World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:11, 2008

Evaluation of Risk Attributes Driven by Periodically Changing System Functionality

Dariusz Dymek, and Leszek Kotulski

Open Science Index, Computer and Information Engineering Vol:2, No:11, 2008 publications.waset.org/13407.pdf

***Abstract*—**Modeling of the distributed systems allows us to represent the whole its functionality. The working system instance rarely fulfils the whole functionality represented by model; usually some parts of this functionality should be accessible periodically. The reporting system based on the Data Warehouse concept seams to be an intuitive example of the system that some of its functionality is required only from time to time. Analyzing an enterprise risk associated with the periodical change of the system functionality, we should consider not only the inaccessibility of the components (object) but also their functions (methods), and the impact of such a situation on the system functionality from the business point of view. In the paper we suggest that the risk attributes should be estimated from risk attributes specified at the requirements level (Use Case in the UML model) on the base of the information about the structure of the model (presented at other levels of the UML model). We argue that it is desirable to consider the influence of periodical changes in requirements on the enterprise risk estimation. Finally, the proposition of such a solution basing on the UML system model is presented.

***Keywords*—**Risk assessing, software maintenance, UML, graph grammars.

I. Introduction

NOWADAYS, computer systems are characterized by a very complicated structure which consists of many subsystems, many single software applications and hardware components. Their complexity grows in the continuous process of changes forced by technology development and increasing (changing) users' requirements. For the effective management of a computer systems evolution, a crucial problem is how to control the system structure, the characteristic of its components and the relations among them. This necessity is reflected in many models of computer systems which are focusing on different aspects of the properties of systems. One of the most important aspects considered here is the quality requirements control. From the engineering point of view "quality" is too imprecise, so in theory and practice we use rather "quality attributes", which are defined as measurable (or observable) properties and are divided into a few categories such as: performance, availability, security, reliability, stability or fault-tolerance. Such an approach is supported in many methods ([1],[2],[3]). Moreover, when we want to assure the fulfillment of quality attributes requirements in newly developed systems, it's very important to have formal tools, which allow as to manage the evolution of the system and to maintain the achieved level of quality attributes ([4],[5)].

The fact that these methods represent only the engineers' point of view is their weakness. Each system component is characterized by values of many quality attributes that reflect on its technical properties, but there are no information about influence of the given software component on the effective realization of the whole system functionality.

One of the most important properties of complex systems is the fact that during regular execution not the whole system functionality is used at the same time. Some system functions are used only in given period of time, depending on users' demands. It means that the system structure change in time even while the usual exploitation. So we should consider not only dynamical evolution of the system structure while its maintenance (long period) but also the periodical evolution of the system during its normal live time. Such a system behavior is typically characteristic for Data Marts systems created as a part of Decision Support Systems (see chapter 2).

The necessity of considering time dependencies has been approved by OMG that incorporate Timing Diagrams into UML 2.0 standard [10]. UML dynamic notation supports both the users' requirements and other details of the system solutions (software and hardware). The information presented at different levels (e.g. in Use Case diagrams and Class diagrams) are not related in a formal way. The introduction of the graph repository [4] allows us to introduce vertical relations among the elements belonging to the different types of UML diagrams (in section 3). These vertical relations we use to calculate the Importance Ratio, that exemplifying the importance of system's components from the business point of view, and we show how this ratio can be used to merge both engineering and business point of view (in section 4). Finally we will show, how the graph repository concept and the introduced of the business measures allow us to project the timing relations (represented by Timing Diagrams for Use Case components) to the software and hardware components (described at the Deployment Diagrams).

II. Time Dependency in the Complex System

Every business organization during its activity generates many single reports. Some of them are created for managers

Manuscript received October 9, 2001.

Dariusz Dymek is with the Institute of Computer Science, Cracow University of Economy, Rakowicka 27, 31-510 Kraków, Poland (e-mail: eidymek@cyf-kr.edu.pl).

Leszek Kotulski is with the Department of Automatics, AGH University of Sciencce and Technology, Al. Mickiewicza 30, 30 059 Kraków, Poland (e-mail: kotulski@agh.edu.pl).

Open Science Index, Computer and Information Engineering Vol:2, No:11, 2008 publications.waset.org/13407.pdf

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)[1]. 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 periodical character – depending of demands 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 consider an example of a Reporting Data Mart system based on Data Warehouse. The relations among DMs and DW are presented on generic schema presented in Fig. 1.

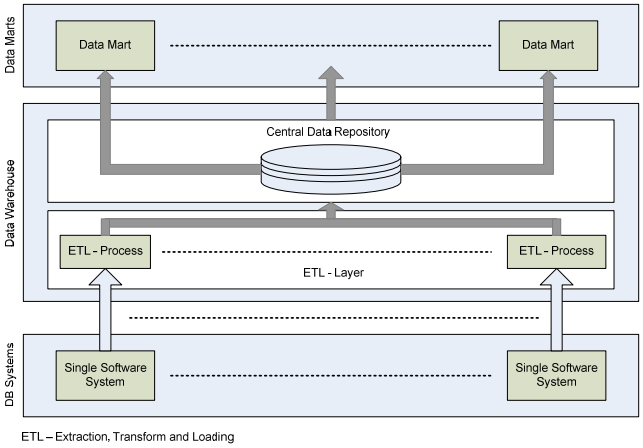

Fig. 1 Relations among Data Marts and Data Warehouse

It can see, the Extraction Transformation and Loading process (ETL process) to a single Data Mart is a result of many single Data Warehouse processes. 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. 2.

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

As we mention above, reports have 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.

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

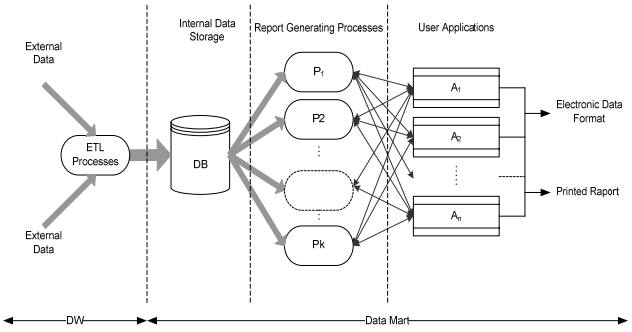

Fig. 2 Schema of Reporting Data Mart

This period is strictly connected with 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 organizational process which can progress even few days, during which the software process is executed many times after the correction of data. So, if we 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.

As an example we can take a simple Reporting Data Mart which functionality is restricted to only three reports categories: weekly, decadal and monthly. For simplification we can assume that we have only one ETL process, common for all the categories of reports. So we have four processes. Organizational processes for those reports categories have the following time characteristic[2]:

- weekly reports have to be ready before Thursday,
- decadal reports have to be ready in five workdays,
- monthly reports have to be ready till 20 day of the next month.

The characteristic of the ETL process is more complicated and depends on its functionality. For purpose our example, we assume that subprocesses associated with weekly, decadal and monthly reports generation are started appropriately 2, 3 or 4 days before the processes of the reports generation. We also assume that the hardware supporting one, two or three of these subprocesses creates the weak, middle or strong workload of the system. Now, we can express the ETL process activity using robust notation of the timing diagrams as presented in the Fig. 3.

Timing diagrams are one of the new artifacts added to UML 2.0. They are used to explore the behaviors of one or more objects throughout a given period of time. The concise notation seems to be very convenient for the presentation of the timing properties of the reporting activities as presented in the Fig. 4. More complex situation is with the loading process, that supports extraction data for one, two or three reports areas, so can be in idle or weak, middle or strong workload states.

[2] Those characteristics (for decadal and monthly reports) are taken from National Bank requirements.

Open Science Index, Computer and Information Engineering Vol:2, No:11, 2008 publications.waset.org/13407.pdf

We must however note that the presented timing behavior of this process represents only the expectation of the system designer and in not associated with the generated system structure. As plain as a pikestaff it is that these dependences will change when the loading process will be realized on several computers. We have a tool inside UML to express the final system architecture (deployment diagrams) but there is no influence of these diagrams evolution on the timing diagram.

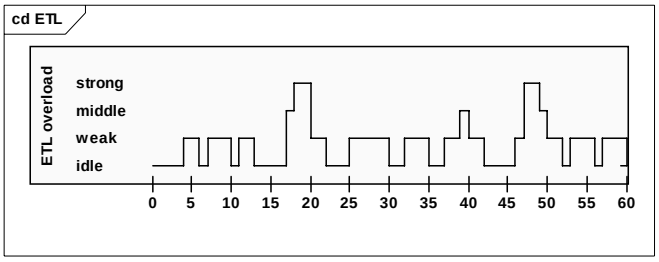

Fig. 3 Aggregated workload created by ETL process

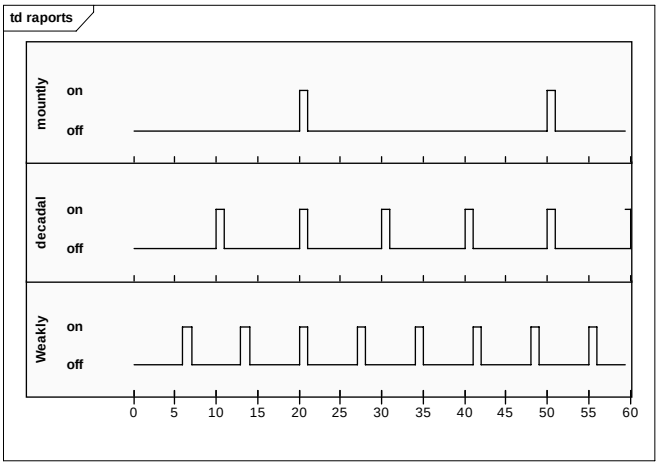

Fig. 4 Time schedule of ETL process associated with reports generating processes

III. Graph Repository and the Relations among the UML's Diagrams

The ULM notation [10] is a recent approach to strengthen the effort of designing an object-oriented modeling language where all main issues of system analysis and design are taken into account. Its visual notation, express (at different abstraction level) the association among requirements, software components and the static associations software components with the hardware ones. The presentation of the separate diagrams causes the loss of the significant part of the project's information such as the influence of the users' requirements on the behavior of a given part of the final system. On the other hand, this problem, as strongly related to the graph isomorphism problem, is unsolved in the polynomial time in a general case.

Fortunately, we can accompany the development of the graph repository to the project's process and simultaneously maintain it. In [9] it was proved that with help of aedNLC graph transformation system [8] we can control the generation of such a graph repository with $O(n^2)$ computational complexity. In the presented solution [9] we can:

- represent deployment of the final objects to the proper computing nodes,
- show nested software structure (introduced by packages),
- trace, inside whose class (in case of class inheritance) the given objects method has been defined.

Finally in the same way we can extend this representation by the:

- association of the object's method with the proper edges in the Interaction Diagrams,
- associate graph representing the Interaction Diagram with the given Use Case activity.

In the graph repository we can to distinguish following layers:

- the Use Case layer (UL),
- the Interaction Diagram layer (IL),
- the Class layer (CL), divided onto the class body layer (CBL) and the Class Method layer (CML),
- the Object layer[3] (OL), divided onto the Object Body layer (OBL) and the Object Method layer (OML),
- the Hardware layer (HL).

In the paper, we are especially interested in the relations among these layers. For example when we consider the user requirement, we can designate either objects that serve this requirement, or directly the object methods; next we can designate classes responsible for the definition structures and algorithms supporting this requirement service.

The relations among elements of the same layer (represented by graph edges) will be called the horizontal relation and the relations among elements belonging to different layers will be called the vertical relation. 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 layer) and the edges induced from the connections inside R. For example, for the full graph R, $R|_{UL\cup OL}$ 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.

Considering the system structure we can distinguish two different ways of viewing it. The first one corresponds with System Architecture presented in [1]. It shows the relation inside Use Case and Class diagrams and among their components and is called the *logical level*. The second one shows the relation inside Use Case and Deployment diagrams and among their components and is called *the execution level*. *The execution level* can be described as $R|_{UL\cup OL\cup HL}$ and the *logical level* can be described as $R|_{UL\cup CL}$.

To manage the vertical relations we introduce Accomplish Relationship (AR) function:

$AR : (Node, Layer) \rightarrow AR(Node,Layer) \subset n\_set(R|_{Layer})$

where:

$Node \in n\_set\ (R|_{XL})$: $XL \in \{UL, CBL, CML, OBL, OML, HL\}$

$Layer \in \{UL, CBL, CML, OBL, OML, HL\}$, $Layer \neq XL$

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

Open Science Index, Computer and Information Engineering Vol:2, No:11, 2008 publications.waset.org/13407.pdf

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∈ n_set ($R|_{UL}$), **AR(r,OBL)** returns a set of objects which support this requirement service,
- for any object o∈ n_set ($R|_{OBL}$), **AR(o,UL)** returns a set of requirements that are supported by any of its methods,
- for any object o∈ n_set ($R|_{OBL}$), **AR(o,HL)** returns a set of computing (hardware) nodes in which given object is allocated,
- 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 live time.

IV. Risk Evaluation

Analyzing large modern business systems, it's easy to realize that their functionality can be divided into several categories, depending on an influence of the given system's function in the achievement of business goals. Our goal is to create the hierarchy of these functions that reflects theirs business importance. As a criterion we can take a possible loss associated with the situation of temporary inaccessibility of the given function. Results of such analyze depends on the type of business organization and the period of inaccessibility time.

For example, in the case of banking, the most substantial functions are these connected with a customer support. Their unavailability, even in a short period of time, can leads to loss of clients, which in consequence can create a significant potential financial loss. On the other hand, the inaccessibility of some analytical or reporting functions even thought one day period in many cases creates the small financial loss, if any. In the presented approach, we don't need exactly calculate the financial loss. We must only to designate the ratio joining the satisfaction of the criterion with the given system's function. This ratio, called Importance Ratio for the given function **f** is described as **IR(f)**. Because all system's functions descent from previous defined users requirements, we can assume, that users of the system can ascribe the Important Ration Function for all requirements defined at the Use Case Layer (**IR(r)**).

Technically for simplify of the calculation and presentation we assume that the value of IR(r) (r∈ n_set ($R|_{UL}$) is unique for each system's function. This assumption implies the descending order relation in set of system's requirements ($IR(r_j)>IR(r_k)$ for j>k). For the same reasons we normalize IR(r) values to [0,1] interval.

Basing on IR defined at Use Case Layer, we can calculate IR values for the parameter belonging to other layers in the way described below.

In the graph repository representing the system model the nodes representing requirement are connected with nodes representing methods. So with the node representing a method m (m∈OML) we can bind a set of requests $\{r_i\}$ such that $r_i \in AR(m,UL)$. Let *n* be a number of all requirements in set n_set ($R|_{UL}$)

Now we can calculate Importance Ratio for any method m∈OML, using weighted average (WA) function, in the following way:

$$IR(M) = WA(AR(m,UL)) = \sum_{x_i \in AR(m,UL)} (pos(x_i) * IR(x_i))/n$$

where $pos(x_i)$ is position of x_i in the n_set ($R|_{UL}$) vector.

Using both normalization and weighted average, we calculate value of IR(m) in a more objective way, by reducing direct user impact on IR value; note that WA function, strengthens the values associated with requirements having higher priority in user ranking.

Having defined IR(m) for methods we can easily define IR for objects and classes:

for any $o \in OBL$, $IR(o) = A(\{IR(m_i) : i=1\ldots k\})$, $m_i \in AR(o,OML)$

for any $c \in CBL$, $IR(c) = A(\{IR(m_i) : i=1\ldots p\})$, $m_i \in AR(c,CML)$

where A means the arithmetical average.

For the computing (hardware) nodes we can define IR values as follows:

for any $h \in HL$, $IR(h) = S(\{IR(o_i) : i=1\ldots q\})$, $o_i \in AR(h,OBL)$

where S means the sum of value[4].

As a result we attribute each system component with IR value, which show the importance of this component from the business point of view. It allows us to designate the key-components of the system. In the previous section we have introduced the logical and execution levels. Analysis on the logical level, where we consider the system architecture, should be made in very early state of the development process, and should give us an essential information about severity of the single component in the future system. On the other hand, analyses on the execution level give us the same information about components of the running system.

The introduced scheme of the inheritance of the business importance ratio defined at the requirements level by the software systems components is applied to the current state of the system. For tracing and managing the system evolution we should introduce the third dimension of the components relation – time.

To express the time relations among the elements of system structure associated with periodical character of system functions (mentioned in section 2) we introduce the Time of Required Availability (TRA) function.

[4] Selection of functions for defining IR can depend on goals of the provided analysis or the kind of the system, e.g. for Hardware Layer in a real-time system function $IR(h) = \max\{IR(o_i): i=1\ldots p\}$ seems to be more suitable.

Open Science Index, Computer and Information Engineering Vol:2, No:11, 2008 publications.waset.org/13407.pdf

For any r∈ n_set ($R|_{UL}$), TRA(r,t) → {0,1}: TRA(r,t) = 1 when in the time t the system function corresponding with the requirement r is demanded by user to be active, and TRA(r,t) = 0 otherwise. Having defined TRA for requirements we can calculate it for objects and hardware nodes.

For any o∈ n_set ($R|_{OBL}$) $TRA(o,t) = \bigcup_{r \in AR(o,UL)} TRA(r,t)$

For any h∈ n_set ($R|_{HL}$) $TRA(h,t) = \bigcup_{o \in AR(h,OBL)} TRA(o,t)$

where ∪ means the logical sum.

Function TRA for Hardware Layer gives us information about the time of activity of the hardware nodes, triggered by execution of processes corresponding with objects allocated at it. To estimate the importance of given hardware node in whole system from the business point of view, we can extend IR for hardware nodes as follow:

$$IR(h,t) = \sum_{o \in AR(h,OL)} (TRA(o,t) * IR(o))$$

Knowing the value of IR in time we are able to indicate the crucial hardware nodes of the running system in the arbitrary period of time. This time characteristic can be used e.g. for choosing best time for the hardware maintenance activity such as reconfiguration, update or others.

Let's notice that function TRA can be used to estimate the level of utilize of hardware equipment. If we are able to estimate the (average, periodical) performance of the object components and the computing power of the hardware nodes (described adequately as *per(o)* and *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 utilizing hardware nodes in the 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. Detail analysis of presented function show us that we have three ways of influence on its value: (1) we can reschedule the user requirements, 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.

V. Conclusion

For any business organization the most important goal is keeping up with trends of the modern market in the dynamically changing environment. Role of the Information Technology is the support this goal by giving the organization right tools for the quick and effective fulfillment the existing and the possible users needs. From such a point of view, business usability can be much more valuable then a technical perfection. It doesn't mean that technical principles are not important, but they must take into account the business principles, too. Having this fact in the mind is one of the most important the distinguishing feature of a modern approach to IT in the business use.

Modern complex software system has not a static structure. Even during regular execution the structure of running system changes according to users' activity. We use UML notation to represent user requirements, the system software structure and the allocation its software components onto the hardware environment.

The introduction of the graph repository, which is built (under the control of the aedNLC grammar) during the designing of the system, allows us to introduce the (vertical) relations among the elements of these layers of abstraction.

For the Time of Required Availability function (TRA) defined at the requirements (Use Case) level, we are able to calculate (with help of vertical relations and other UML diagrams) the TRA for all objects in this system. Let us note that extended TRA changes not only when the user changes timing requirements of the system functionality, but also when the software or hardware structure changes. It gives us a tool for better understanding system behavior, so we are able to better utilize the system's recourses. Merge up with IR functions gives us a detail view of system structure in given period of time from users' points of view. It can be very useful in unexpected situation such as system breakdown, when we can use such information for planning system recovery policies.

The other possibility of using IR and TRA functions is the analyze of quality attributes. We can use them as a weight quotient in the process of estimation value of these attributes, which allow us to incorporate the time and business importance factor of any system components.

Proposed approach is the supplement of architectural based methods of analyzing the characteristic of software systems such as ATAM [8] or MAAP [2], particularly of analyzing the quality attributes, which is crucial in risk management. It allows us to connect in one method the engineering and the business points of view of the software system. The ways in which we introduce and use the IR function (describing the components importance) enable its utilization both in the development and the maintenance phases of system life cycle. It can be used to plan new system or during execution of existing one in a way to decrease the risk connected with failure of any system component.

References

[1] R. Allen, R. Douence and D. Garlan, "Specifying and Analyzing Dynamic Software Architectures", *Lecture Notes in Computer Science*, vol. 1382, pp. 21-36, 1998.

[2] J.C. Alberts and J.D. Audrey, "Mission Assurance Analysis Protocol (MAAP): Assessing Risk in Complex Environments", *Technical Report* CMU/SEI-2005-TN-032, 2005.

[3] R. Bahsoon and W. Emmerich "Evaluating Software Architectures: Development, Stability, and Evolution", *Proceedings of ACS/IEEE Int. Conf. on Computer Systems and Applications,* Tunis, Tunisia, July, 2003

[4] D. Dymek and L. Kotulski, "On Hierarchical Composition of the Risk Management Evaluation in the Computer Information Systems", *accepted for 25-th IASTED Multi-international Conference on Applied Informatics, Software Engineering,* February 13-15, Insbruck, 2007.

[5] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A Method for Analyzing the Properties of Software Architecture", *International Conference on Software Engineering*, pages 81-90, 1994.
[6] R. Kazman, M. Klein and P. Clements, "ATAM, Method for Architecture Evaluation", *Technical Report CMU/SEI-2000-TR-004*, 2000.
[7] M. Klain and R. Kazman, "Atribute-Based Architectural Styles", Technical *Report CMU/SEI-99-TR-022*, 1999.
[8] L. Kotulski, „Model wspomagania generacji oprogramowania w środowisku rozproszonym za pomocą gramatyk grafowych - DSc degree", *Wydawnictwo Uniwersytetu Jagiellońskiego*, Kraków, ISBN 83-233-1391-1, 2000.
[9] L. Kotulski, "Nested Software Structure Maintained by aedNLC graph grammar"' *Proceedings of the 24th IASTED International Multi-Conference Software Engineering*, February 14-16, pp 335-339, Innsbruck, Austria, 2006.
[10] OMG-Unified Modeling Language, v2.0. www.rational.com

Open Science Index, Computer and Information Engineering Vol:2, No:11, 2008 publications.waset.org/13407.pdf

Leszek Kotulski received:
- the M.S. degree in Computer Science from Institute of Computer Science, Jagiellonian University, Kraków, 1979,
- the Ph.D. degree in Computer Science from AGH University of Science and Technology, Krakow, 1984,
- DSc degree in Theoretical Computer Science from Wroclaw University of Technology, Wrocław, 2002.

He works as a Professor at AGH University of Science and Technology. His research interests include graph grammars, foundation of distributed computing, agents systems and software development methodology.
Prof. Kotulski is member of ACM.

Dariusz Dymek received:
- the M.S. degree in Mathematic from Institute of Mathematic, Jagiellonian University, Kraków, 1989
- the M.S. degree in Computer Science from Institute of Computer Science, Jagiellonian University, Kraków, 1991,
- the Ph.D. degree in Economics Science from Krakow University of Economy, Krakow, 2000.

He works as Professor Assistant in the Institute of Computer Science at the Krakow University of Economy. His research interests include project management, information systems, risk management, software quality management and software development methodology.