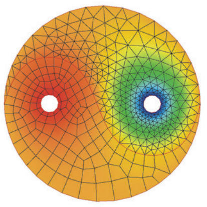
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE
Informatyka w Technologii Materiałów

Vol. 7, 2007, No. 3



ON THE SUPPORT UML DIAGRAMS UNDERSTANDING DURING THE SOFTWARE MAINTENANCE

LESZEK KOTULSKI[*], ARTUR BASIURA[#]

[*] Department of Automatics, AGH – University of Science and Technology, al. Mickiewicza 30, Kraków, Poland
[#] DRQ S.A., ul. Podwale 3, Kraków, Poland
Corresponding Author: kotulski@agh.edu.pl (L. Kotulski)

Abstract

UML diagrams are generally accepted technique of supporting computing modeling and maintenance systems that are independent from the domain supported by the created system. Final model describes the system with help of object oriented techniques such as class inheritance, packages or the final software deployment diagrams. The mentioned techniques introduce some hierarchy of the developed concepts. An UML model describing all aspects of a system is hard to fully visualized. Usually they contain thousands of elements and relations which are difficult to present in form that can be easy to understand. Unfortunately, we are not able to represent such structures with help of the 2-dimensions manner (monitor screens or paper sheets) so we present only some "flat" aspects of these structures; it is desirable that we should be able move from one "flat" visualization of the hierarchical structure to another one. In this paper, we formalize the term "a flat visualization of the hierarchical graph", and specify the synthesis and analysis operations, that allow us to move between different flat forms. Practical aspect of this proposition is discussed for UML deployment and class diagrams.

Key words: graph transformation, UML, image understending, software visualization

1. INTRODUCTION

The effective communication among computer system and a man seems to be a challenging problem There are developed several techniques supporting understanding of the documents (Baird Lopresti Davison & Pottenger, 2004) and images generated by human environment (Routledge, Bird & Goodchild, 2002; Ogiela & Tadeusiewicz, 2005; Tadeusiewicz & Ogiela, 2004 and 2007; OMG, 2005). There are two problems: how to collect all necessary information and how present it in the right form which can be easily understood by men.

In this paper we would like consider the second problem: perception of software system model. We consider problems appearing during the visualization (i.e. presentation of the data maintained by a computer), where a display a large quantity of information can exploit human capabilities for rapid recognition. The special problem appears during visualization of the nested (hierarchical) structures (Can Keskin & Vogelmann, 1998). In particular for large system diagrams or graph (representing such a structure) with hundreds of nodes we observe the following problems:

– the display of a graph structure on the 2D screen tends to be clustered and is difficult to understand,
– while viewing details of a layered diagrams it is difficult to preserve the mental map due lack of context information.

ISSN 1641-8581

Let us notice that the mentioned problem is one of the essential problems of software engineering. Hierarchical models are a natural model of software systems. Such a systems can be replaced by graphs, software entities like classes, methods, or packages can be represented by graph nodes. Relations among software entities like inheritance, method calls and attribute access can be represented by graph edges.

The comprehension, evaluation and improvement of the structure of large software systems requires view on different levels of abstractions: global view showing relations among (nested) packages, detailed view showing interaction of methods and attributes of the given class, and views including different level of detail, e.g. showing the methods and attributes of a class in their global context. Using a graph transformation mechanism we will look for one that both has enough descriptive power for the controlling generation and modification graphs representing hierarchical structure of a distributed system and, what is also important, made it with polynomial time of computational complexity (as for example aedNLC graph grammar described by Kotulski, 2000). The generated hierarchical graphs (Kotulski, 2005) are, however, difficult to exemplification, so in the paper we analyse relations occurring in the graph representing nested objects and we propose the notation of flat (2-dimension) graph. In such a representation the only one layer, from a few ones representing the abstraction of the model of nested system, is shown.

It is obvious that models help to understand a system behaviour at the right level of abstraction. The key feature of modelling language such as ULM is that it is multi-faceted, enabling multiple consistent view of the same problem, instead linear one. Thus, UML notation seems to be good reference point for presentation of problems appearing during visualization of the hierarchical software structure.

The scope of the paper is the following:
In chapter 2 an example of nested system structure which is introduced and described both by UML diagrams and as the hierarchical graph. In chapter 3 we formulate a formal background necessary for the definition the flat graph notation and for the introduction of two operations synthesis and analysis that modify the current flat graph in the context of the hierarchical graph.

Initially, we represent the hardware/software system using Deployment Diagrams; during the software maintenance phase we should however consider the another abstraction levels introduced in ULM – object diagrams and class diagrams (with the inheritance relation). The extension of the hierarchical graph structure in a way that enables maintaining the information represented by UML class diagrams and its visualization in the flat graph context is considered in chapter 4.

Finally, we discuss some practical aspects of the visualization of the knowledge covered inside the hierarchical graphs in the software engineering context.

2. NESTED STRUCTURE AND ITS REPRESENTATION

Object oriented techniques, group concepts (Kramer, Ng, Magee & Dulay, 2005; Kotulski Jurek & Moczurad, 1992) or an UML package (Booch Rumbaugh & Jacobson, 1999) introduces hierarchy of modules. Models help understand a system at the right level of abstraction. Modelling complex system has several general benefits. Some specific situations in which the modelling efforts is worthwhile include:

- better understanding the business or engineering situations at hand ("as-is" model) and to craft a better system("to-be" model)
- building and designating a system architecture
- creating visualizations of code and another forms of implementations.

On the other hand, when several analysts capture requirements of different stakeholders, it results in a set of overlapping and partly conflicting requirements models. Moeover, the UML meta-model is not consistent and correct (Fuentes, Quintana et all, 2003), so some form of formalization, for example the graph notation, should be introduced to make precise the notions of conflict and dependency between functional requirements expressed by different use cases (Taentzer, Fischer, Koch, Vole, 1999; Zhang, Zhang, Cao, 2001).

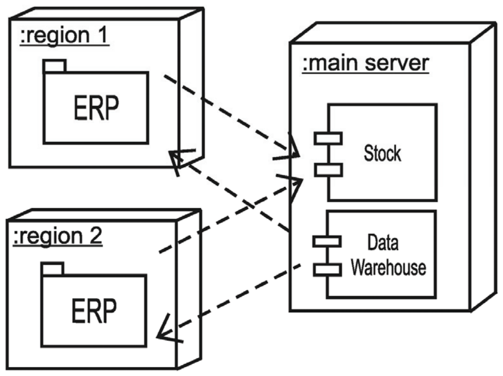


Fig. 1. UML Deployment diagram

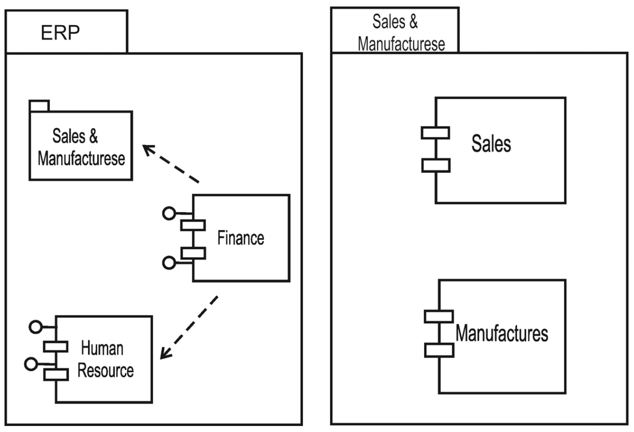


Fig. 2. UML Package diagram

In this chapter we consider relations introduced by ULM deployment diagrams. Let us consider a management system based on a Data Warehouse concept, which stores of data obtained form ERP systems (see figure 1). However, the ERP system consists from many cooperating subsystems (such as Sales&Manufactures, Finance and Human Resource packages – as in the figure 2), because of that someone may want to see direct associations among these subsystems and the Data Warehouse repository. This simple example shows that exist a necessity of introduction more than only two levels of abstraction (software and hardware) suggested by (Taentzer, 2003).

The representation with different levels of abstraction is difficult to visualization and understanding. For example Sales component is represent both as an object and as a part of Sales&Manufactures and ERP packages. When we will visualize Sales component at the Sales&Manufactures point of abstractions than the context its cooperation with ERP system is covered up. The separate visualization of the nested software components does not preserve the mental map due the lack of context information. On the other hand visualization at the same time the hierarchy of all nested components and associations among them creates diagram is too complex and unintuitive.

So in the paper we formalize notation which allow us to represent the hierarchical graph in such way that each component will be represented only in one abstraction level.

3. RELATIONS INSIDE THE HIERARCHICAL GRAPH

The hierarchy of nested packages can be intuitively developed under the control of a graph transformation system (Kotulski, 2005). The figure 3 shows the graph representation of the mentioned problem with one extension (we assume that Sales&Manufactures package are represented by two separate objects Sales and Manufactures) and one reduction (for simplicity only one ERP system has been represented). Some edges of the graph presented in the figure 3 are responsible for expressing the hierarchical relations.

Graph nodes used for describing of a distributed system can be labelled as follows: P - for a package component, E – for an export interface, I – for an import interface (stub), O – for an object instance component, N – for a computing node representation. All additional information about the graph node can be described by its attributes. Graphically, we expose only a component's name. The only correct labelling pattern[1] includes the following set of triples (E,b,P), (E,b,O), (P,c,I), (O,c,I), (N,d,P), (N,d,O), (O,op,P), (I,l,E), (N,n,N). Edge labels b, c, d, l and n are appropriately abbreviation for **b**elonging, **c**alls, **d**eploy, **e**ncapsulated, **l**inked and **n**ode interconnection. Let us note that two edges:

– an edge labeled by "op", represents relation the object is a part of the package,
– an edge labeled by "d", represents relation the object or the package is deployed in the computing node,

introduce hierarchical relation inside the graph.

Formally such a graph can be specified in a form:

Definition 1. An attributed direct node- and edge-labeled graph, EDG graph, over Σ and Γ is a quintuple

$$H = (V_H, D_H, \Sigma, \Gamma, \delta), \qquad (1)$$

where:

V_H – is a finite, non-empty set of graph nodes, to which unique indices are ascribed, showing the order within the set.
Σ – is a set of attributed node labels.
Γ – is a set of attributed edge labels
D_H – is a set of edges of the form (v, μ, w) where $w,v \in V$ and $\mu \in \Gamma$
$\delta: V_H \rightarrow \Sigma$ – is a function, which labels the nodes ■

For the simplicity, the nodes will be identified by indices, i.e. the elements of V set will be natural numbers from the range 0 to MAX. The node labels

[1] Term labeling pattern for edge (v,μ,w) means triple (δ(v) ,μ,δ(w)), where δ(v) and δ(w) returns labels of v, w node indices appropriately.

and edges labeling pattern has been introduced in the in the description following this definition.

In (Kotulski, 2005) it was shown that aedNLC graph transformation system is able effectively control the transformation UML structure to the hierarchical graph, so in this paper we will concentrate on the way of its visualization. The next few definitions allow us to formalize used informally term "flat representation".

Let $V=V_H|_{\{\underline{P},\underline{O},\underline{N}\}}$ denote restriction of set V_H to the nodes labeled by $\underline{P}$, $\underline{O}$ or $\underline{N}$ (i.e. $X\in V \Rightarrow \delta(X)\in\{\underline{O},\underline{P},\underline{N}\}$) then we can formally introduce a few relations of structural dependency in the graph H.

Definition 2. Let $X,Y\in V$ and $\delta(X),\delta(Y)\in\{\underline{O},\underline{P}\}$ then X is directly structural dependent from Y if $(X,\underline{op},Y)\in D_H$ what will be denoted as $X\prec Y$ ■

In the paper we will interested only in the graph for which this relation is acyclic and unique from right side.

Definition 3. Let $X,Y\in V$ then X is structural dependent from Y (denote as $X\prec^* Y$) if one of the following conditions is fulfilled:

$$X=Y \tag{1}$$

$$\exists n>1\ W_1,\dots,W_n\in V_H\colon W_1=X \text{ and } W_n=Y \text{ and } \forall k\ 1\le k<n\ W_k\prec W_{k+1} \ ■ \tag{2}$$

It is easy to prove that structural dependency is relation of the partial order.

Definition 4. For every $X,Y\in V$ we will say that they belongs to the same graph representation of the object (denoted as $X\approx Y$) if one of the following conditions is fulfilled:

$$X=Y \tag{1}$$

$$\exists Z\in V\colon X\prec Z \text{ and } Y\prec Z \ ■ \tag{2}$$

Theorem. Relation of the graph representation is equivalency relation.

Proof gets out of foundation that the given object or package instance can be directly nested in at most one package.

Let us return to the UML graph visualization and introduce a formal definition of the flat (2-dimension) graph G in which for any component is visualized only by one of its graph representation and associations between it and the graph representation of other objects is induced from the hierarchical graph.

Definition 5. For each X such that $\delta(X)\in\{\underline{O},\underline{P}\}$ its local representation we will call the set:

$$\mathrm{LocRep}(X)=\{\underline{Y\colon \delta(Y)=I \wedge (X,c,Y)\in D_H}\}\cup \{Y\colon \delta(Y)=\underline{E \wedge (Y,b,X)\in D_H}\}\cup\{X\} \ ■$$

Definition 6. Graph $G = (V_G,D_G,\Sigma_G,\Gamma_G,\delta_G)$ we will call flat representation of the hierarchical graph $H = (V_H,D_H,\Sigma_H,\Gamma_H,\delta_H)$
if $V_G\subseteq V_H$ and for the partition V_H by relation $\approx$ (i.e.$V_{\approx}$) the following condition are fulfilled:

$$\forall X,Y\in V_G|_{\{\underline{P},\underline{O},\underline{N}\}}\colon [Y]\neq[X] \Rightarrow \neg(Y\prec^* X \text{ or } X\prec^* Y) \tag{1}$$

$$\forall U\in V_H|_{\{\underline{P},\underline{O},\underline{N}\}}\ \exists W\in V_G\cap V\colon (U\prec^* W \text{ or } W\prec^* U) \tag{2}$$

$$\forall X\in V_G|_{\{\underline{P},\underline{O},\underline{N}\}} \Rightarrow \mathrm{LocRep}(X)\subset V_G \tag{3}$$

D_G and δ_G are the restriction of δ_H to elements belonging to V_G ■ (4)

Definition 7. Let $H = (V_H,D_H,\Sigma_H,\Gamma_H,\delta_H)$ be the hierarchical graph than we can construct the graph of maximal elements $ME(H)=(V_M,D_M,\Sigma_M,\Gamma_M,\delta_M)$ where $V_M=\{X\in V_H\colon\ \forall W\in V_H\ (X\prec^* W\Rightarrow\ W=W)\}$ and $D_M,\Sigma_M,\Gamma_M,\delta_M$ are the restriction of $D_H,\Sigma_H,\Gamma_H,\delta_H$ to elements belonging to V ■

It can prove that a graph of the maximal elements is the flat representation of the hierarchical graph. Figure 4 represents graph of the maximal of elements constructed from the hierarchical graph represented in figure 3.

Let us note that for elements representing hardware components the relation of the structural dependency is introduced via the equivalence condition so a part of the hierarchical graph representing these hardware connections is simply copied to the graph of maximal elements and is not changed in the flat graph representation.

We believe that the flat representation is well adapted to the presentation of the information in the 2- dimension space. The drawback of the flat representation is the fact that the same idea can be represented in a few context but only one of them is represented by the flat representation; for example considering ERP system model we can express either its internal or external structure. Fortunately, we can switch from one presentation to another on a designer demand.

Now we introduce two operations called analysis and synthesis which transfer one flat graph representation onto another (more detailed/ more abstract) one.

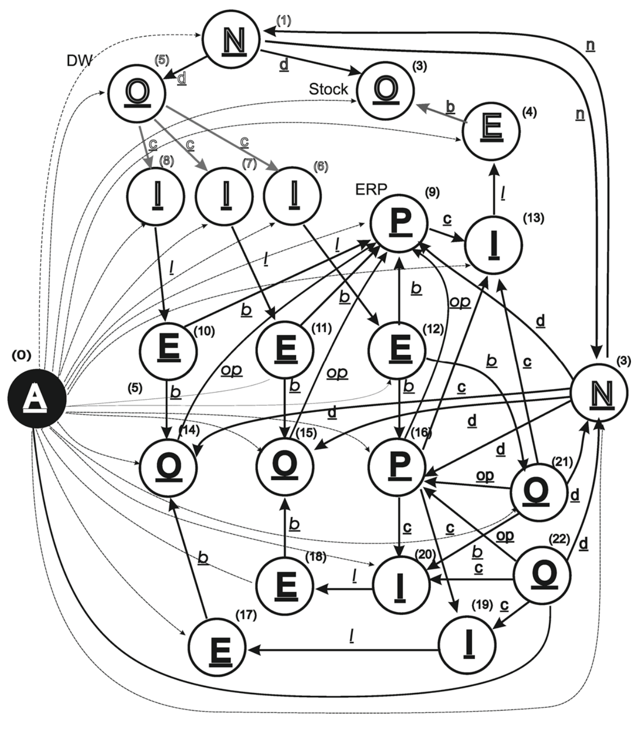


Fig. 3. Hierarchical graph

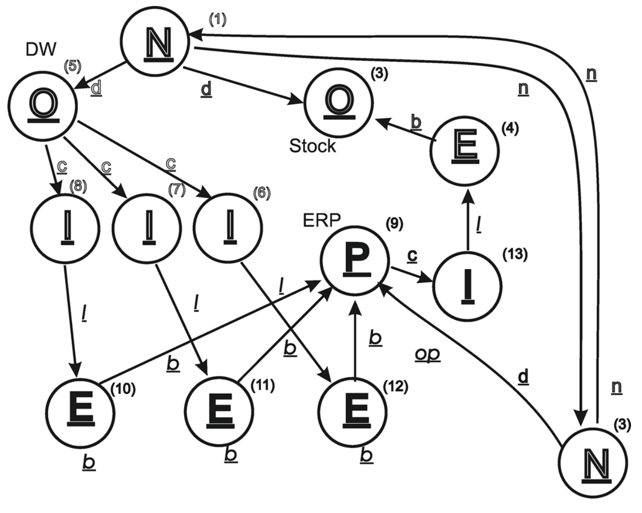


Fig. 4. Graph of maximum elements

Definition 8. Let H be the hierarchical and K its flat representation the for $X \in V_H$ such that $\delta(X)=P$ graph $A(K,H,X)=(V_A,D_A, \Sigma_H,\Gamma_H,\delta)$ where for any T such that $T \prec X$ (X represent a package so at least one such T exist).

$V_A = (V_K\text{-LocRep}(X)) \cup \text{LocRep}([T]_{\approx})$,

$D_A = D_K - \{(U,\mu,W) : U,W \in V_H,\ \mu \in \Gamma$ and
$(U=X \text{ or } W=X)\}$
$\cup\ \{(Y,\underline{c},W):\ Y \in [T]_{\approx}\ \wedge (X,\underline{c},W),\ (Y,\underline{c},W) \in D_H\}$
$\cup\ \{(W,\underline{b},Y):\ Y \in [T]_{\approx} \wedge (W,\underline{b},X),\ (W,\underline{b},Y) \in D_H\}$

δ_G is the restriction of δ_H to elements belonging to V. will be called **analysis of the flat representation** K of the hierarchical graph H in node X ■

Definition 9. Let H be the hierarchical and K its flat representation the for $T \in V_H$ such that $\delta(T) \in \{P,O\}$ graph $S(K,H,T)=(V_S,D_S,\Sigma_H,\Gamma_H,\delta)$ will be **called synthesis of the flat representation** K of the hierarchical graph H in node T if $\exists X \in V_H$ and such that $\delta(X)=P$ and $T \prec X$ then

$V_S = (V_K\text{- LocRep}([T]_{\approx})) \cup \text{LocRep}(X)$,

$D_S = D_K - \{(U,\mu,W) : U,W \in V_H,\ \mu \in \Gamma$ and $(U \in [T]_{\approx}$
or $W \in [T]_{\approx})\}$
$\cup\ \{(X,\underline{c},W):\ \exists Y \in [T]_{\approx} \wedge (X,\underline{c},W),\ (Y,\underline{c},W) \in D_H\}$
$\cup\ \{(W,\underline{b},X):\ \exists Y \in [T]_{\approx} \wedge (W,\underline{b},X),\ (W,\underline{b},Y) \in D_H\}$

δ_G is the restriction of δ_H to elements belonging to V
■

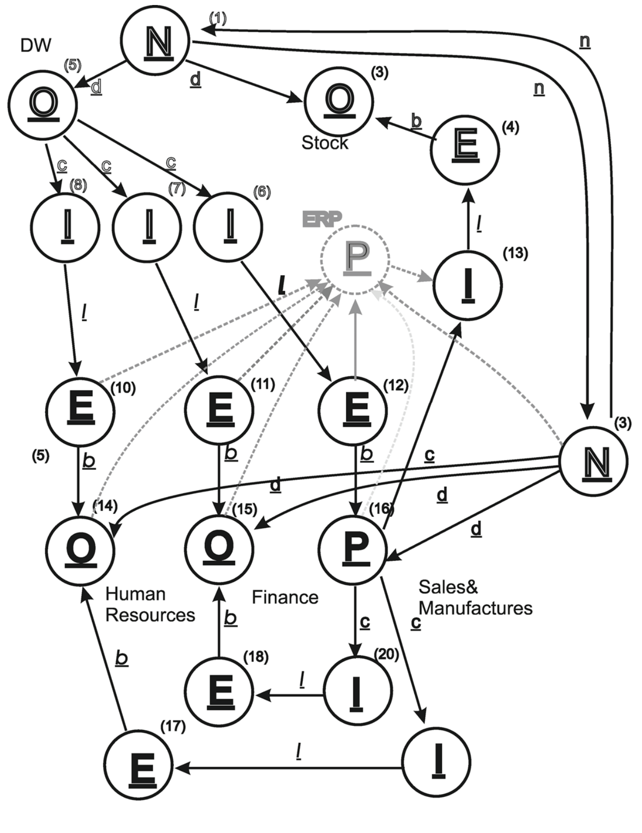


Fig. 5. Graph after analysis of graph of maximal elements

In the figure 5 black part shows the flat graph after analysis of graph of maximal elements ME(H) of the hierarchical graph H (presented in figure 4) in the ERP package context. The grey and dashed part of the graph represents the node and the edges, which has been removed from the analysed graph and exchanged by the appropriate sub graph representing more the detailed description of the ERP package. Any flat graph can be next analysed or synthesed. The figure 6 represents the analysis of this flat graph in context of Sales&Manufactures package (formally A(A(ME(H),H,ERP), H, Sales&

Manufactures)). The operation synthesis is opposite to the analysis so the synthesis of the graph represented in figure 6 for the objects Sales or Manufactures creates the graph represented as black part of figure 5 and the next operation of the synthesis for any element of the ERP package returns the figure 4 (i.e. ME(H)).

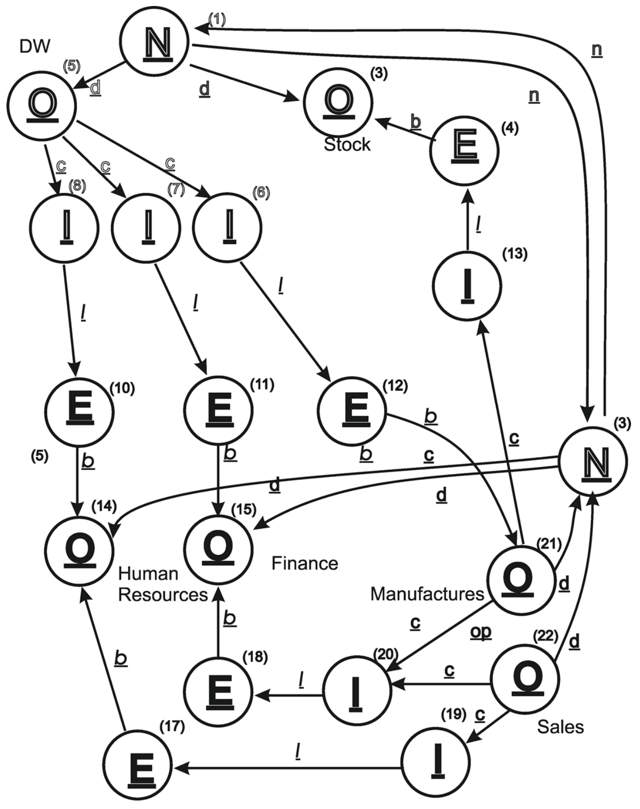


Fig. 6. Flat graph in context of Sales&Manufactures package

4. CONNECTING PHYSICAL AND LOGICAL LAYERS

The proposition of the flat graph visualisation improves our understanding of the software system at the level designated by the UML Deployment Diagram. Such a point of view helps us to understand the connections among the objects in order to improve system effectiveness by better objects allocation in a distributed environment (Kotulski, 2000). It is however insufficient when we would like improve system effectiveness by changing some part of its code.

Any software changes (code improvement, refactoring or extension of system's function) are associated with information presented in UML Class Diagrams. There is not complex work to extend the hierarchical graph by some set of sub-graphs representing class diagrams (figure 7 presets an example).

In order to represent classes inside the hierarchical graph we extend the set of node labels by M (representing a method) and C (representing a class). New relation among these nodes and node labeled by E are also introduced:

r – means that the object entry (labeled by E) represents the class method (labeled by M),

i – means that the class on the left side of edge inheritances the properties of the class on the right one,

o – means that the method on the left side of edge is overridden by the method on the right one

We extend meaning of the label b, which also means that the method (labeled by M) has been directly defined inside the class (labeled by C),

The graph representing class inheritance can not be flatten, because when subclass define a new method the 3-th condition of the flat graph definition (definition 6) can not be fulfilled. This has strong influence on the way of hierarchical diagram visualization proposed in the next chapter.

Let us notice that the node labeled by M will be connected with proper entries of all objects generated from the class, in which this method has been defined. So, during maintenance of the system it will be able to trace the influence some class modification on the behavior of this system.

The software maintenance processes are highly dynamic. Many changes have to be taken into account while software is maintained such as changing requirements or feedback to earlier stages of the software life cycle. In the second case the possibility of tracing of a history of changes executed in the system makes easier taking up the proper decision by the designer.

The possibility of dynamical modification of the hierarchical graph structure in a natural way imply the introduction a graph transformation system to control preservation of the graph properties during its modification. The "history" problem can be simply solved by remembering the production applied to the creation of the current hierarchical graph, and eventually by the filtration them in the context given graph substructure.

Modeling of distributed systems by a distributed graph transformation is not a new idea. Unfortunately, most of the grammars used have either too weak description power to solve the problem mentioned above, or they do not have parsers with polynomial complexity. Layered graph grammars (Rekers, Schűrr, 1997) are proposed to specify visual languages, but it is reported that parsing grammars using Rekers-Schűrr algorithm reaches exponential time (Vermeluen, 1996). Reserved graph grammars

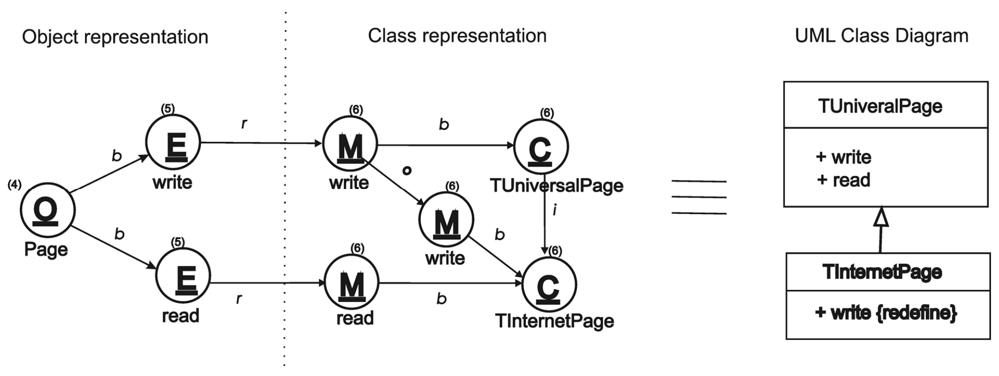


Fig. 7. *Graph representation of UML diagram*

(Zhang, Zhang & Cao, 2001) have also exponential time complexity, but under some conditions (that limits the application scope) a parsing algorithm of polynomial time complexity can be developed. Moreover, most of the propositions of using graph grammars focus on modeling and specifying a distributed system with the help of a graph morphism (Taentzer, Fischer, Koch, Vole, 1999).

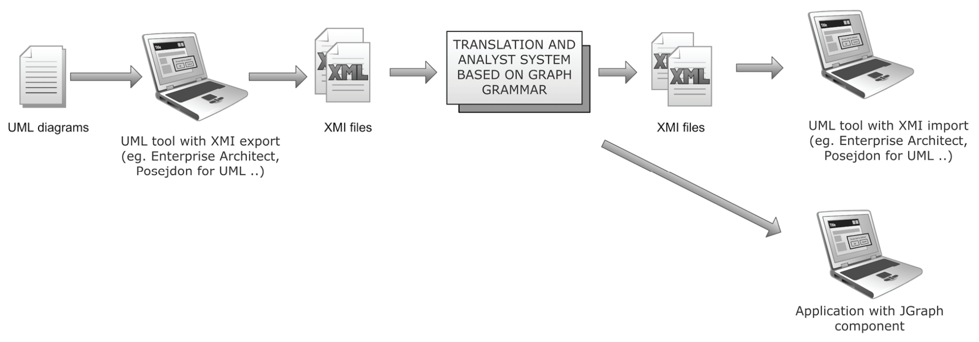

UML diagrams
UML tool with XMI export (eg. Enterprise Architect, Posejdon for UML ..)
XMI files
TRANSLATION AND ANALYST SYSTEM BASED ON GRAPH GRAMMAR
XMI files
UML tool with XMI import (eg. Enterprise Architect, Posejdon for UML ..)
Application with JGraph component

Fig. 8. *Visualization process*

NLC-grammars, introduced by Janssens and Rozenberg (Janssens Rozenberg & Verraedt, 1982), are some of the most interesting families of graph rewriting formalisms. Therefore, for the last 20 years, their formal properties have been extensively studied and a lot of their extensions and variations have been defined (see e.g. Ehrenfreuch Main & Rozenberg, 1984; Flasiński, 1998; Janssens Rozenberg & Verraedt, 1982; Janssens & Rozenberg, 1983). The subclasses of NLC-grammars, defined by Flasiński (1989) – ETPL(k) offer both an efficient parsing algorithm and enough descriptive power to solve a problem of distributed allocation control.

The aedNLC graph transformation system is the final version of a few years investigation on using graph grammars for controlling the distributed software allocation (Flasiński & Kotulski, 1992; Kotulski Jurek & Moczurad, 1992; Kotulski, 2000; Kotulski, 2005). Attributed IE-graphs, generated with the help of it, describe the current allocation state. Moreover, we are able to directly derivate the next allocation state with the linear computational complexity (with respect to the number of allocation graph nodes). The aedNLC graph grammar preserves properties of production and embedding transformation of ETPL(k) graph grammars, thereby offering solutions of parsing and a membership problem with polynomial computational complexity ($O(n^2)$). The aedNLC graph transformation system, using ETPL(k) embedding transformation, has also enough descriptive power to control generation both the nested structure of the UML Deployment Diagrams and inheritance introduced by UML Class Diagram.

5. VISUALIZATION

The proper Man-Machine communication can consist from few tasks: recognizing (hand) drown UML diagrams, gathering an information, its filtration for purpose of better understanding and a visualization of the covered by machine knowledge.

In the case of the software engineering systems basing on ULM concept the first task is solved either by systems understanding diagram notation (Lank, Thorley, Chen, 2000; Tilley, Huang, 2003) or by the context tools (like IBM Rational Software Architect, Enterprise Architect or open source StarUML) supporting UML diagrams creation and storing this information in a data base system. Fortunately, most of these tools are able to export the stored information in the standard form (like RDF (Resource Description Framework), UXF (UML eXchange Format) or XMI (XML Metadata Interchange)). We prefer to use XMI standard because it was specified by OMG (OMG, 2005) and it bases on XML. The most tools cooperate with version 1.1[2], so we prefer that version to store information. In our case XML notation seems to be especially effective, because it

[2] Currently the 2.1 version is already defined.

can be effectively convert onto presented in the above chapters the hierarchical graph notation. Moreover, large systems are designed by developer teams, that often work over a distributed environment (see figure 8) so the introduction of the graph repository enable them coordinate their work, even in such a complex work as a parallel refactoring (Kotulski & Nowak, 2006)

The graph notion offers the possibility of describing very large systems at the human perception level (Calitz & Munro, 2001; Collberg Kobourov Nagra Pitts & Wampler) such a presentation, which is abstracting from details, allows us to trace the behavior of the system and make easier finding the parts of the program that was lately modified or should be improved. Next, when we want to modify the chosen part of the program we ought to return back to detail and trace direct connections among the objects or packages. Moreover, should trace these connections at different points of abstraction introduced either by inheritance or package nesting relations. The presented, in chapter 3, hierarchical graph concept meets the requirements of this task. The strictly graph notation seems to be less intuitive than the flat diagram one. The figure 9 represents an information equivalent to the information maintained by flat graph presented on figure 4. Boxes represent packages or object, and we can make the operation analysis or synthesis to fix the level of the abstraction (in the representation of details) required by the designer.

The graph representing class inheritance can not be flatten, so we will represent them strictly as in the UML class diagram, with one modification: we include context of the object that has been generated from this class, what allows us moving from this class diagram to the flat deployment diagram and back from the given object we can move to its class representation. Any class can be a pattern (direct or by inheritance) many of the generated object; multiple edges coming to the node representing a method (labeled by <u>M</u>) would decrease a diagram readability, so we suggest to represent a class in context only one pointed object generated with help of this class. The other object to be presented in associated with this class diagram the list of currently created objects, and simple pointing an element of this list allows us to move context from the current object to the pointed one. To avoid crossing lines the association between the same method in object and class is outlined only by names, text color and blinking on the class diagram a method pointed in object representation. We do not consider here the technical

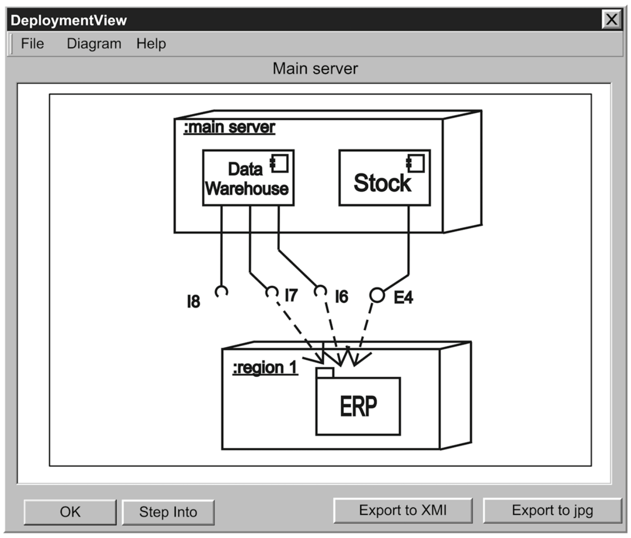


Fig. 9. *UML Deployment View after graph transformation*

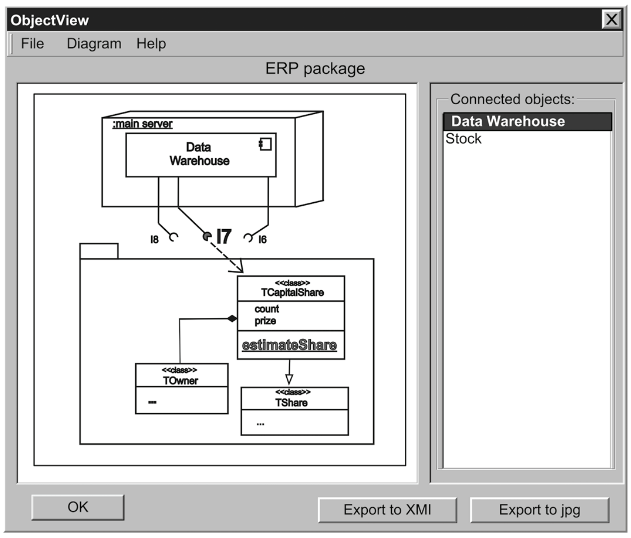


Fig. 10. *UML Object View after graph transformation*

aspects of the class diagram visualization, because there are defined a few tools (Gutwenger Jünger Klein Kupke Leipert & Mutzel, 2003) supporting for UML class diagrams following aesthetic preferences: crossing minimization, bend minimization, orthogonality, horizontal labels and joined inheritance arcs. These preferences are supported by Go Visual graph drawing library.

6. CONCLUSIONS

The visualization of the knowledge gathered inside the computer system, in a way that exploits human capabilities for rapid, recognition is the challenging problem. In the paper, we have suggested that the use of hierarchical graphs is an efficient formalism for gathering and formally represent information contained into UML deployment and class diagrams. The transformation from ULM diagrams (represent in XML notation) to the hierarchical graph structure (defined in the paper) can be made effectively with help of the aedNLC graph transformation system. This transformation assures the correctness of the graph during the software maintenance phase (for example during the refactoring of the system (Kotulski & Nowak, 2006) and help to solve the problem of remembering the history of the software modification. For large systems the hierarchical graphs becomes very complicated, so we define the formal background of the simplification of their notation. Introduced in chapter 3 flat graph notation, with the operations analysis and synthesis, seems to be an attractive proposition of a presentation UML diagrams at different levels of abstraction. While a system maintenance phase we enable exposing the association of the given object with the set of classes (defined by inheritance hierarchy) from which it was generated. We can also check how the modification of the given class influences to the whole system.

At the end the technical problems of the gathered information visualization are considered and some suggestions of using the existing systems have been formulated. To resume, the proposed solutions seam to signify both the theoretical (formally defined semantics) and the practical (because the presented graph transformation mechanism has enough descriptive power to control the software allocation and visualization process and made it with the polynomial time complexity and there exist method supporting the hierarchical graph visualization at the human perception level).

7. REFERENCES

Baird, H.S., Lopresti, D., Davison, B.D., Pottenger, W.M., Robust document image understanding technologies. In *Proceedings of the 1st ACM Workshop on Hardcopy Document Processing* (Washington, DC, USA, November 12 - 12, 2004). HDP '04. ACM Press, New York, NY, 9-14.

Booch, G., Rumbaugh J., Jacobson I., *The Unified Modeling Language – User Guide*. Addison Wesley Longman, Inc. 1999.

Can Keskin, Vogelmann V., Effective *Visualization of Hierarchical Graphs with the Cityscape Metaphor*, University of Karlsruhe, Telecooperation Office, ACM 1998 l-58113-051-1

Calitz, A.P., Munro, D., Representation of hierarchical structures in 3D space. In *Proceedings of the 1st international Conference on Computer Graphics*, Virtual Reality and Visualization (Camps Bay, Cape Town, South Africa, November 05 - 07, 2001). AFRIGRAPH '01. ACM Press, New York, NY, 59-64.

Collberg, C, Kobourov, S, Nagra, J, Pitts, J, Wampler, K., *A System for Graph-Based Visualization of the Evolution of Software*, Association for Computing Machinery, Inc.

Ehrenfreuch, A., Main, M.G., Rozenberg, G., Restrictions on NLC graph grammars. *Theoretical Computer Science*, 1984.

Flasiński, M., Characteristic of edNLC-graph Grammars for Syntactic Pattern Recognition. *Computer Vision, Graphics and Image Processing*, 1989.

Flasiński, M., Power Properties of NCL Graph Grammars with a Polynomial Membership Problem. *Theoretical Computer Science*, 1998.

Flasiński, M., Kotulski, L., On the Use of Graph Grammars for the Control of a Distributed Software Allocation. *The Computer Journal*, 1992.

Fuentes, J.M., Quintana, V., Lloren,s J., Génova, G., Prieto-Díaz, R., Errors in the UML metamodel. *SIGSOFT Software Eng. Notes* 28, 6 (Nov. 2003), 3-3.

Gutwenger, C., Jünger, M., Klein, K., Kupke, J., Leipert, S., and Mutzel, P. ,A new approach for visualizing UML class diagrams. In *Proceedings of the 2003 ACM Symposium on Software Visualization* (San Diego, California, June 11 - 13, 2003). SoftVis '03. ACM Press, New York, NY, 179-188.

Janssens, D., Rozenberg, G., Verraedt, R., On Sequential and Parallel Node-rewriting Graph Grammars. *Computer Graphics and Image Processing*, 1982.

Janssens, D., Rozenberg, G., Graph grammars with node-label controlled rewriting and embedding, *LNCS*, 1983.

Kotulski, L., Jurek, J., Moczurad, W., Object-Oriented Programming in the Large Using Group Concept. Computer Systems and Software Engineering - *6th Annual European Conference*, Hague 1992.

Kotulski, L., Nowak, A., Graph repository as a core of environment for distributed software restructuring and refactoring. Accepted for publication in *24-th IASTED international conference APPLIED INFORMATCS*, Innsbruck 2006.

Kotulski, L., Model systemu wspomagania generacji oprogramowania współbieżnego w środowisku rozproszonym za pomocą gramatyk grafowych. *Postdoctoral Lecturing Qualifications*. Jagiellonian University Press, ISBN 83-233-1391-1, 2000.

Kotulski, L., Graph representation of the nested software structure. V.S. Sunderam et al. (eds), *ICCS 2005*, LNCS 3516.

Kroll, P., Kruchten, P., *The Rational Unified Process Made Easy A Practitioner's Guide to the RUP*, Addison Wesley 2003

Kramer, J., Ng, K., Magee, J., Dulay, N., *The System Architecture's Assistant – A Visual Environment for Distributed Programming*. 28th HICSS, Hawaii, 1995, Software Track.

Lank, E., Thorley, J.S., Chen, S.J., An interactive system for recognizing hand drawn UML diagrams. In *Proceedings of the 2000 Conference of the Center For Advanced Studies on Collaborative Research* (Mississauga, Ontario, Canada, November 13 - 16, 2000). S. A. MacKay and J. H. Johnson, Eds. IBM Center for Advanced Studies Conference. IBM Press, 7.

Ogiela, M.R., Tadeusiewicz R., Non linearity processing and semantics context analysis In medical imaging approach.

IEEE *Transactions on Instrumentation and Measurement*, 2005.

OMG - Object Management Group , MOF 2.0/XMI Mapping Specification, v2.1, formal/05-09-01, http://www.omg.org/technoogy/documents/modeling_spec_catalog.htm#XMI, 2005

Rekers, J., Schűrr, A., Defining and parsing visual languages with layered graph grammars. *J. Visual Languages Comput.* 1997.

Routledge, N, Bird L., Goodchild, A., UML and XML Schema, Australian Computer Society, *Conferences in Research and Practice in Information Technology*, Xiaofang Zhou, Ed. 2002.

Tilley, S., Huang, S., A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding, *SIGDOC'03*, October 12–15, 2003, San Francisco, California, USA.

Tadeusiewicz, R., Ogiela, M. R., Medical Image Understanding Technology, Series, *Studies in Fuzziness and Soft Computing*, Springer-Verlag, Berlin - Heidelberg - New York, 2004.

Tadeusiewicz, R., Ogiela, M.R., Why Automatic Understanding? In Beliczynski B., Dzielinski A., Iwanowski M., Riberiro B. (Eds.), *Adaptive and Natural Computing Algorithms, Lecture Notes on Computer Science, Part II*, Springer-Verlag, Berlin - Heidelberg - New York, 2007.

Tadeusiewicz R., Ogiela L., Ogiela M.R., Cognitive Analysis Techniques in Business Planning and Decision Support Systems, in Rutkowski L. (et al. eds.), *Artificial Intelligence and Soft Computing, Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin - Heidelberg - New York, 2006.

Taentzer, G., Fischer I., Koch, M., Vole, V., Visual design of distributed graph transformation. In *Handbook of Graph Grammars and Computing by Graph Transformation, Concurrency*, Parallelism, and Distribution. World Scientific, 1999.

Taentzer, G., *A Visual Modeling Framework for Distributed Object Computing. In Formal methods for open object-based distributed systems*, Kluwer Academic Publishers, 2002

Vermeluen, J.T., Viability of Parsing Algorithm for Context-sensitive Graph Grammars. *Technical Report*, Leiden University, 1996.

Zhang, D., Zhang, K., Cao, J., A context-sensitive Graph Grammar Formalism for the Specification of Visual Languages. *The Computer Journal*, 2001.

Zhao, W., Chella, A., R., Phillips, P.J., and Rosenfeld, A., 2003. Face recognition A literature survey. *ACM Comput. Surv.* 35, 4 (Dec. 2003).

WSPARCIE ANALIZY DIAGRAMÓW UML PODCZAS PROCESU UTRZYMANIA SYSTEMÓW INFORMATYCZNYCH

Streszczenie

Diagramy UML są powszechnie stosowaną techniką wspomagającą modelowanie systemów obiektowych; stają się również wysoce użyteczne podczas modyfikacji systemu spowodowanej zmianami wymagań użytkowników. Pełny model systemu ilustruje jednak wielowymiarowe relacje pomiędzy setkami (tysiącami) składników, często opisywanych w różnych kontekstach (typach diagramów) i poziomach szczegółowości. Takie nasycenie elementów i relacji pomiędzy nimi powoduje bardzo często że model systemu przekracza możliwości percepcyjne człowieka i często jest on pomijany podczas procesu rozwoju. W artykule podejmujemy próbę czytelnego przedstawienia tych powiązań na dwuwymiarowym ekranie komputera. Proponujemy wyświetlanie informacji przedstawiającą tylko jedną perspektywę (tzw. płaską wizualizację grafu) zawierającą przefiltrowany zbiór informacji dotyczących danej części systemu (logicznej lub fizycznej). Udostępniamy użytkownikowi możliwość iteratywnej zmiany tej prospektywny na inną, która z nią jest powiązana. Proces ten opieramy na bazie grafów hierarchicznych, podajemy formalną specyfikację pojęcia płaskiej wizualizacji grafu, oraz definiujemy operacje analizy i syntezy pozwalające zmienić analizowaną perspektywę na inną bardziej (mniej) szczegółową. Wykorzystanie wprowadzonego aparatu formalnego, przedstawione jest na przykładzie analizy diagramów klas i diagramów wdrożenia.

Received: April 5, 2007
Received in a revised form: May 22, 2007
Accepted: May 24, 2007