SmartModels – A Model Oriented Approach Validated by a Prototype Based on Eclipse Platform

E. Țundrea, D. Pescaru, C. B. Chirilă
"Politehnica" University of Timişoara
Department of Computer Science, V. Pârvan no 2, Timişoara, România
emanuel@emanuel.ro, dan@cs.utt.ro, chirila@cs.utt.ro

Abstract - **Emergent behavior is that which cannot be predicted through analysis at any level simpler than that of the system as a whole. Emergent behavior, by definition, is what is left after everything else has been explained [6]. This is one of the main concerns of the Object-Oriented Programming (OOP) principles which did not cure important issues faced by software companies these days on developing complex software for reuse and protecting the more and more evolving applications against technological obsolescence.**

This paper presents:
- *an approach*: it reviews the state-of-the-art of SmartModels approach briefly introducing its principles, basic entities and main elements when defining a business-model. It also addresses the Meta-Object Protocol (MOP) which lays the foundation of SmartModels' mechanism to fill the gap between the semantics and the reification of a model entity;
- *a prototype*: SmartFactory which is based on Eclipse platform and its role is to validate the new approach.

Keywords - **software, model, generative programming, prototype, factory**

I. SMARTMODELS – AN APPROACH BASED ON MODELS

SmartModels proposal relies on previous works which deal on the one hand with meta-modeling [2], and on the other hand with the design of a software factory called SmartTools [1]. It intends to enrich both approaches in order to make easier the development of domain-specific applications. It is a first attempt to create our practical interpretation of MDA [] principles.

The main objective of SmartModels is on the one hand, to clearly identify, thanks to a meta-level, the semantics of concepts used for the modeling of a given domain, and on the other hand, thanks to approaches by separation of concerns and generative programming [3], to equip, in a modular way, the applications related to this domain.

SmartModels is a set of domain specific models dedicated to the development of software. This approach is original and may be distinguished from other approaches by the following characteristics:

- it introduces on top of the entities which structure the model (reification level), a semantic layer which enables to define and factorize the basic functionalities related to the domain;
- it provides a set of facilities (in order to quickly build applications related to the model), which strongly relies on the two levels of the model (data and semantic models);
- it ensures a clear separation between the model and the technologies which makes the model executable by a software platform.

The main interest of such an approach is to provide the power to define the semantics of the entities which are addressed by a model, independently from any application. In general, the semantics is spread out in the applications which may directly handle the model.

SmartModels does not make any difference between the modeling of the business model and the modeling of its applications. Thanks to the semantics which is encapsulated in the entities, related applications may handle directly this knowledge without going through some implementation phases (the generation process takes care of this).

It is very important to know that contributions of both generative programming and separation of concerns are used in order to achieve a better flexibility and modularity of the applications related to the model.

II. MAIN ELEMENTS OF A MODEL IN SMARTMODELS

This section briefly introduces the main elements we can use to describe a business-model in SmartModels. For a more detailed presentation please see [13, 14, 15]. A business-model is defined through the identification of its entities according to the know-how of a specific domain. This approach follows the Domain-Driven Development [4] principles and therefore offers a framework for development of domain-specific applications.

The process consist in producing an XML document (i.e. by a parser of the domain specific language) compliant with the AST (or DTD) which describes a model in our approach. This document will drive the generation process of a class (Java class in the current version) for each entity. Then this set of generated classes, considered an implementation of the

business-model, is attached to our MOP as sub-hierarchies of the built-in kernel. Our MOP encapsulates on one hand features for handling access in the specialized / meta hierarchies and its extension, and on the other hand for loading/saving instances of entities from/into XML streams.

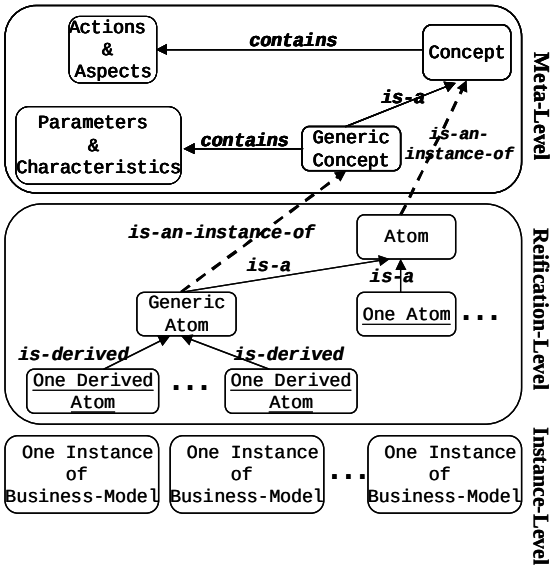

Figure 1. Elements of a business-model in SmartModels

Next paragraphs will present each entity of SmartModels with respect to the level it manifests. Figure 1 distinguishes between the different levels of the architecture of our meta-model: the main elements proposed by SmartModels in order to define business models. They will be used by generators in SmartFactory in order to produce code attached to the MOP.

A. SmartModels Meta-Level

First of all, the *meta-level* is the top level of SmartModels business-model reification and it handles the meta-information through concepts. A concept participates to the definition and the management of the meta-information of a business-model. It encapsulates the semantics of entities and their treatments. It can be related to one or a number of atoms and drives their behavior. Just as a forward-looking we mention that in our approach an atom is the structure which encapsulates the description of an entity.

A concept makes the clear distinction between the semantics (meta-level) of the entities of a business-model and their reification (reification-level). As a result there are a couple of very positive consequences:

- the maintenance of the semantics (updating and redefining of the semantics) deals only with concepts;
- the support for reuse of the semantics in other (closely related) business models; and
- the model transformation which is one of the key points of our approach.

The semantics of a business-model stored in a concept are reified through a set of hypergeneric parameters and characteristics [2] (which form the meta-information) and a set of actions (which perform treatments on the entities according to their meta-information). The identification of the parameters and characteristics and their possible values is the job of the meta-programmer which addresses the know-how of the business-domain.

The hypergeneric parameters customize the behavior of the entities (it reffers to generic atoms – see section B., and not their instances) of a business-model. Their role is to capture and express the properties which compound the definition of the generic entities. A parameter expresses a basic type property, e.g. a boolean or an integer value, an enumeration, a tuple type or a collection of values. A characteristic expresses a property whose value is defined by an atom or a set of atoms (enumeration, tuple or collection). In order to describe the behavior of a generic entity the programmer has to set those values. For example, a business-model built to encapsulate the structures (entities) and semantics of an object-oriented programming language may define *parameters* like: *cardinality* which expresses if there is simple or multiple inheritance, *generator* which specifies if the given entity can create or not its own instances; or *characteristics* like: the collection of valid kinds of classifiers for a given type of inheritance.

Actions are “first-class” entities addressed by concepts in order to dynamically manage the behavior of atoms according to their meta-information. The body of an action encapsulates the execution which can be performed by that action. The execution of an action depends on:

- querying the parameters and characteristics of the generic atom to which the action is attached;
- a set of invariants, preconditions and postconditions;
- an optional set of aspects;
- additional information provided by the meta-programmer.

An action must be completely independent from the application related to the business model. Therefore a typical scenario is that the behavior of a given application relies on the semantic model, that is to say call those actions or query hyper-generic parameters.

Thanks to AOP [7] paradigm, it is also possible to insert new concerns (we call them aspects), with respect to the semantics of business-model. This is completely independent from the category of visit-entities [11] from the potential applications (see section C.) and they were implemented in order to easy add new pieces of behavior which are orthogonal to the semantics.

We arrived at the line of demarcation between semantics (*the meta-level*) and data of a business-model (*the reification-level*). As we anticipated in the previous section, an atom is the reification of entities of a business-model. Identifying the atoms of a domain is an important task of a programmer.

B. SmartModels Reification-Level

The description of the business-model entities relies on well-known concepts that may be found in most programming languages or meta-models. In SmartModels meta-model, the definition of an atom, the structure which supports the description of an entity, it is very close to the MOF [16] "class" notion. However, the concept of class in MOF is, from our point of view, too much related to programming languages whereas business models require a more abstract concept (that is why SmartModels defines the meta-level).. But at reification-level, the features provided by MOF to describe the contents of a class (such as attributes, operations, generalization relationships) are sufficient to define most of the reification of an entity.

The designer of a business model may create atoms either for improving the structuring and factorization of information within the model hierarchy, or for describing atoms which have instances within applications. SmartModels provides a way to address those two issues; MOF does it through the notion of abstract class. If it means that the class must have at least an abstract method or that all the methods must be abstract, then we believe that this mechanism is not sufficient. In particular, some applications may be interested by some atoms whereas others are not; it is not the same thing to say that whatever is the context of use, one atom may not have instances because it is only partially defined. We believe that a more accurate information according to the atom status will improve the readability of the code produced by generators, and the facilities that may be provided or not to the programmer of application according to it. The interest to be able to associate different status with an atom is even greater if the business model may import atoms from another business model.

Although not all atoms use this facility, each atom has its meta-information in the corresponding concept. An atom is seen as an instance of its concept (see figure 1). However, there are two axes of co-ordinates that we use to distinguish between atoms:

- atoms which are generic or not. The support of generic entities (generic atoms) is an important issue for business models. The genericity is a reflection of the semantic-level which specifies if the meta-information of a given entity has or not parameters and characteristics. A business-model designer may define entities which need semantic information (which becomes part of an atom definition) and we call them "generic atoms". There are atoms which do not need additional customization besides their reification (their behavior does not depend on parameters) and we call them "basic atoms" or "atoms without parameters"; and
- atoms which have instances within applications or not. The generic atoms may or may not have instances at the application-level. We saw that MOF [16] makes this distinction through the notion of abstract class. According to the arguments from the previous paragraph, that is why SmartModels has the notion of *derived* atom (see figure 1) which is an instance of a generic atom obtained through relevant combination of values associated with the sets of characteristics and parameters which participate to the definition of its *generic* atom.

To exemplify we turn again to the case of an object-oriented language. Let us take an example of one business model which is dedicated to record both the structures and semantics of Java programs. Possible applications with respect to this model may implement functionalities of programming environments (metrics, various wizards or editors, etc.). Possible atoms of this model represent, for example, *attribute*, *method*, *method parameters*, *modifiers*, etc.. But the most interesting ones deals with the different kinds of classifiers and relationships (aggregation-like or inheritance-like). Most semantics may be encapsulated within classifiers and relationships and other atoms mentioned above may have a very minimal semantics mostly represented by their reification. This is possible because they are driven by the semantics associated with classifiers and relationships. In fact, there are several kinds of classifiers (e.g. *class*, *inner class*, *interface*, etc.) and relationships (e.g. *extends* between interfaces, *extends* between classes, *implements* between one interface and one class) in this business model. Then it is meaningful to be able to record their definitions as generic atoms (One generic entity for modifiers, one for inheritance-like relationships and one for aggregation-like relationships). All this properties are recorded in their meta-level through parameters.

Therefore, the genericity comes from a set of hyper-generic parameters and a set of characteristics which records the differences and the commonalities between all the foreseen derived entities (This is the term which is quite often used in the state of the art, to refer instances of generic entities; e.g. all the Java classifiers). Intuitively, generic atoms are quite similar to the concept of generic class in the Eiffel language and derived atoms are obtained through the relevant combination of values associated with the sets of characteristics and parameters which participate to the definition of the generic atom.

In [13, 15] we explained why we chose to use generic atoms instead of inheritance relationships for modeling the atoms and there are other interesting issues concerning them. Applying appropriately the SmartModels principles described in the previous sections should lead to a much more effective application building process with SmartFactory. The next paragraphs address the description of applications (and we arrived at application-level in SmartModels) which capitalize the atoms of the business model. As it has been mentioned earlier, we can distinguish two kinds of application:

- those which describe model transformations and
- those which query, compute and update the instances of the business model.

C. SmartModels Instance-Level

At this point, it is straightforward that the specification of those applications will slightly differ from classical object-oriented applications, even if both rely on the object paradigm.

Intuitively, building an application it is a process which consists of a set of traversals of the graph of atoms corresponding to a business-model. During this traversal, the behavior contained in application facets are performed sequentially. While these facets are processed, it is possible to trigger the execution of aspects which allows to integrate orthogonal services. The reification of both the business model and the application is handled by the meta-object protocol which contains also additional functionalities.

A type of traversal is the main entity which influence upon the way an application must be developed and we call it *facet*. The organization by facets of an application draws from SOP [5] and ASoC [7].

A *facet* represents one concern of the application with respect to the business-model. This is a vertical cross-cutting (this is itself defined as an independent business model, so that it may also be associated with a DSL) of the application whereas inheritance relationship would provide an horizontal cross-cutting which introduces several levels of abstraction into the business model or the application. The model supports hierarchies of atoms, concepts, visit-entities, facets and more generally of any first-class entity. Each facet corresponds to a part of the treatment to be processed on one entity. Typically one facet of a given application would rather address the same set of atoms as the other facets (even if there is no constraint).

III. SMARTFACTORY – THE PROTOTYPE

The SmartFactory prototype is built in the framework of Eclipse Platform and it is the first step of the research conducted in the Domain-Driven Development framework [4]. It is a development environment generator that provides a structure editor and semantic tools as main features. It was built on Java and XML technologies as a research project in the I3S laboratory from Sophia-Antipolis, France. Therefore it offers support for designing of new software development environments for programming languages as well as domain specific languages defined with XML.

Eclipse Platform, which is the base of SmartFactory's development, is designed for building integrated development environments (IDEs). It also makes use of a couple of Eclipse Tools Projects such as: Eclipse Modeling Framework (EMF) - an open source code generation tool, capable of creating complex editors from abstract business models (OMG's PIM); Graphical Editor Framework (GEF) - designed to allow editing of user data, generally referred to as *the model*, using graphical rather than textual format; a set of code definition templates defined in a template language called Java Emitter Templates (JET); Eclipse Rich Client Platform (RCP) – a new proficient way to build Java applications and others. These tools were very helpful to add value such as including a GUI for writing a model, automated code generation and automated creation of rich client applications.

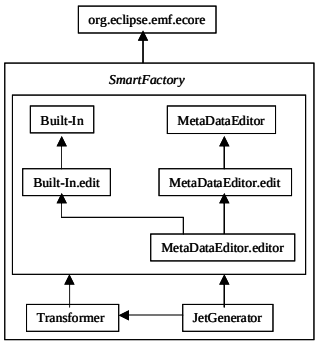

Figure 2. SmartFactory Plug-ins

There are seven plug-in Java projects which work together to implement the SmartModels MOP's principles and rules. Table 3 presents each one of them with the role they have in the approach, the Eclipse features that they use and the design choices we have made in order to build them.

Figure 2 presents the SmartFactory plug-ins architecture which highlights the links and the dependencies between them. From the very beginning it is important to observe that all the plug-ins make use of EMF Ecore (`org.eclipse.emf.ecore`) tool plug-in.

TABLE 1
DESCRIPTION OF SMARTFACTORY PLUG-INS

Plug-in Name	Description
Built-In	- it is the kernel of SmartFactory prototype; - it implements the Meta-Object Protocol approach; - it reifies the SmartModels entities.
Built-In.edit	- it contains EMF content provider classes to describe entities using the editor.
MetaDataEditor	- it represents the SmartFactory meta-data editor; - it customizes the EMF ecore entities in order to support the SmartModels entities specific properties; - it implements the SmartModels methodology to describe a model.
MetaDataEditor. edit	- it contains EMF content provider classes to describe the meta-data editor specific entities for the editor.
MetaDataEditor. editor	- it is the GUI of the meta-data editor; - it provides a wizard and EMF panes to edit a SmartModels model; - it is an Eclipse RCP application.
Transformer	- it performs a model transformation from the meta-data editor format to an EMF ecore format so we can reuse the EMF.CodeGen to leverage the code-generation process; - it uses annotations containing Java pure code to add the SmartModels approach value to EMF entities.
JetGenerator	- it updates the EMF.CodeGen to take into account, when generating code, the SmartModels specific annotations attached by the Transformer to the EMF ecore model.

The heart of the *SmartModels_MetaDataEditor* plug-in is again an EMF ecore model file. Using just EMF framework (its UML diagram editor) we do not have all the tools to describe all the particularities of SmartModels' entities so we needed to create our own editor. However, in order to reuse this flexible platform we decided to enriched the sample EMF ecore editor with support for SmartModels entities.

It is also important to see the SmartModels methodology to describe a business model. This is a five-step process:

1. to identify and to specify the basic atoms of the model,
2. to identify the generic atoms,
3. to define the criteria of genericity (the hypergeneric parameters) - typically this is a step that must be performed by an expert of the domain. It represents a part of the knowledge of the business model;
4. to specify the actions attached to generic and non-generic atoms,
5. to specify the instances of the generic atoms (derived atoms).

The three last steps deals with the specification of the meta-level (the concepts).

As a result we may conclude that there are three main entities that we have to define in a SmartFactory model:

- to elaborate the list of *Atoms*;
- for those who are generic to add their semantic information (the hypergeneric parameters, characteristics and actions) in the list of *Concepts*;
- to compose the list of the *DerivedAtoms* setting their semantic values.

Therefore we created the *SmartModelsEditor* which is the root of the editor and acts as a database holder of SmartModels entity reifications. It is the container of this three lists and has only one constraint: the list of Atoms can not be empty. The root implements the `org.eclipse.emf.ecore.ENamedElement` interface and this means it inherits all the properties of an EMF EObject and we also can add annotations and the name of the model.

For each of the three main entities of our editor we created a correspondent model object and we named them after the original entities adding the suffix "*Editor*". They all have `org.eclipse.emf.ecore.EClass` as a super-type and this choice has many advantages:

- we can benefit from the EMF (UML oriented) framework which is currently under a rapid development and it is more than likely that we will have more rich models in the future;
- it saves us of a lot of work to build a representation of the entities of an object-oriented programming language;
- it draws the modeling phase closer to the implementation language (which has to be an OOPL);
- the programmer is free to add other attributes, references or methods that it may help him describe better the model entities.

The problem is that the editor saves the resources in an XML encoding stream, but not ecore format because our entities add more information to the standard ecore entities. Therefore, in order to use the EMF CodeGen for generating code for SmartModels models they need to be transformed according to the ecore format.

This role is accomplished by the SmartModels Transformer Plug-In (from now on we will call it "*the Transformer*") which can be found as a runtime library "`SmartModels_Transformer_PlugIn.jar`" in the SmartFactory framework. This plug-in makes a contribution to the menu bar which has the same name as the plug-in and adds the action (called *Transform*) that will do the job.

The architecture of the meta-model of the Transformer has two parts:

- a set of six components each one of them dealing with a part of the transformation;
- a hierarchy of classes which help the transformer to handle the different types of model serialization.

In order to run the Transformer a user has to specify two sources: the Built-In ecore model file (the kernel of SmartFactory), the Meta-Data Editor model file (the output of the editor) and one target: the ecore resource where the model is stored after transformation.

To ease the utilization of the Transformer (if a user needs to run it as a standalone plug-in) we designed a *wizard* similar to the EMF editor (see figure 3) where he specifies this three files before running the Transformer.

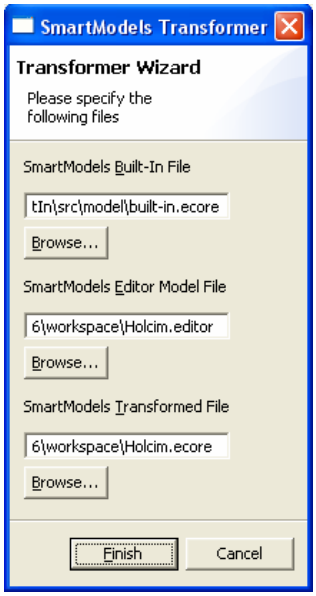

Figure 3. The SmartModels Transformer

Afterward, the code-generation process follows the same rules like for a standard EMF ecore file and gain all the advantages to reuse from such an evolutive platform like Eclipse. We hope you will understand more at the presentation of the prototype which will accompany this article in the conference.

ACKNOWLEDGMENT

This article is a synthesis of principles and practices that were born in the midst of a joint research team we are part of: the team from the Laboratoire Informatique Signaux et Systemes de Sophia-Antipolis, France (I3S: `http://www.i3s.unice.fr`) which belongs to Institut Universitaire de Technologie de Nice et de la Cote d'Azur, department Informatique, under the coordination of professor

Philippe Lahire, and the team from Universitatea "Politehnica" din Timişoara, România, Faculty of Automatics and Computers, (`http://www.cs.utt.ro`) under the coordination of professor Ioan Jurca.

CONCLUSIONS

We have to acknowledge a very important principle in software engineering: in the world of software everything evolves: technologies, methodologies and applications.

We believe that in order to provide an approach centered on models, which capture the know-how of a domain, it is of primary importance to ensure the independence between both the model and the software platform and between the model and the possible applications. This research report promotes the idea that model-oriented programming is a better approach to solve this new challenges.

The first two chapters introduces SmartModels, an approach centered on models of the framework of Model-Oriented Programming. They present its principles, basic entities and main elements when defining a model, which aim to match the requirements of an approach centered on models. A second contribution is the third chapter which address the paradigm of how to practically implement this approach through the proposal of SmartFactory prototype (it deals with important implementation issues based on Eclipse Platform) which is an interpretation and validation of SmartModels.

PERSPECTIVES

The perspectives are twofold. Firstly, to experiment this approach for the description of various business models and their applications. We started to investigate the business models of Romanian companies. The objective is to get feedbacks in order to improve the expressiveness of SmartModels – how to ease the job of a meta-programmer to describe a model, as well as a better automation (in SmartFactory prototype) of:

- the generation of the behavior, and
- the semantics transformation of both models and applications when they evolve toward another model or application.

Secondly, we want to improve the expressiveness of the models [17] for the description of derived atoms and applications, and then to implement them with SmartFactory. Through the definition of those models which are dedicated to enrich the meta-model itself, we aim to improve the quality and the percentage of the code automatically generated so a software company can gain competitive advantages like:

- preserving company investment (future legacy code);
- following rapid technology evolution;
- reacting faster to technology changes;
- improving productivity.

REFERENCES

[1] Isabelle Attali, Carine Courbis, Pascal Degenne, Alexandre Fau, Didier Parigot, Claude Pasquier and Claudio Sacerdoti (May 2001), SmartTools: a development environment generator based on XML technologies, *XML Technologies and Software Engineering*, Toronto, Canada, ICSE'2001 workshop.

[2] Pierre Crescenzo, Philippe Lahire (2002), Using both specialisation and generalisation in a programming language: Why and how?, vol. 2426, *Lecture Notes in Computer Science*, pages 64-73.

[3] Krysztof Czarnecki and Ulrich W. Eisenecker (June 2000), Generative Programming: Methods, Techniques, and Applications, Addison-Wesley.

[4] Krzysztof Czarnecki and John Vlissides (2003), Domain-Driven Development, Special Track *OOPSLA'03*, `http://oopsla.acm.org/ddd.htm`

[5] William Harrison, Harold Ossher (October 1993), Subject-Oriented Programming – A critique of pure objects, *Proceedings ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, ACM Press, pages 411-428.

[6] Georges Dyson, Darwin Among the Machines, Allen Lane publisher, 1997, page 9

[7] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Marc Loingtier, John Irwin (June 1997), Aspect-Oriented Programming, *ECOOP'97 – Object-Oriented Programming 11th European Conference*, Jyväskylä, Finland, vol 1241, *Lecture Notes in Computer Science*, pages 220-242, Springer-Verlag.

[8] Philippe Lahire, Didier Parigot, Carine Courbis, Pierre Crescenzo, Emanuel Ţundrea, *An Attempt to Set the Framework of Model-Oriented Programming*, 6th International Conference on Technical Informatics (CONTI 2004), Timişoara, România, May 27-28, 2004, Proceedings Periodica Politechnica, *Transactions on Automatic Control and Computer Science*, Vol. 49 (63), 2004, ISSN 1224-600X.

[9] Object Management Group (March 2000), Meta Object Facility (MOF) Specification, Version 1.3, Technical Report, OMG.

[10] Object Management Group, Model-Driven Architecture, http://www.omg.org/mda

[11] Jens Palsberg, Barry Jay (August 1998), The Essence of the Visitor Pattern, *COMPSAC'98, 22nd Annual International Computer Software and Applications Conference*, Vienna, Austria.

[12] Clemens Szyperski (1998), Component Software: Beyond OOP, ACM Press and Addison-Wesley.

[13] **Emanuel Ţundrea** (April 30, 2004), Modeling Complex Software Systems: SmartModels - An Approach For Developing Software Based On Models, *Research Report as a part of the doctorate program research*, "Politehnica" University of Timişoara, Automatics and Computer Science Faculty, supervisor prof. dr. ing. *Ioan Jurca*

[14] **Emanuel Ţundrea**, P. Lahire, D. Parigot, C. Chirilă, D. Pescaru, (May 25-26, 2004), SmartModels – An Approach For Developing Software Based On Models, *1st Romanian - Hungarian Joint Symposium on Applied Computational Intelligence SACI'2004*, Timişoara, Romania, ISBN 963-7154-26-4, pages 231-240

[15] **Emanuel Ţundrea**, Ioan Jurca (May 13-16, 2004), Proficiencies of a new approach in software engineering: the Model-Oriented Programming, *2004 IEEE-TTTC International Conference on Automation, Quality & Testing, Robotics* ***AQTR 2004*** *(THETA 14)*, Cluj-Napoca, Romania.

[16] Object Management Group (March 2000), Meta Object Facility (MOF) Specification, Version 1.3, Technical Report, OMG.

[17] PierreCrescenzo, PhilippeLahire, **Emanuel Ţundrea**, SmartModels: la généricité paramétrée au service des modèles métiers, **LMO 2006**, pages 151-166, 22-24 March 2006, Nîmes, France