BULETINUL Universității Petrol – Gaze din Ploiești	Vol. LVIII No. 3/2006	13 - 22	Seria Ştiinţe Economice

Cycle: UML FOR MANAGERS (I)[1]
Notations and Fundamental Concepts

Liviu Dumitraşcu, Gabriel Irinel Marcu

Universitatea Petrol-Gaze din Ploieşti, Bd. Bucureşti 39, Ploieşti
email: ldumitrascu@upg-ploiesti.ro, gimarcu@upg-ploiesti.ro

Abstract

*In order to construct a system, one must first model it. Modelling requires a language. This paper presents the symbols and the fundamental concepts of UML language (**U**nified **M**odelling **L**anguage), an industrial unified modelling standard developed under the direct responsibility of OMG (Object Management Group)*

Key-words: *Class Diagram, Package Diagram, Component Diagram, Deployment Diagram, Composite Diagram, Use Case Diagram, State Machine Diagram, Activity Diagram, Sequence Diagram, Communication Diagram, Interaction Overview Diagram, Timing Diagram*

The Models Necessity

Basic Principles of Modelling

Frequently, in every day life, we meet various *models*.

Generally, we can say that we cannot realize certain object without having in mind, at the beginning vaguely, in order to become more and more overtly expressed, a model of the respective object.

If we talking about a system in general or about a system of data applications in particular, we can say that their project of developing indubitably starts with a model which reduces the final system to its essential elements, thus allowing both the systematic approach and the overall control during the execution.

A model is, thus, an abstract representation of a system which make possible the planning, study, analysis, conception and over checking of the system performances before its proper realization, while at the same time creating the design documentation hence facilitating communication between the teams of designers.

[1] Editor's note: Starting with this number we will be issuing nine articles under the heading "UML for Managers" (**U**nified **M**odelling **L**anguage), an industrial modelling standard. Authors: Prof. PhD Liviu Dumitrascu, Assoc. Prof. PhD. Gabriel Irinel Marcu

The model proves itself highly useful in three different stages of the system design : in requirement specifications of the new system, in the analysis activity and formulation of software solution for the designed system and in the stage of system developing and realization.

The modelling principles [4], brilliantly formulated by ‘the three amigos’- Booch Grady, Rumbaugh James, Jacobson Ivar (the founding fathers of UML) are the following: choosing the models essentially influences the way in which problems and their solving are approached; models can have various levels of precision; the best models are based on the sense of reality; since a model is not enough in itself, it is preferable to destroy the big system into a group of small almost independent models which could be studied and tested separately.

Models can have various uses: they are necessary for the delimitation of complex systems thus facilitating their monitoring; meant to optimize the system organization; they allow for the full and precise description of the user’s/client’s requirements; they facilitate the conception of a system by realizing an approximate scale model etc.; they allow to test various solutions requiring minimal costs and to choose the most appropriate solution for the proposed problems.

Object-Oriented Modelling

Why an Object-Oriented Language?

In order to realize a system one must first model it.

Object-oriented modelling generates discrete models which allow the regrouping of an assembly of possible system configurations which can be implemented in a programming language. The object-oriented modelling of a system starts with the definition of the objects belonging to the studied domain.

The great advantage of the object-oriented approach is that if we choose for software application developing an object- oriented programming language (Java), the objects are preserved and they will be developed in the same structure and relationship, starting from analysis and up to the complete accomplishment.

The manager of the project needs instruments in order to fulfil the potential demands, to refine and improve them during the entire life span. The object-oriented development can provide such instruments.

The object-oriented development includes the so-called *use case* which model the functional demands of the user/client. The *use case* forms a bridge between the client’s and the designer’s point of view, thus improving the understanding in the same way of the observed system phenomena.

Basic Principles of the Object-Oriented Approach

As opposed to other computer programmation tehniques, the object-oriented programming, described as a unified process, is based on object-oriented observation starting from the beginning of the system analysis. An object can be the abstract counterpart of a specific entity used in the analysis or the component of a program (solution) still in the realization stage. The relation between these is much more obvious if the used language is itself object-oriented (Java).

Object-oriented development starts by defining the human actors and the functions of the system as objects interacting in order to produce a certain distinct and important result for the project. This is how to generate the use cases. Then, use cases are fragmented into classes of objects of the same type, objects which contain characteristic data and their use instructions. At the end, the behaviour of the system is described by means of diagrams of sequence and communication (collaboration) between these classes which end up by accomplishing all the functions of the designed project.

The basic principles of the object-oriented approach are the following: encapsulation, inheritance and polymorphism.

Encapsulation

The main purpose of an object-oriented language (Java) is to emulate *objects* from the real world by defining a certain class into a program. At the same time, we can consider that a class is a new type of data which can be defined and used in order to generate instances of the same class.

A class combines the attributes and behaviours associated to an object into a new type of data. This process is called *encapsulation*. Programmers like to say the fact that the attributes and components are encapsulated into the definition of the class.

*Remarks.*According to other authors, encapsulation is more restrictive in meaning, referring to protecting internal data of the objects so that they cannot be directly accessible (by using the point operator). Encapsulation generates a filed of strength around the instance variables so that nobody can assign them the inappropriate values.

Inheritance

Some people consider that inheriting a fortune is the best thing that could happen to them because there is no need for work and saving in order to get what they want. Somebody else did that for you, leaving you only the task of thinking of how to spend the money. [1]

Inheritance has a similar effect when you write a program (Java) also. Even if you still have to count every dime, the inheritance allows you to use all the classes written by somebody else. Instead of writing all the classes from the beginning you can concentrate on the classes which have not yet been written. [1]

The programmers are trying to expose the instance variables (attributes) and to define the member methods (behaviour) common to more objects into a single class (super-class). Then, the classes (subclasses) which define related objects can inherit the respective class.

Polymorphism

The Greeks have coined a term in order to denominate something having more than one shape: *polymorphism.*

This term is also used by programmers (Java) in order to describe the capacity of the object-oriented language (Java) to have a method with more significations (shapes) according to the context in which it is used in a program (Java).

Polymorphism allows you the preservation of the same term for similar operations but which have different functions, the identification being made according to both the call-in format (the shape and the number of the parameters), and to the class it belongs to. This allows you the simplification of the procedures and action logics understanding, as well as of the data applications that generate them.

UML, an Unified Modelling Language

The Historical Context of UML

UML has been created by the fusion of three distinct modelling methods: The Booch method, developed by Grady Booch; Object Modelling Technique (OMT), developed under James Rumbaugh's coordination; the objectory method, conceived by Ivar Jacobson.

Familiarly called "the three amigos", Booch, Rumbaugh and Jacobson have introduced the first UML version in 1994. In 1997, under the direct responsibility of OMG (Object Management Group) there has appeared UML v1.1.

Starting with this époque, UML has constituted the object of various revisions and improvements as follows:UML 1.2 (July, 1998);UML 1.3 (June, 1999);UML 1.4 (May, 2002);UML 1.5 (March, 2003);UML 2.0 (October, 2004).

UML 2.0 contains significant changes and real extensions as compared to 1.X version.

Version 2.0 contains the biggest number of specifications (600 pages used to describe the model); this version is indubitably the clearest and most compact ever published.

Remarks.

- UML is not a programming language;
- UML cannot be used in order to model on-going processes or to validate a system;
- UML cannot be used to generate a full accomplishable program, but it allows for the generating of code fragments (for example: the structure of the classes- attributes, methods);
- UML is an instrument of visual modelling;
- UML is an extensible language (for example: the stereotypes which extend the basic UML concepts, introducing model specific concepts).

UML-Connected Approaches

UML is not a programming language. It is a modelling language, based on events/ messages.

UML (Unified Modelling Language), as the name itself indicates it, represents a modelling language and not a method. This statement is important from the point of view of those who have conceived it, who have wanted to dissociate, contrary to the prior attitude, the method from its notation. (Java/.Net)

UML is independent from the platform as well as from the programming language.

UML is a visual modelling instrument.

A method of development defines a modelling language and a process. The modelling language is a notation (mainly graphical) used by the models in order to represents the conception. The process materializes the way to be followed during the conception phase. UML language proposes only a (graphical) notation defined by means of a standard but not a complete methodology. Up to the moment, there are known several methods of complete development based on UML: Unified Process (UP), Rational Unified Process (RUP); Model Driven Architecture (MDA).

UP (Unified Process) is a process of development for data applications, process described as interactive and instrumental, lead by means of use cases.

RUP (Rational Unified Process) represents a process of development of software applications entirely based on UML.

The difference between RUP and UP resides in the fact that RUP contains a significant number of available models. The use of UML doesn't impose the subsequent use of RUP. RUP is presented as a HTML site (http://www.rational.com).

MDA (Model Driven Architecture) has as a main objective the conception of systems based only on the domain modelling, regardless of technological aspects.

In MDA, the model of the domain objects is called PIM (Platform Independent Model). PIM is made from a group of elements whose conception has to preserve its independence from the programming language or from technology. This model is then manually or automatically turned into a model specific to a platform (Java/.Net) and to a programming language. Such a specific model is called PSM(Platform Specific Model).

The connection with UML is realised at PIM level. UML offers the advantage of a fine description of the objects, thus remaining independent from technologies. At a processing level, *the Action Semantics* extension of UML must allows it to fulfil all the user described requirements.

Structural and Behavioural Modelling

UML allows the construction of various models of a system, each of those emphasizing different aspects: functional, static, dynamic and organisational.

Generally, an UML model is made of one or more diagrams. A diagram is a graphic representation of the objects and of the relations between them.

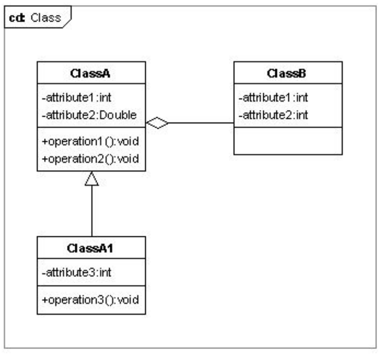
These objects can represent the objects of the real world, software elements etc.; at the same time they can describe the behaviour of a particular object. An object can appear in more diagrams because it describes the entity which is to be modelled from various points of view (design, implementation, configuration, installation, process).

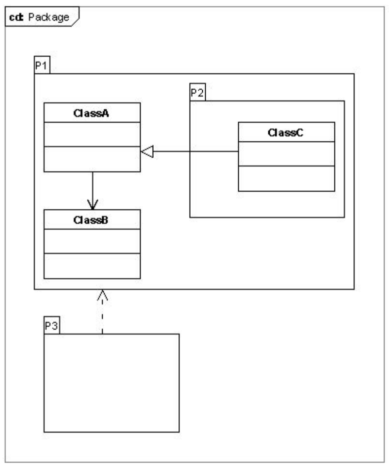
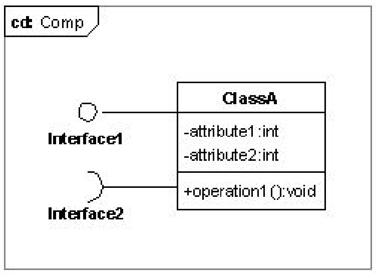
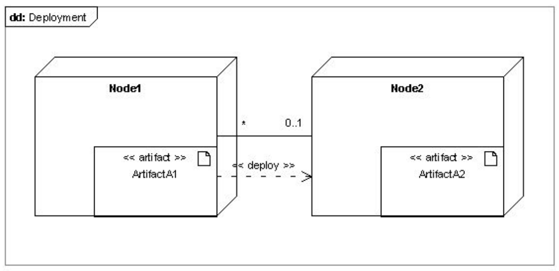
Remark. The concept of approach is not, strictly speaking, part of UML.

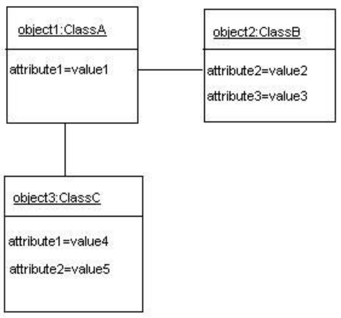
Structural Modelling facilitates the understanding and the functioning of the component elements owned by the system. UML language uses for the structural modelling a system of 6 static (structural) diagrams: Class Diagram, Object Diagram, Package Diagram, Component Diagram, Composite Structure Diagram, Deployment Diagram.

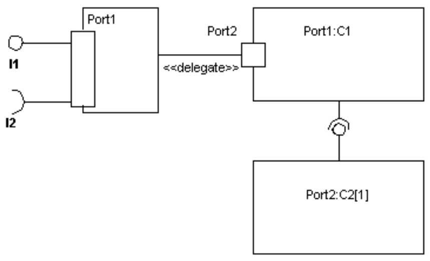
Table1 briefly presents UML 2 structural diagram family.

Table 1

Diagram	*UML Version*	*Role*	*Graphic Representation*
Class Diagram	UML 1	Structural class description and relationship description among classes	 cd: Class ClassA -attribute1:int -attribute2:Double +operation1():void +operation2():void ClassB -attribute1:int -attribute2:int ClassA1 -attribute3:int +operation3():void

Package Diagram	UML 2	Logic organization of model and relationship between package	
Component Diagram	UML 1	Complex structure description	 cd: Comp ClassA Interface1 -attribute1 :int -attribute2:int +operation1 ():void Interface2
Deployment Diagram	UML 1	Artefact distribution into system resources	 dd: Deployment Node1 Node2 * 0..1 << artifact >> ArtifactA1 << deploy >> << artifact >> ArtifactA2

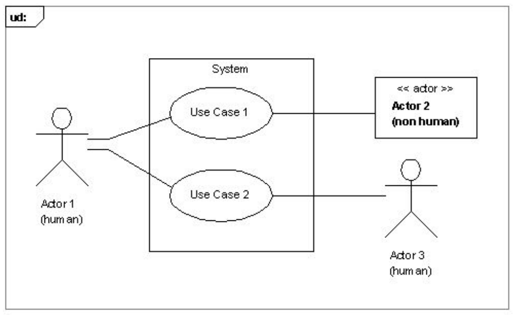
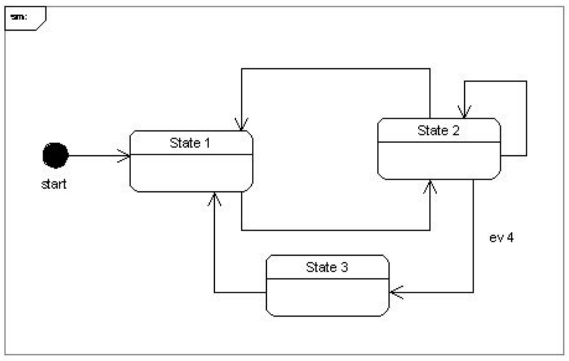
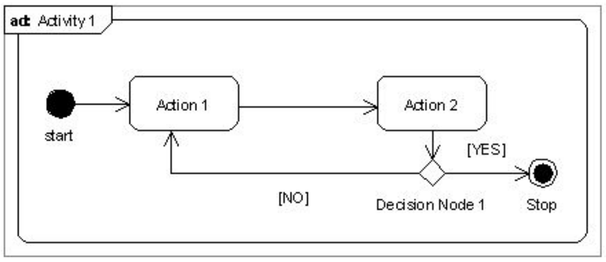
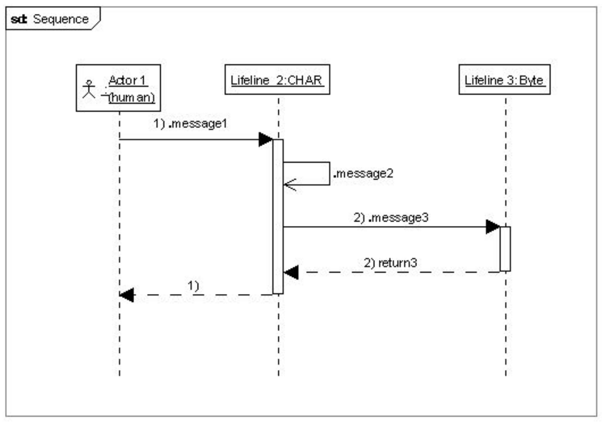
Object Diagram	UML 2	Classes instances description (instant image of software application)	 object1:ClassA
			attribute1=value1
			object2:ClassB
			attribute2=value2
			attribute3=value3
			object3:ClassC
			attribute1=value4
			attribute2=value5

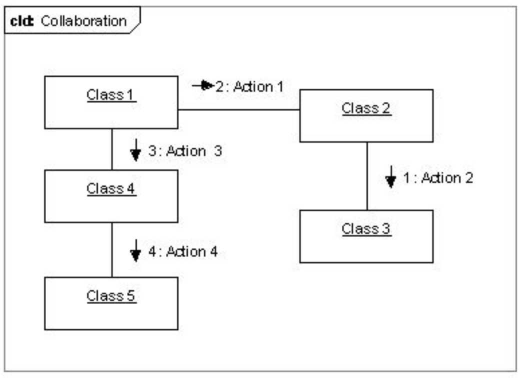
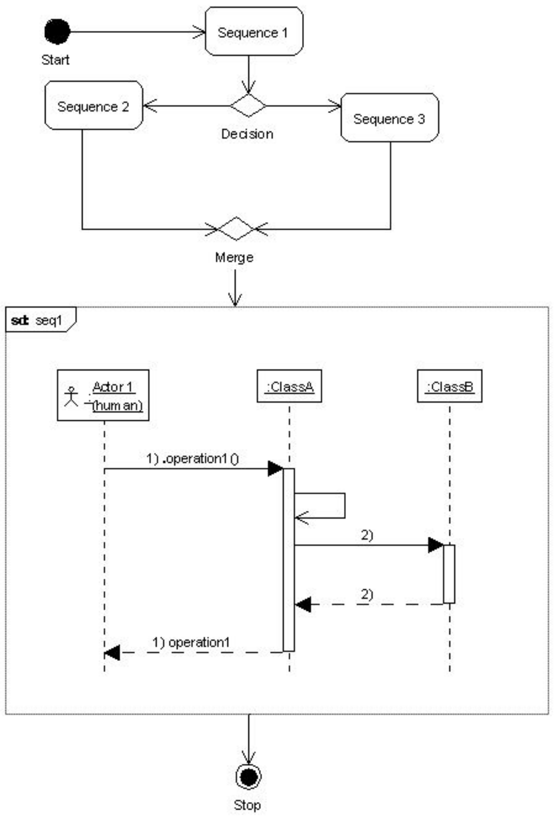
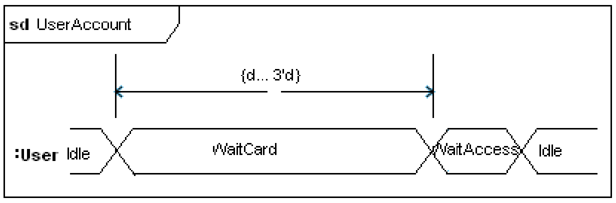
Composite Structure Diagram	UML 2	Internal arrangement of complex static element	 Port1 I1 I2 Port2 <<delegate>> Port1:C1 Port2:C2[1]

Behavioural modelling facilitates the understanding and description of the way in which the elements of a system interact and collaborate in order to ensure its functionalities.

UML language uses for the behavioural modelling of a system 7 dynamic (behavioural) diagrams: Use Case Diagram; Sequence Diagram; Communication Diagram; Interaction Overview Diagram; Timing Diagram; Activity Diagram; State Machine Diagram. Table 2 briefly presents UML 2 behavioural diagram family.

Table 2

Diagram	*UML Version*	*Role*	*Graphic Representation*
Use Case Diagram	UML 1	Functional interaction description between actors and system	 ud: System Use Case 1 Use Case 2 << actor >> Actor 2 (non human) Actor 1 (human) Actor 3 (human)
State Machine Diagram	UML 1	Classes behavior description using their transition and state machine	
Activity Diagram	UML 1	Chain model for actions or decisions. Can replace the state machine diagram	 act Activity 1 start Action 1 Action 2 [YES] [NO] Decision Node 1 Stop
Sequence diagram	UML 1	Vertical ordered sequence description of messages between objects	 sd Sequence Actor 1 (human) Lifeline 2:CHAR Lifeline 3:Byte 1) .message1 .message2 2) .message3 2) return3 1)

Communication diagram	UML 1	Message flow between objects presentation showing the basic associations (relationships) between classes	 cld Collaboration Class 1 2 : Action 1 Class 2 3 : Action 3 1 : Action 2 Class 4 Class 3 4 : Action 4 Class 5
Interaction Overview Diagram	UML 2	Is variant of UML activity diagrams which overview control flow joining the other types of UML interaction diagram(sequence diagram, communication diagram, timing diagram, interaction overview diagram) or interaction occurrence which indicate an activity or operation to invoke.	 Start Sequence 1 Sequence 2 Decision Sequence 3 Merge sd seq1 Actor 1 (human) :ClassA :ClassB 1) operation1 () 2) 2) 1) operation1 Stop
Timming diagram	UML 2	Presentation of behaviors of one or more objects throughout a given period of time	 sd UserAccount {d... 3*d} :User Idle WaitCard WaitAccess Idle

Software Tools for UML 2

UML Software Tools Market

UML has been created in order to be also used by (brand) software applications which allow both the construction of UML diagrams and the code generating process (Java, C#, PHP etc.) in order to accomplish object-oriented application projects.

The market for UML software tools has not yet reached full maturity since there are still hundreds of instruments with different philosophies and qualities. Generally, the UML tools provide various features but the most important factors are: user interface look and feel, support for code generation and code import, conformance with the latest UML standard, and integration with the environment. The UML tools must not save the UML diagrams in their own formats which is not compatible with the formats of other UML tools.

The UML software tools increase the program developer productivity as well as the quality of software application modelled.

A recent market study of the UML tools identifies three categories of software products: multi-purpose software tools, code generating software tools and MDA (Model Driven Architecture) software tools.

Multi- Purpose Software Tools

Multipurpose software tools belong to the first generation of UML tools and can be used in the analysis, conception and code generating phases of the development of software applications.

The main UML software products pertaining to this category are: IBM- Rational Rose; Borland Together Control Centre; Softeam Objecteering, Embarcadero Describe; Gentleware Poseidon for UML; Popkin Support Architect; Tigris ArgoUML; Visual Paradigm for UML; Enterprise Architect etc.

Code Generating Software Tools

Code generating software tools grant the data application generators the possibility to immediately visualise the structure of the code by synchronising the generated code and the edited UML class diagrams.

The main software products which form part of this category are: Borland Together; IBM Rational XDE; Oracle JDeveloper 10g; Omondo UML; IDE Visual Studio 8 etc.

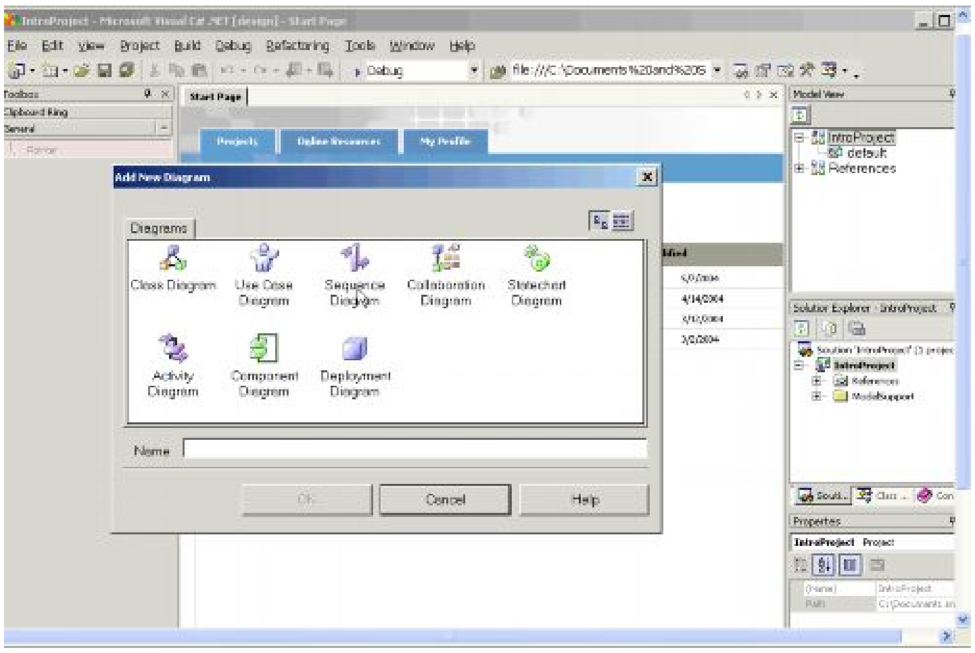
Figure 1 presents the architecture of the product Borland Together.

MDA Tools

MDA is an approach to software development that provides a set of guidelines for structuring specifications expressed as models [4].

MDA standard is mainly based on two standards of OMG: MOF (Meta-Object Facility) for the model data and XMI (XML Metadata Interchange) for the exchange between tools, but is related to other multiple standards including EDOC (Enterprise Distributed Object Computing), SPEM (Software Process Engineering Metamodel), CWM (Common Warehouse Metamodel).

Figure 1



The analysis of the UML software tools market has demonstrated that there is no clear demarcation between the software products approaching MDA and the software tools belonging to the other two categories.

The main software products which represent MDA solutions are the following: Compuware Optimal J; Arcstyler; Codagen Architect; AndroMDA; Kabira; Kennedy Carter UML; I-Logix Rhapsody etc.

References

1. K e o g h , J. - *Java fără mistere*, Editura Rosetti Educational, 2006
2. P i l o n e , D., P i t m a n , N. - *UML 2 en concéntre. Manuel de référence*, O'Reilly, 2006
3. R o q u e s , P. - *Memento UML*, Eyrolles, 2006
4. ***, http://en.wikipedia.org/wiki/Model_Driven_Architecture

Ciclul UML PENTRU MANAGERI (I)
Notații şi concepte fundamentale

Rezumat

Pentru a realiza un sistem trebuie mai întâi să-l modelăm. Modelarea are nevoie de un limbaj. În acest prim articol se prezintă notațiile şi conceptele fundamentale ale limbajului UML (Unified Modelling Language), standard industrial de modelare unificat dezvoltat sub directa responsabilitate a OMG (Object Management Group).