all the information needed for creating a new link. The name of link *name*, associated aggregates *source_Ag* and *target_Ag* has to be denoted.

CR=(name: source_Ag → target_Ag)

The main condition is that both aggregates must belong to the same coordinator. The new link is added to the coordinator's set of relationships $R(t_m)$. In general case, if an aggregate gets a request to add new link, it checks: if the request includes the signal with its name equal to *target_Ag*; if the request includes the signal with its name equal to *source_Ag*.

In the first case, the input signal is added to the set of input signals X and additional transition operator H for processing this signal. In the second case, the new output signal is added to the set of output signals Y, and additional output operator G, which generates the new signal. All transition and output operators which are added must be predefined in specification of aggregate. During structural changes these operators are added to the main H and G sets of aggregate.

CR

$c : A$

$a_i : A$

$a_j : A$

$r? : CR$

$result!: REZ$

$a_i \in c.child \wedge a_j \in c.child \wedge$

$r?.source = a_i.my\_ID \wedge r?.target = a_j.my\_ID \wedge$

$c.R' = c.R \cup r? \wedge$

$(a_i.Y' = a_i.Y \cup \{r?.signal\} \wedge$

$a_i.E' = a_i.E_1 \cup a_i.E_2' \wedge a_i.E_2' = a_i.E_2 \cup \widetilde{E}? \wedge$

$a_i.z' = a_i.z \cup \bigcup_{e \in \widetilde{E}} \{e \mapsto (\mathbf{R}^+ \to \mathbf{R}^+)\} \wedge$

$a_i.H' : a_i.E' \times a_i.z' \to a_i.z' \wedge a_i.G' : a_i.E' \times a_i.z' \to a_i.Y') \wedge$

$(a_j.X = a_j.X \cup \{r?.signal\} \wedge$

$a_j.E' = a_j.E_1' \cup a_j.E_2 \wedge a_j.E_1' = a_j.E_1 \cup \{\widetilde{e}?\} \wedge$

$a_j.H' : a_j.E' \times a_j.z \to a_j.z) \vee$

$result! = error$

Fig. 4. *Z schema of CR operator*

DR

$c : A$

$a_i : A$

$a_j : A$

$r? : DR$

$result! : REZ$

$a_i \in c.child \wedge a_j \in c.child \wedge$

$r.source = a_i.my\_ID \wedge r.target = a_j.my\_ID \wedge$

$r \in c.R \wedge c.R' = c.R / \{r?\}$

$(a_i.Y' = a_i.Y / \{r?.signal\} \wedge$

$a_i.G' : a_i.E' \times a_i.z \to a_i.Y') \wedge$

$(a_j.X = a_j?.X / \{r?.signal\} \wedge$

$a_j.E' = a_j.E_1' \cup a_j.E_2 \wedge a_j.E_1' = a_j.E_1 \cup \{\widetilde{e}\} \wedge$

$a_j.H' : a_j.E' \times a_j.z \to a_j.z) \vee$

$result! = error$

Fig. 5. *Z schema of DR operator*

Appropriative 3 cases for source aggregate (Table 1) and 1 for target aggregates is defined in this operator (Table 2). Below all cases for source aggregate are listed:

1. If output operator G, which generates the new output signal, is invoked by event e, which exists in aggregate $e \in E(t_m)$ at the moment tm, where $E(t) = E'(t) \cup E''(t)$. For example, transport A has event – milk delivery every morning, during which it delivers milk to some stores y(milk). Store B wants to create a link with it and get a milk as well x(milk).

Table 1. Addition of new link "new" in point of source aggregate

	Cases	Actions
1	$H(e)$: ..... $G(e) : y(new)$	The new output signal is added to the set of output signals of source aggregate $Y(t_{m+1}) = Y(t_m) \cup new$. Output operator G is added to the set of output operators $G(t_{m+1}) = G(t_m) \cup G(e)$
2	$H(e''_?)$: ..... $e''_? \notin E''(t_m)$ $G(e''_?) : y(new)$	The new output signal and the new internal event, during which new output signal is generated, are added $Y(t_{m+1}) = Y(t_m) \cup new$, $E''(t_{m+1}) = E''(t_m) \cup e''_?$. Transition H and output G operators are included to the additional sets of aggregate $G(t_{m+1}) = G(t_m) \cup G(e''_?)$ and $H(t_{m+1}) = H(t_m) \cup H(e''_?)$. Since the set of internal events is changed, the state of an aggregate is changing as well $z(t_{m+1}) = z(t_m) \cup w(e''_?(t_m))$
3	$H(e'_?)$: ..... $e'_? \notin E'(t_m)$ $G(e'_?) : y(new)$	The new output signal and G operator are added $Y(t_{m+1}) = Y(t_m) \cup new$, $G(t_{m+1}) = G(t_m) \cup G(e'_?)$. Whereas, doesn't exists the external event, after which processing, the new output signal is generated, extra CR has to be invoked for this external event.

2. If source aggregate's output operator G, which generates the new output signal, is invoked by internal event e", which doesn't exist in the aggregate $e'' \notin E''(t_m)$ at the moment *tm*. For example, internal event of transport A – milk delivery every morning, which generates signal y(milk) doesn't exist at the moment. Store B wants to create a link with it and get milk.

3. If source aggregate's output operator G, which generates the new output signal, is invoked by internal event e', which doesn't exist in the aggregate $e' \notin E'(t_m)$ at the moment *tm*. For example, external event of transport A – a permit of morning delivery, after which output signal y(milk) can be generated doesn't exist in aggregate at the moment. Store B wants to create a link with it and get milk.

At the point of target aggregate the structural changes are more general; new input signal and H(new) operator has to be added. We are not interested if appropriative G(new) generates any signals or not, because we don't include it to the G operators set. For example, store B wants to get milk x(milk) from transport A.

Table 2. Addition of new link "new" in point of target aggregate

	Cases	Actions
1	$H(new)$:	The new input signal is added to the set of input signals of target aggregate $X(t_{m+1}) = X(t_m) \cup new$. Transition operator H is added to the set of output operators $H(t_{m+1}) = H(t_m) \cup H(new)$

DR operator is presented on Fig. 5. Operator keeps all the information needed for deleting a link. The name of link *name*, associated aggregates *source_Ag* and *target_Ag* has to be denoted.

DR=(name: source_Ag → target_Ag)

The main condition is the same as in CR operator; both aggregates belong to the same coordinator.

If an aggregate gets a request for a link deletion it checks, if the request includes such a signal, where its name is equal to source_Ag. If it's true, then additional output signal is removed from its output signals set *Y*, and output operator *G* for this signal generation. If its name is equal to tartget_Ag, and removes additional input signal with transition operator *H* for denoted signal. Meanwhile, coordinator removes denoted link from its set of relationships.

Appropriative 3 cases for source aggregate (Table 3) and 2 for target aggregates are defined in this operator (Table 4).

1. If source aggregate's output operator G, which generates the old output signal, is invoked by event e, which changes the state of the source aggregate. For example, transport A has event – milk delivery every morning, during which it counts days of morning shift $Count(t_{m+1}) = Count(t_m) + 1$ and delivers milk to some stores y(milk). Store B wants to delete a link and refuse delivery of milk x(milk).

2. If source aggregate's output operator G, which generates the old output signal, is invoked by internal event e, which doesn't change the state (neither continuous nor discrete component) of the source aggregate. For example, transport A has event – milk delivery every morning, during which it doesn't change his state and generates signal y(milk) only. Store B wants to delete a link with it and refuse delivery of milk x(milk).

3. If source aggregate's output operator G, which generates the old output signal, is invoked by external event e, which doesn't change the state (neither continuous nor discrete component) of the source aggregate. For example, transport A gets external event – a permit of milk delivery every morning, during which generates signal y(milk). Store B wants to delete a link and refuse delivery of milk x(milk).

Table 3. Deletion of link "old" in point of source aggregate

	Cases	Actions
1	$H(e)$: $z(t_{m+1}) \neq z(t_m)$ $G(e): y(old)$	The old output signal is deleted from the set of output signals of source aggregate $Y(t_{m+1}) = Y(t_m) / old$. Output operator G is deleted from the set of output operators $G(t_{m+1}) = G(t_m) / G(e)$
2	$H(e''_?)$: $z(t_{m+1}) = z(t_m)$ $G(e''_?): y(old)$	The old output signal and internal event, during which this signal is generated, are deleted $Y(t_{m+1}) = Y(t_m) / old$, $E''(t_{m+1}) = E''(t_m) / e''_?$. Transition H and output G operators are deleted as well $G(t_{m+1}) = G(t_m) / G(e''_?)$ and $H(t_{m+1}) = H(t_m) / H(e''_?)$. Since the set of internal events is changed, the state of an aggregate is changing as well $z(t_{m+1}) = z(t_m) / w(e''_?(t_m))$
3	$H(e'_?)$: $z(t_{m+1}) = z(t_m)$ $G(e'_?): y(old)$	The old output signal and G operator are deleted $Y(t_{m+1}) = Y(t_m) / old$, $G(t_{m+1}) = G(t_m) / G(e'_?)$. Whereas, external event, after which processing, the old output signal is generated doesn't change the state of aggregate, extra DR has to be invoked for this external event.

Two cases for the target aggregate using DR operator are defined below in Table 4. In this case we are interested if the output operator of the deleted signal generates any signal or not.

1. If operator G of deleting signal doesn't generate any signal. For example, store B wants to delete a link with transport A, refusing delivery of milk.

2. If operator G for deleting signal generates external signal. For example, store B wants to delete a link with a transport A, refusing delivery of milk, but store B getting a milk from transport A x(milk) distributes it to others stores y(distribute).

Table 4. Deletion of link "old" in point of target aggregate

	Cases	Actions
1	$H(old)$: ..... $G(old):\varnothing$	The old input signal is deleted from the set of input signals of target aggregate $X(t_{m+1})=X(t_m)/old$. Transition operator H is deleted from the set of transition operators $H(t_{m+1})=H(t_m)/H(old)$
2	$H(old)$: ..... $G(old):y_1(\times\times\times)$	The old input signal and the other signal, which is generated by output operator G of old signal, are deleted $X(t_{m+1})=X(t_m)/old$, $Y(t_{m+1})=Y(t_m)/y(\times\times\times)$. Transition H and output G operators of old signal are deleted as well $G(t_{m+1})=G(t_m)/G(old)$ and $H(t_{m+1})=H(t_m)/H(old)$. Whereas, G operator generates a signal, it has to be removed. Extra DR is invoked for this external event

Using CA operator (Fig. 6), only coordinator can create children aggregates of lower level, defining the name of the new aggregate and its type. Coordinator creates new aggregate, adds it to the set of its children aggregates $A(t_{m+1})=A(t_m)\cup\{new1:type_i\}$ and creates relationships $R(t_{m+1})=R(t_m)\cup\{dyn\_struct:new1\to 0, dyn\_struct^*:0\to new1, done^*:new1\to 0\}$ for dynamic structural changes dyn_struct, dyn_struct*, done*. All aggregates in DPLA have these signals by default.

Deleting aggregates with operator DA (Fig. 7), primarily all links with an aggregate, which has to be deleted, are removed using DR operator. After that coordinator start structural changes in its level: deletes this aggregate from its set of child aggregates and removes default links with it.

CA

$c:A$

$a_i?:CA$

$r_1,r_2:R$

$result!:REZ$

$a_i?\notin c.child\ \wedge$

$c.child'=c.child\cup\{r_1,r_2\}$

$r_1.source=c.my\_ID\wedge r_1.target=a_i?.my\_ID\wedge$

$r_2.source=a_i?.my\_ID\wedge r_2.target=c.my\_ID\wedge$

$c.R'=c.R\cup\{r_1,r_2\}$

$result!=error$

Fig. 6. Z schema of CA operator

DA

$c:A$

$a_i?:DA$

$r_1,r_2:R$

$result!:REZ$

$a_i?\in c.child\wedge$

$c.child'=c.child/\{a_i\}\wedge$

$ai?.z'=NULL\wedge$

$r_1.source=c.my\_ID\wedge r_1.target=a_i?.my\_ID\wedge$

$r_2.source=a_i?.my\_ID\wedge r_2.target=c.my\_ID\wedge$

$c.R'=c.R/\{r_1,r_2\}\vee$

$result!=error$

Fig. 7. Z schema of DA operator schema

The above-mentioned three messages common for all DPLA specifications, which invoke defined operators:

(dyn_struct) – includes the information of what structure changes are expected: CR, DR, CA, DA. All other messages include such information as well. An aggregate informs its higher level aggregate – coordinator about expected structural changes;

(dyn_struct*) – an aggregate (coordinator) requests its additional children aggregates to make structural changes;

(done*) – an aggregate informs its coordinator about executed structural changes.

4. Application: the Population of Live Organisms

Chosen application represents the population of live organisms (i.e. cells, birds or animals). This scenario consists of three types of organisms (male form and female form) and a family, which consists of more than one organism. All organisms subsist on food, given by surrounding environment. Environment has limited recourse of food, which means that if organism's population grows too large, some of them won't survive. Every organism, which is not in family have the main goal – to make a pair with another organism. Two organisms in family can reproduce and increase population.

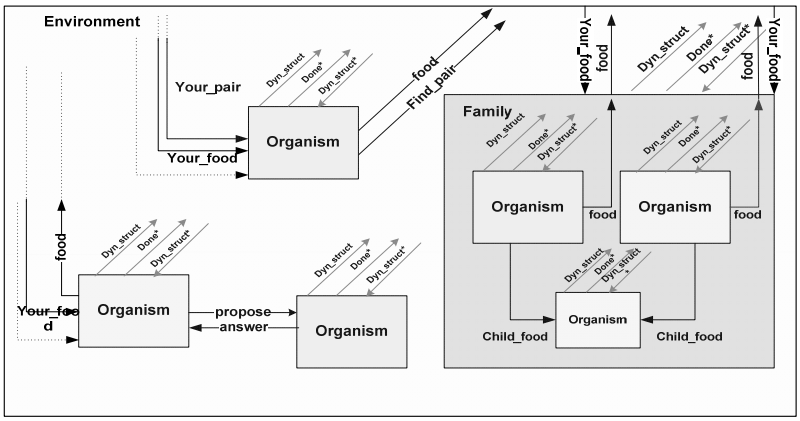

Fig. 8. An example in DPLA

In DPLA specification, every organism is considered as an aggregate (Fig. 8). All aggregates interact in environment, which is defined as higher level aggregate, containing lower level aggregates – all organisms and families. All single aggregates start theirs existence from the looking for pair. An aggregate- organism sends signal to its coordinator-environment with a request to find available organism for copulation. Whenever it gets a name of available aggregate, it sends the signal to coordinator with request to add new relationships. This signal invokes structural changes (CR).

Aggregate-Organism i

$$H(your\_pair)=a:$$
$$pair(t_{m+1})=a$$
$$G(your\_pair):$$
$$y(dyn\_struct)=CR$$
$$here,\quad CR=\begin{pmatrix} propose: my\_ID \to a \\ answer: a \to my\_ID \end{pmatrix}$$
$$H(dyn\_struct*)=CR:$$
$$Organism_i(t_{m+1})=(CR,(Organism_i(t_m)))$$
$$G(dyn\_struct*):$$
$$y(done*)=CR$$

Aggregate- Environment

$$H(dyn\_struct)=CR:$$
$$G(dyn\_struct):$$
$$y_i(dyn\_struct*)=CR$$
$$H(done*)=CR:$$
$$Environment(t_{m+1})=(CR,(Environment(t_m)))$$
$$G(done*):\varnothing$$

If an aggregate, initiated communication gets a positive answer for copulation, it starts creation of family; otherwise it removes relationships with "a" aggregate and starts its search for a pair anew. After two organisms create a family, they procreate offspring, which means that new aggregate is added in a "family" aggregate. In DPLA specification creation of new aggregate in family is invoked by CA operator.

$$H(e_1''):$$
$$\left.\begin{array}{l} Child(t_{m+1})=1 \\ family_i(t_{m+1})=(CA,(family_i(t_m))) \end{array}\right\} \quad {}^{\#}A(t_m)=2$$
$$\left.\begin{array}{l} Child(t_{m+1})=Child(t_m) \\ w(e_1'',t_{m+1})=t_m+\xi \end{array}\right\} \quad {}^{\#}A(t_m)\neq 2$$

$$G(e_1''):\quad y=\varnothing$$
$$here,\quad CA=(name_*:ORG)$$

A new organism, added into the family, grows up after some period. From this moment it can exists as a single aggregate. For such existence child aggregate has to be removed from "Family" content and moved up to the coordinator "Environment". From this situation another group of structural changes – transportation was detected. To improve functionality of DPLA we decided to add new protocols: move up – MU and move down – MD. In these protocols actions are based on above described operators DR, CA.

In MU protocol (Fig. 9) an aggregate "Ag1", sends the signal to its coordinator "AB" with a request to be moved up from a current level. Coordinator deleted all links with this aggregate and removes aggregate "Ag1" from the set of its children aggregates. After that, the higher coordinator "A" adds aggregate "Ag1" to its set of children aggregates and creates default links with it.

In MD protocol (Fig. 10) an aggregate, which wants to move, has to know want kind of aggregates exists in the same level of hierarchy. For this purpose "Ag1" sends the signal "View" to the coordinator. Coordinator responds to this message sending the names of its children. Aggregate "Ag1" has to choose one of the children of its coordinator "AB". Coordinator deletes all links with "Ag1" and removes it from its children set. Whenever

aggregate is removed from the content of "AB", the new coordinator "Ag2" adds the new aggregate "Ag1" to the set of children aggregates.

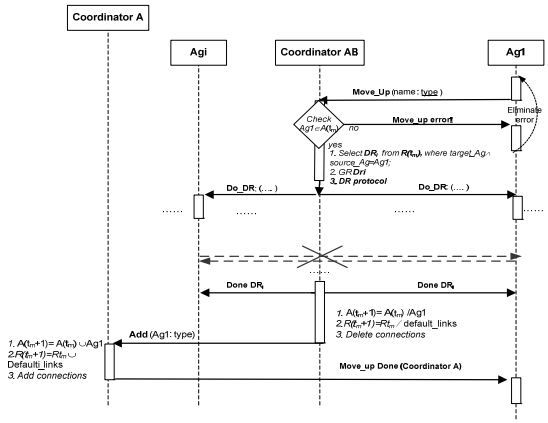

Fig. 9. *MU protocol*

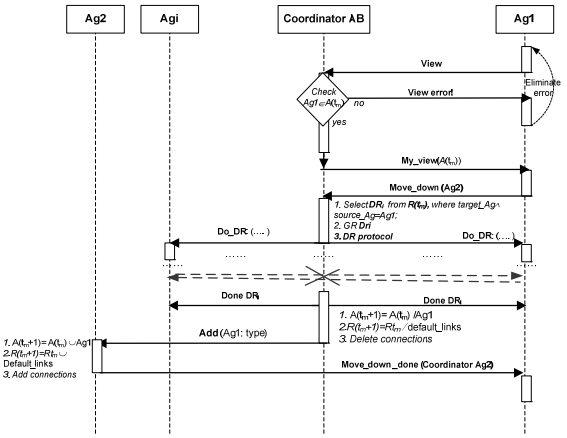

Fig. 10. *MD protocol*

In our specification grown child leaves the family and starts to exist as a normal organism. The grown child invokes MU operator and CR operator for communication with the new coordinator.

$$H(grown):$$
$$.....$$
$$G(grown): y(dyn\_struct) = MU$$
$$y(dyn\_struct) = CR$$

$$CR = \begin{pmatrix} food : my\_ID \to 0 \\ find\_pair : my\_ID \to 0 \\ your\_food : 0 \to my\_ID \\ your\_pair : 0 \to my\_ID \end{pmatrix}, \quad MU = (my\_ID$$

All organisms can die in two ways: starvation or after defined lifetime.

First way. e_1'' – Internal event, which defines nutrition stage of organism. If the organism is not dying form hunger, sends signal for a food to its environment; otherwise it starts structural changes with DA operator.

$$H(e_1''):$$
$$\left.\begin{array}{l} \mathrm{Res}(t_{m+1}) = \mathrm{Res}(t_m) - \theta \\ w(e_1'', t_{m+1}) = t_m + \alpha_1 \end{array}\right\} \quad \mathrm{Res}(t_m) > 0$$
$$z_\upsilon(e_1'', t_{m+1}) = \varnothing \qquad \mathrm{Res}(t_m) < 0$$

here, Res(t)– food resources;

$$G(e_1''):$$
$$y(food), \qquad \mathrm{Res}(t_m) > 0$$
$$y(dyn\_struct) = DA, \qquad \mathrm{Res}(t_m) < 0$$
$$here, \quad DA = (my\_ID : ORG)$$

Second way: e_2'' – internal event, which defines death of organism. The organism terminates all internal events and sends signal to environment for structural changes.

$$H(e_2''):$$
$$z_\upsilon(e_1'', t_{m+1}) = \varnothing$$
$$G(e_2''):$$
$$y(dyn\_struct) = DA$$

The above-mentioned approach permits using a single mathematical scheme to create models both for simulation and validation. The occurrence graph enabled us to validate specification and to analyse such features of the system as accessibility, necessity of continuous coordinates, bound of state coordinates, deadlocks and loops. This validation method is implemented in specification analysis tool PRANAS2, based on the aggregate approach.

5. Conclusions

In this paper, the dynamic approach of the formal method PLA is presented for formalization of dynamic structure systems. DPLA has enabled us to describe the structural changes of such systems. New two communications protocols – MU, MD are presented, which enable us to define such structural changes as transportation of aggregates into the new positions. These protocols as previous, define the order of the actions for dynamic structure changes applying some constrains (i.e., only coordinator can remove its children aggregates) to avoid the conflicts. Z schemas allow us to describe the presented protocols in formal description;

including operators which correspond to them. Usability of defined operators enable us to specify structural changes of systems (add of new aggregates, delete aggregates, add new links etc.) significantly to reduce the size of DPLA specification, to improve functionality and to formalize systems in compact, readable description. We represent DPLA and new operators using a concrete application, which involves various kinds of structural changes.

References

1. Fox, J., Beveridge, M., and Glasspool, D. Understanding intelligent agents: analysis and synthesis, *AI Communications,* Vol. 16, 2003, pp. 139-152.
2. Brandt, Joel R. Run-time modification of the class hierarchy in a Live Java Development Environment. Washington University, Department of Computer Science and Engineering, USA, 2004.
3. Guéhéneuc, Y.G. Dynamic class modification and creation in Java – http://gueheneuc.nexenservices.com/404.html, 2002.
4. Rohl, M., Uhrmacher, A.M. Controlled Experimentation with Agents – Models and Implementations. In: *5th International Workshop. "Engineering Societies in the Agents World".* Toulouse, France, October, 20-22, 2004.
5. Uhrmacher, A.M. A System Theoretic Approach to Constructing Test Beds for Multi-Agent Systems. In: *Discrete Event Modelling and Simulation Technologies. A Tapestry of Systems and AI-Based Theories and Methodologies: A Tribute to the 60th Birthday of Bernard P. Zeigler*: *Lecture Notes on Computer Science.* New York: Springer, 2001, pp. 315-339.
6. Schattenberg, B., Uhrmacher, A.M. Planning Agents in James. In: *Proceedings of the IEEE*, Vol.89, No2, February, 2001, pp. 158-173.
7. Barros, F.J. Representation of Dynamic Structure Discrete Event Models: A Systems Theory Approach. In: *Discrete Event Modelling and Simulation: Enabling Future Technologies*. Springer Verlag, 2000, pp. 167-184.
8. Graff, C.J., Giardina, C. Dynamic Petri Nets: new modelling technique for sensor networks and distributed concurrent systems. In: *Military Communications Conference, 2005, MILCOM 2005, IEEE*, Vol. 1, 2005, pp. 506-512.
9. Xu, D., Deng, Y. Modelling mobile agent systems with high level Petri Nets. In: *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC'2000)*, Vol.5, 8th -12th October, pp. 3177-3182.
10. Pranevicius, H. *Formal specification and analysis of Computer Net Protocols: aggregate method*. Kaunas, Lithuania: Technologija, 2003. 273 p.
11. Uhrmacher, A.M. Dynamic Structures in Modelling and Simulation – A Reflective Approach. In: *ACM Transactions on Modelling and Simulation*, Vol.11, No2, April 2001, pp. 206-232.
12. Pranevičius, H., Makakas, D., Paulauskaitė-Tarasevičienė, A. Multi-Agent Systems Formalization Using Formal Method DPLA. In: *The 21st annual European Simulation and Modelling Conference. St. Julian, Malta, October 22-24, 2007.* Malta, 2007, pp.427-431.
13. Potter, B., Sinclair, J., Till, D. *An Introduction to Formal Specification and Z.* Trowbridge, UK: Prentice Hall, 1996. 434 p.