On the Modeling Timing Behavior of the System with UML(VR)

Leszek Kotulski[1] and Dariusz Dymek[2]

[1] Department of Automatics, AGH University of Science and Technology
al. Mickiewicza 30, 30 059 Krakow, Poland
[2] Department of Computer Science Cracow University of Economics,
31-510 Krakow, Poland
kotulski@agh.edu.pl, eidymek@cyf-kr.edu.pl

Abstract. UML notation is assumed to be independent from any software modeling methodology. The existing methodologies support the creation of the final system model, but they do not care about the formal documentation of the reasoning process; the associations between the elements belonging to different types of UML diagrams are remembered either as informal documentation outside the UML model or are forgotten. Described in the paper Vertical Relations try to fill this gap, and allow to look at the use of *timing diagrams* from the new, more complex, perspective. Usefulness of Virtual Relations in evaluation of the timing properties of the Data Warehouse Reporting systems is presented.

1 Introduction

Unified Modeling Language (UML) [1] is an open standard controlled by the Object Management Group (OMG). UML is a family of graphical notation backed by single meta-model. It can be used for describing and designing software systems, in particular those using object-oriented paradigm.

In actual version of the UML standard (ver. 2.0) there are 13 types of diagrams, with precisely defined semantics [2]. Variety of diagram types allows us to describe different aspects of designed system, in particular:

- the ***use case diagram*** shows interactions of users or other software system,
- the software structure is dealt with the ***class diagram***, the configuration of instances of classes is shown on the ***object diagram***, the ***package diagram*** represent the compile-time structure of classes, ***composite structure diagram*** dealing with runtime decomposition of a class (such as a federate),
- the ***activity diagram*** convey the procedural and parallel behavior of classes, the ***state machine diagram*** shows how events change the interior states of an object,
- to align interactions between objects the ***sequence diagram*** is used, the ***communication diagram*** is used for emphasizing the links to be used by interactions, the ***timing diagram*** is used to cope with the timing aspects of interactions,
- the ***deployment diagram*** shows the structure of running system.

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 386–395, 2008.

In general, UML standard allows to cope with dynamical description of system beyond the semantic level. UML enables us to describe, how a system and its components interact externally as well as internally.

UML as a tool became a base for some software development methodologies like RUP (Rational Unified Processes) [3] or ICONIX [4]. UML bases on such fundamental concept like object-oriented paradigm or distributed and parallel programming but is independent from those methodologies. This fact gives UML some advantages; especially it can be treated as a universal tool for many purposes.

In general, software development methodologies based on UML are the sequences of informal recommendations how to, step by step, design a software system using different kind of UML diagrams. Final result is expressed in UML and the whole designing process is informally documented.

Possibility of creation of various software development methodologies based on UML flows from the fact, that inside UML, formal dependencies among diagrams of different kinds are not defined. It leaves a blank for various methods of reasoning for software development methodologies.

In this paper we show that capability of establishing the formal relations among different kinds of UML diagrams gives some new advantages. Presented (in section 2) way of introduction of such relations is independent from any methodology and does not affect the UML structure and properties. We named this type of relation the Vertical Relation to distinguish it from relations between elements of every single kind of diagrams which we called the Horizontal Relation. The proposed approach is an extension of UML and we named it UML(VR) to emphasizing the existence of additional relations. The consistency of the relations among different kinds of UML diagrams is maintained on the base of graph theory.

The introduction of UML(VR) allow us to suggest the application of the ***timing diagrams*** to describe the timing behavior of the Actors appearing in the ***use case diagrams***. In section 3 an example of such a solution in case of the Reporting System based on the Data Warehouse concept is presented. Moreover, having defined the Vertical Relation we are able to use ***timing diagrams*** associated with the ***use case diagrams*** to generate the ***timing diagrams*** associated with elements of the ***class***, the ***object*** and the ***deployment diagrams*** (see section 4), what can be useful in refactoring decisions.

2 UML(VR) Concept

UML itself defines the relation between elements from the given kind of diagrams or among diagrams from the same class. Generally, UML does not formally define the relation between various kinds of diagrams. Version 2.0 introduces <<trace>> and <<refine>> stereotypes for specifying models elements that represents the same concept in different models [2], but does not extends their use at metamodel level. This property allows using different kinds of reasoning methods for development methodologies and is one of advantages of the UML. But lack of the formal linkage among elements from different kinds of diagrams can cause loosing some information during software system designing, e.g. it's hard to find the connections between users' requirements and servicing them software components.

The problem of considering both horizontal and vertical consistency of UML model has been already pointed out a few years ago [5], but in practice those investigations has been concentrated on the horizontal consistency.

Fortunately, the UML diagrams can be expressed as EDG graphs using XMI standard [7]. During the process of software system designing we can translate each UML diagrams into a form of graph and create a Graph Repository, which will gather the information from every phase of designing process. It gives us a possibility to take advantage of graph grammar to trace the software system designing process, treating this process as a sequence of graphs transformations. We are able to participate in the designing process and simultaneously modify the Graph Repository. In [8] it was proved that, with the help of aedNLC graph transformation system [9], we can control the generation of such a Graph Repository with $O(n^2)$ computational complexity. This solution enables us to establish the formal linkage between elements from different kinds of UML diagrams as the Vertical Relation. To illustrate the capability of Vertical Relation we present below one of its exemplifications called Accomplish Relation (AR) [10], [11].

In the Graph Repository we can distinguish various layers (relevant to UML diagrams): the ***use case*** layer (UL), the ***sequence*** layer (SL), the ***class*** layer (CL) (divided onto the class body layer (CBL) and the class method layer (CML)), the ***object*** layer[1] (OL) (divided onto the object body layer (OBL) and the object method layer (OML)), the ***timing*** layer (TML) and the hardware layer (HL).

For any G, representing a subgraph of the graph repository R, the notation $G|_{XL}$ means the graph, with the nodes belonging to the XL layer (where XL stands for any UML type of diagram) and the edges induced from the connections inside R. For example, $R|_{UL\cup OL}$ means the graph with all the nodes (n_set $(R|_{UL\cup OL})$) representing user requirements and all the objects, servicing these requirements, with the edges (e_set$(R|_{UL\cup OL})$) representing both horizontal and vertical relation inside the graph repository.

Now we can present a definition of Accomplish Relation function:

AR:(Node,Layer) $\rightarrow$ AR(Node,Layer) $\subset$ n_set$(R|_{Layer})$ is the function where:
Node $\in$ n_set $(R|_{XL})$: XL $\in$ {UL, CBL, CML, OBL, OML,HL}
Layer $\in$ { UL, CBL, CML, OBL, SL, OML,TML, HL}, Layer $\neq$ XL

AR(Node,Layer) is a subset of nodes from n_set$(R|_{Layer})$, which stay in relationship of the following type: "support service" or "is used to" with given Node, based on role performed in the system structure. For better understanding, let's consider an example:

- for any user requirement r$\in$ n_set (R|UL), AR(r,OBL) returns a set of objects which supports this requirement service,
- for any object o$\in$ n_set (R|OBL), AR(o,UL) returns a set of requirements that are supported by any of its methods,
- for any object o$\in$ n_set (R|OBL), AR(o,HL) returns a set consists of the computing (hardware) node, in which given object is allocated,
- for any object x$\in$ n_set (R|UL$\cup$CBL$\cup$OBL$\cup$SL$\cup$HL), AR(x,TML) returns a set consists of the ***timing diagram*** describing the timing properties of its behavior,

[1] Packages introduce some sub-layers structure inside this layer.

– for any class c∈ n_set (R|CBL), AR(c,UL) returns a set of requirements that are supported by any of its method,

The above relations are maintained by the repository graph structure, so there are no complexity problems with their evaluation. Moreover, the graph repository is able to trace any software or requirement modification, so these relations are dynamically changing during the system life time.

In the next section the way of using the AR function in practice is presented.

3 Association Timing Diagrams for Use Case Actors

One of the most interesting type of diagrams introduced in UML 2.0 standard are the ***timing diagrams***. They are used to show interactions when a primary purpose of the diagrams is to reason about the time. Their properties became exhibited in many areas; one of the most interesting was, presented by Bunker, the solution of problem protocol compliance verification [11]. ***Timing diagrams*** focus on conditions changing within and among lifelines along a linear time axis. They describe behavior of both individual classifiers and interactions of classifiers, focusing attention on time of occurrence of events causing changes in the modeled conditions of the lifelines [2]. The classifier is defined in UML 2.0 standard as: "A collection of instances that have something in common. A classifier has the features that characterize its instances". Classifiers include interfaces, classes, data types, and components [2]. The introduction of ***timing diagrams*** is illustrated in OMG documents only by its application to the ***sequence diagram***.

Remembering (as some kind of Vertical Relations) the influence of existence elements of one type of diagram (e.g. ***use case diagram***) onto creation of elements of another type of diagrams (e.g. ***sequence*** or ***class diagrams***), during the modeling process, creates new possibility of using ***timing diagrams***.

We suggest using the ***timing diagrams*** for describing how Actors activity will change during the system exploitation. Let us notice that the system overload is caused if at least one, from mentioned below, event will follow:

– two or more processes that consume most of the computing system resources will start at the same time,
– some process will be activated by great population of users.

Information about the possible Actors (defined in the ***use case*** level) schedule can be remembered by association of a ***timing diagram*** with each of them. However, this information will be useful only if we are able to translate it into the ***timing diagrams***, describing the structure of the other elements of the software system (i.e. ***class***, ***object*** and ***deployment diagrams***). The VR creates such a possibility.

In [13] we consider a typical reporting systems. Every business organization during its activity generates many single reports. Some of them are created for managers and executives for an internal use only; others are created for external organizations, which are entitled to monitoring state and activity of given organization. For example, in Poland commercial banks have to generate obligatory reports inter alia to the National Bank of Poland (WEBIS reports), the Ministry of Finance (MF reports) and the

Warsaw Stock Exchange (SAB reports)[2]. In all external reports, it is few hundreds of single sheets with thousands of single data. In general, these reports base on almost the same kind of source data, but external requirements on format and contents causes that different software tools (based on different algorithms) are needed. These reports have the periodical character – depending on demands, a given report must be drown up every day, week, decade, month, quarter, half of the year or year, base on data from the end of the corresponding day.

Let's assume that we have the Reporting Data Mart system based on Data Warehouse. To simplify the example we skip the organization of the Extraction Transformation and Loading (ETL) processes and assume that all necessary information are maintained by the Data Warehouse Repository. It's ease to realize that for different Data Marts the set of used DW processes can be different. Analyzing the information content of reports we can divide them into a few categories, based on kind of source data and the way of their processing. Each of those categories, regardless of periodical character, is generated by different processes. Their results are integrated on the level of the user interface depending on period and external organization. The schema of data flow for Reporting Data Mart is presented in Fig 1.

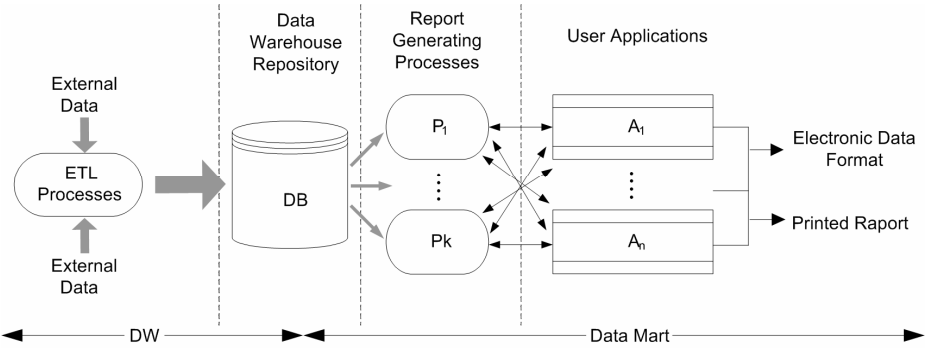

Fig. 1. Schema of Reporting Data Mart

Each User Application represents functionality associated with the single period and with the single type of obligatory reports. Because of that, we can treat these applications as user requirements, defining Data Mart functionality.

As we mention above, reports have the periodical character. It means that processes associated with these reports category have also the periodical character. They are executed only in the given period of time. This period is strictly connected with the organizational process of drawing up the given type of reports. Let us notice that the obligatory reports for the National Bank of Poland must fulfill many control rules, before they can be send out. In practice, it means that those reports are not generated in a single execution of the proper software process. Instead of this, we have the organizational process which can progress even a few days, during which the software process is executed many times after each data correction. Because of that, if we

[2] Structure and information contents of those reports are based in international standards so the same situation we can meet in other countries.

analyzing the time of availability of system functionality connected with those reports, we must take into account the larger time of the readiness of the hardware environment than in the case of the single process execution.

For the purpose of this example we can take a simple Reporting Data Mart with functionality restricted to only three reports categories: weekly, decadal and monthly. We assume that processes associated with weekly, decadal and monthly reports generation are started appropriately 2,3 or 4 days before of the reports delivery time.

The second type of the reports are ad hoc reports generated by consultants or verification of the hypothesis prepared by them. The mentioned activities are represented at ***use case diagram*** presented in Fig. 2.

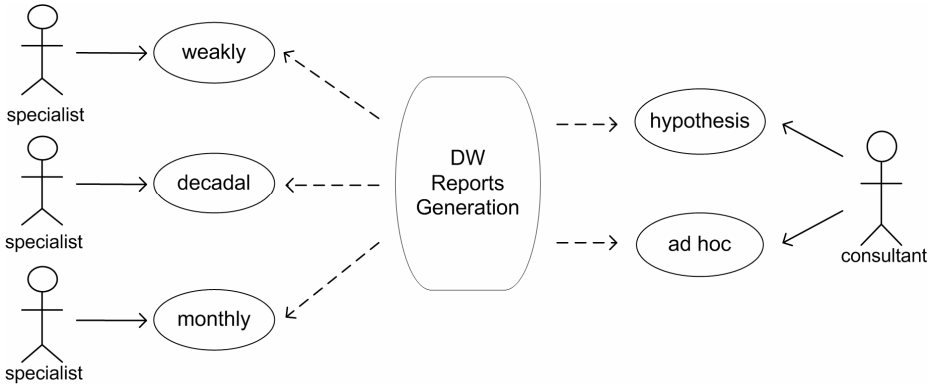

Fig. 2. Schema for Reports Generation activities

To estimate the system workload first we have to estimate the external usage of each system function. Each Actor artifact represents a group of users with a similar kind of behavior. Only Consultant Actor uses more that one systems function (ad hoc report generation and hypothesis verification). Thus we have to create five ***timing diagrams*** to express user behavior timing characteristics [13].

Three first ***timing diagrams***, representing the weekly, decadal and monthly reports generation processes activity, are presented in Fig. 3, where the number of active processes of a given type is either 0 or 1. The example of a Consultant population activity with respect to the ad hoc reports generation and the hypothesis verification are presented in Fig. 4 (the Y axis is scaled to 1:10 in comparison to Fig. 3), basing on assumption that the ad hoc reports are generated during worktime, and the hypothesis verification is made in the background.

Table 1. Process overloading

weakly report	30%
decadal report	30%
monthly report	30%
ad hoc report	1,5%
hypothesis verification	4,5%

If we are able to estimate the overload made by a single Actor request (the way of such estimation will be considered in the next section), we can evaluate the total system workload. For the purpose of this example assumed estimation is presented in Table 1.

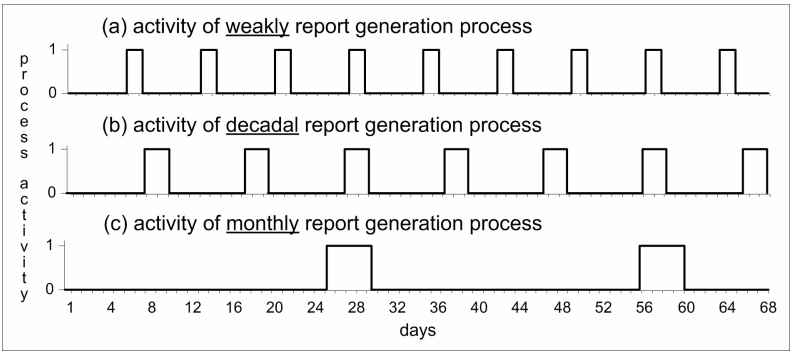

Fig. 3. Timing diagrams for periodical reports generation

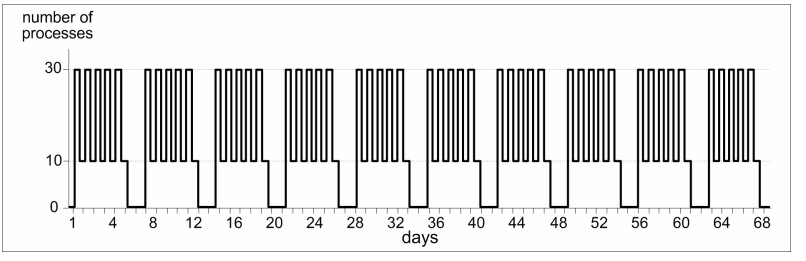

Fig. 4. Timing diagrams for Consultant activities

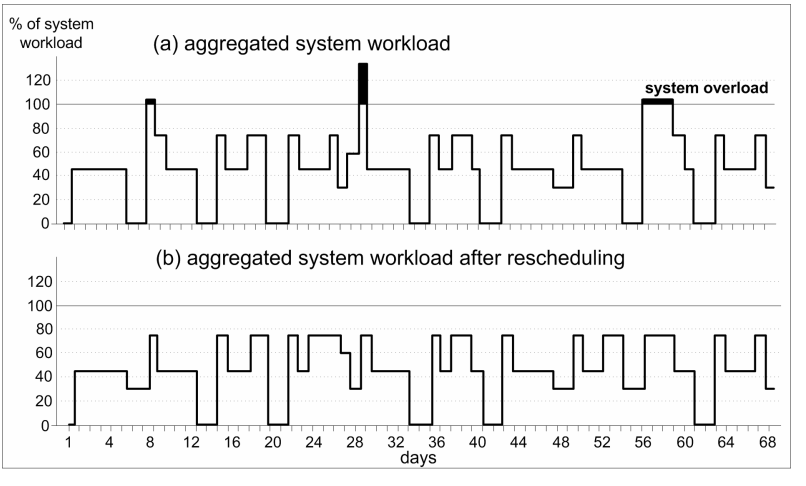

Fig. 5. System workload before (a) and after (b) evaluation

Thus (after a simple calculation) we can generate ***timing diagram*** representing the final system overloading as presented in Fig. 5a.

We can observe that user demand exceeds computing power of the system at 9-th, 30-th and from 57-th to 60-th day of system observation. Fortunately, data for

monthly and decadal reports generation usually a prepared by ETL process a few days earlier so we can start: decadal reports evaluation on 7-th and 29-th day, monthly reports on 25-th and 54-th days. Fig. 5b represents the overloading evaluation in such a case. The improvement of these processes effectiveness made by the distribution of some processes will be considered in the next section.

4 System Workload Estimation

The solution presented in the previous section bases on the assumption that we are able to estimate the workload of the computing system caused by an Actor request. In such a system as Data Warehouse, where evaluations of the same requests are repeated, such estimation can be made by the observation of the real system. However, it seams to be desirable to consider the influence of the information gathered in the ***timing diagrams*** (describing Actors timing behavior) on the final model of the developed software system.

In all methodologies using UML the ***use case diagrams*** (and ***class diagrams*** – for illustration of Domain Model) are the first diagrams generated during the system modeling. Here, we assume that ***timing diagrams*** associated with Actors activities are generated at the ***use case*** level to express the time relations among the elements of system structure associated with the periodical character of the system functions. The vertical relation AR, introduced in section 2, allows us to designate for each Actor's request r:

- the set of classes modeling the algorithms used during its service (AR(r,CBL)),
- the set of object that are responsible for the servicing of the request r (AR(r,OBL),
- the deployment of the mentioned in the previous point objects ((AR(o,DL)).

Thus we are able to estimate the workload of the software and the hardware components in the following way.

Let, for each r∈ n_set (Rl_{UL}), TM(r,t) represents ***timing diagram*** associated with ***r*** (more formally TM(r,t)=AR(r,TML)(t)). Having defined TM for requirements we can calculate it for methods, class, objects and hardware nodes.

For any m∈ n_set (Rl_{cmL}) $$TM(m,t) = \bigcup_{r \in AR(m,UL)} TM(r,t)$$

For any c∈ n_set (Rl_{CBL}) $$TM(c,t) = \bigcup_{r \in AR(c,UL)} TM(r,t)$$

For any o∈ n_set (Rl_{OBL}) $$TM(o,t) = \bigcup_{r \in AR(o,UL)} TM(r,t)$$

For any h∈ n_set (Rl_{HL}) $$TM(h,t) = \bigcup_{o \in AR(h,OBL)} TM(o,t)$$

where ∪ means the logical sum.

Timing diagrams generated for methods and classes helps us to better understanding of the modeled system structure and can be very useful in finding the system elements that should be refactored [14].

Timing diagram generated for Hardware Layer gives us information about the time of the hardware nodes activity, triggered by the execution of processes corresponding with objects allocated at it.

Let's notice that ***timing diagrams*** generated for the object can be used to estimate the level of utilize of the hardware equipment.

Let's assume that:

- we are able to estimate the (average, periodical) performance of the object components (described as *per(o)*); this estimation should be associated with the computational complexity of algorithms used inside the object.
- we know the computing power of the hardware nodes (described as *cp(h)*)

Then the function

$$EF(h,t) = \frac{\sum_{o \in AR(h,OBL)} (TRA(o,t) * per(o))}{cp(h)}$$

shows us the efficiency of the hardware nodes utilization in time. It can be used to indicate the periods of time in which the hardware equipment is almost not used or is very close to overloading. Brief analysis of presented function shows us that we have three ways of influence on its value: (1) we can reschedule the user requirements by changing business processes, (2) we can decrease performance demanded by the object's processes by rewriting software modules or (3) we can increase the hardware computing power.

5 Conclusions

The recent release of UML 2.0 has corrected a lot of design difficulties encountered in the 1.x revision. One of the new introduced capabilities is the possibility of characterization of the timing behavior for some components of the modeled system (with help of ***timing diagrams***). Unfortunately still actual is Engel's observation that a general consistency of UML model is still missing [7].

In the paper the idea of the formal remembering (as a kind of vertical relations) the associations between elements belonging to the different kinds of the UML diagrams was presented. Those associations appear during the reasoning process, while system modeling. However, this formal approach has a specific context; it means that the mentioned associations are remembered as a graph structures (equivalent to the UML Interchange standard [15]), so their maintenance and/or evaluation is possible with help graph transformation. In this sense this approach differs from other formal approaches supporting UML modeling with such formalisms as SCP [16] or B language [17].

Based on this idea, an application of *timing diagrams* as a tool for description of Actors timing behavior was shown. The capability of the automatic generation of the *timing diagrams* associated with objects and classes points out the part of the system that should be consider for possible refactoring. It is all the more important that the refactoring techniques in general are based on the system developer intuition (who discovers "bad smells" part of program [14]).

Presented UML(VR) concept seems to be a very promising approach. It can be used for different purpose in the development of the software system. The using of the

AR functions, which is an exemplification of the Vertical Relation, has been also studied by authors in such an area as the test generation [18].

References

1. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley Longman Ltd, Amsterdam (1999)
2. Unified Modeling Language, OMG v 2.1.2., http://www.omg.org
3. IBM Rational Unified Process, http://www-306.ibm.com/software/rational/
4. Rozenberg, D., Scott, K.: Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example. Addison-Wesley, Reading (2001)
5. Kuźniarz, L., Reggio, G., Sourrooille, J., Huzar, Z.: Workshop on Consistency in UML-based Software Development, http://www.ipd.bth.se/uml2002/RR-2002-06.pdf
6. Sourrouille, J., Caplat, G.: A Pragmatic View about Consistency of UML Models. In: Workshop on Consistency Problems in UML-based Software Development II, San Francisco (2003)
7. Engels, G., Groenewegen, L.: Object-Oriented modeling: A road map. In: Finkelstein, A. (ed.) Future of Software Engineering 2000, pp. 105–116. ACM Press, New York (2000)
8. Kotulski, L.: Nested Software Structure Maintained by aedNLC graph grammar. In: Proceedings of the 24th IASTED International Multi-Conference Software Engineering, Innsbruck, Austria, pp. 335–339 (2006)
9. Kotulski L.: Model wspomagania generacji oprogramowania w środowisku rozproszonym za pomocą gramatyk grafowych. Wydawnictwo Uniwersytetu Jagiellońskiego, Kraków (2000) ISBN 83-233-1391-1
10. Dymek, D., Kotulski, L.: On the hierarchical composition of the risk management evaluation in computer information systems. In: The Second International Conference DepCoS - RELCOMEX 2007, Szklarska Poreba, Poland, pp. 35–42 (2007)
11. Dymek, D., Kotulski, L.: Evaluation of Risk Attributes Driven by Periodically Changing System Functionality. Transaction on Engineering, Computing and Technology 16, 315–320 (2006)
12. Bunker, A., Gopalakrishnan, G., Mckee, S.A.: Formal hardware specification languages for protocol compliance verification. ACM Transactions on Design Automation of Electronic Systems 9(1), 1–32 (2004)
13. Kotulski, L., Dymek, D.: On the load balancing of Business Intelligence Reporting Systems. In: Proceedings of the AIS SIGSAND European Symposium on Systems Analysis and Design, University of Gdansk, Poland, pp. 121–125 (2007)
14. Flower., M., Beck, K., Brant, J., Opdyke, W.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co. Inc., Amsterdam (2000)
15. UML Diagram Interchange, OMG, version 1.0, http://www.omg.org/technology/documents/modeling_spec_catalog
16. Engels, G., Küster, J.M., Heckel, R., Groenewegen, L.: A methodology for specifying and analyzing consistency of object-oriented behavioral models. In: The 8th European Software Engineering Conference held jointly with ESEC/FSE-9, pp. 186–195. ACM, New York (2001)
17. Snook, C., Butler, M.: UML-B: Formal modeling and design aided by UML. ACM Transaction on Software Engineering Methodology 15(1), 92–122 (2006)
18. Dymek, D., Kotulski, L.: Using UML(VR) for supporting the automated test data generation. In: The Third International Conference DepCoS - RELCOMEX 2008, Szklarska Poreba, Poland (2008)