CONTI'2006
The 7th INTERNATIONAL CONFERENCE ON TECHNICAL INFORMATICS, 8 - 9 June 2006, TIMISOARA, ROMANIA

SmartFactory – A Prototype for Model Oriented Software Engineering Based on Eclipse Platform

E. Ţundrea*, P. Lahire**, 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

** Laboratoire I3S (CNRS-UNS)
2000 Route des Lucioles, BP121, Sophia-Antipolis, France
Philippe.Lahire@unice.fr

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. Also the complexity of a system is not usually found inside components or at their interfaces, but errors are more likely to be found in the interactions among software components. 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.***

The Model-Driven Architecture (MDA) project from OMG promotes the use of meta-modeling in order to drive the system's design and implementation. In this context the paper presents:
- ***an approach: it reviews the SmartModels approach briefly introducing its principles, basic entities and main elements when defining a business-model;***
- ***a prototype: SmartFactory which is based on Eclipse platform and its role is to validate the new approach. It addresses the paradigm of how to practically implement MDA principles and rules for software engineering. Therefore, it deals with important implementation issues based on Eclipse Platform.***

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

I. SMARTMODELS – AN APPROACH BASED ON MODELS

SmartModels aims to address in a practical way the MDE [6] principles. It is an approach which integrates these concepts and proposes a way of developing domain-specific software based on models as a more flexible option to the MOF [6] plus UML [6] aproaches. It gains know-how from a previous research also on meta-modeling [5] and together with it we developed a prototype called SmartFactory in order to validate and get feed-backs from possible users.

MDA approach, as OMG[6] established it, has the advantages of stability and platform-independence through defining business functionality and behavior in a base PIM technology-independent way. This means that an approach based on models has many advantages primarily form the design point of view, but also from the future implementation (application coding, management and maintenance). There are many proven examples on developing standards like SQL, GUI builders, HTML or regular expressions.

One of the main problem the companies face today is that even if we have a perfect model the programmers have to make a lot of compromises when trying to implement the model using a specific programming language and mapping to a platform.

SmartModels tries to reduce this gap between the design and implementation and to assure the independence between the model, future family of applications and the technology (which evolves so fast). Through its small kernel and a set of basic entities, it provides a framework to describe models and a software factory to generate code automatically as much as possible. This means that the applications may be re-generated at any time if the model or the technology evolves and also the model instances can drive the behavior of the application at code generation time or at run-time. Therefore, SmartModels applies MDA through reusing the know-how of promissing platforms for building integrated development environments (IDEs) like SmartTools [1] or Eclipse.

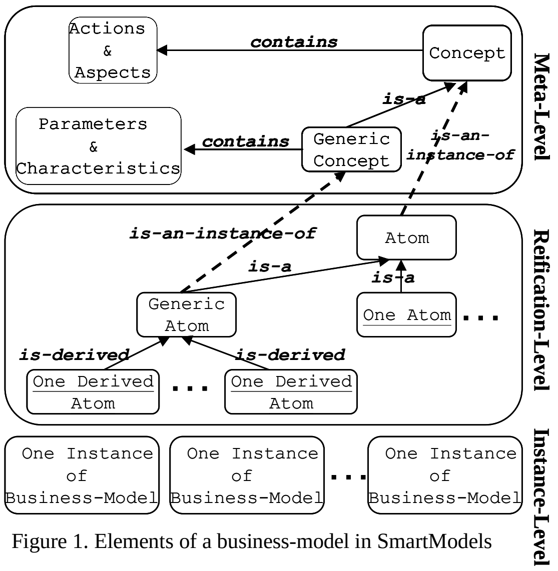


Figure 1. Elements of a business-model in SmartModels

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 [2], to equip, in a modular way, the applications related to this domain ([9]).

Next sections 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. The second chapter presents the process of deploying this models using our SmartFactory prototype.

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. A separation between the meta-level and the reification-level is important in order to clearly identify the relevant common and variable concerns of a family of entities which compound a domain and even identify inter-related concerns of a set of closely-related domains. The meta-level contains the model's entities information (abstract) and rules which engineer their behavior and set the foundation for describing their infrastructure requirements.

The definition of the meta-information of a model is part of the project engineering of the entities of a business domain. Therefore 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.

The main entity which points out the clear separation between the semantics (meta-level) and their reification (reification-level) of the SmartModels model entities is the *Concept*. The choice to encapsulate the meta-information at the meta-level has a couple of very important advantages:
- the support for reuse of the semantics in other (closely related) models;
- the maintenance of the semantics (updating and redefining of the semantics) deals only with the meta-level (concepts);
- the model transformation which is one of the goals of our approach.

The semantics of a business-model stored in a concept are reified through a set of hypergeneric parameters and characteristics [5] (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. Based on the set of parameters corresponding to a concept we can make a differentiation between the families of entities of a domain and based on their values we can distinguish the entities within a family.

The hypergeneric parameters customize the behavior of the entities (it refers 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 those values. A characteristic expresses a property whose value is defined by atom(s) which are defined within the model (enumeration, tuple or collection). The programmer has to set those values to describe the behavior of a generic entity. For example, a business-model built to encapsulate the structures (entities) and semantics of chemical technology product-line may present parameters like:
- *timeBeforeTechnicalInspection* which expresses the time for next technical check up or
- *temperature* which specifies the optimum temperature for a given recipe,

and characteristics like:
- *attachedFilters* which indicates the list of required filters or
- *ingredients* which stipulates the list of substances needed in the chemical process where both the filters and ingredients are reified in the model through Atoms.

Actions are "first-class" entities described 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. This execution usually takes into account the set of parameters and characteristics of the generic atom to which the action is attached and optional can present a set of aspects [4], invariants, preconditions and postconditions.

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 entity of a business-domain is encapsulated in a model hierarchy through *atom* – a structure which holds the entity data and which is similar to the MOF [6] "class" notion and available in most of the object-oriented programming languages (OOPL). It can be used to factorize the data of a domain and has instances within the applications which rely on the given business-model.

In this context it is important to learn about the SmartModels distinction between *basic and generic atoms*. An atom is generic if its meta-information presents parameters and/or characteristics. If an atom does not need additional semantics besides its data-model, we call it *basic* and it will have direct instances within applications.

Now is the time to introduce also 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.

If we go back to the previous example of a chemical technology product-line we can identify:

- basic atoms: filters, ingredients, fuels
- generic atoms: a kiln with hyper-generic parameters and characteristics like those presented at section I.A, with possible action like *checkLastTechnicalInspection* which verifies the corresponding parameter and can send an alarm.
- derived atoms: examples of product-lines with kilns having different technical specifications - like what type of fuel uses or which filters have to be connected to: i.e., a white cement kiln derived atom is a kiln atom with the following meta-information:
    - temperature (fast heated up): 800^0C.;
    - fuleType (minimum): an enumaration of tuple type [8] of COAL (120Kg/h), COKE (35 Kg/h), and TIRE (7.7 Kg/h);
    - attachedFilters (mandatory): O_2 and NH_3.

Actions will check the conformance of the derived atoms' parameter values during the industrial processes implementing the constraints (fast heat up, minimum or mandatory).

C. SmartModels Instance-Level

At this point we can see the benefits of the SmartModels approach. A possible application to the example mentioned above can be a software system which controls and adjusts the performance of a robot which coordinates a conveyor of a product-line like that.

Briefly, building an application consists in writing a set of *facets* – a set of traversals of the graph of atoms corresponding to the business-model. A facet relies on the Visitor pattern [7] and represents a thread of the execution of the application defined by functionality provided by the behavior of the atoms that compound each part of the visit. Certainly, the sustenance for the behavior of the atoms is given by the meta-level (its semantics) and reification-level (data modeled in the atom or derived-atom).

Therefore, a facet integrates the model with the source-code of an application in a very oriented environment: it can have access to the meta-information (this proves the openness of our platform), but it is not always neccessary (this reduces the complexity: a facet just implements a scenario and the handling of the meta-information is the job of the actions in concepts).

This chapter summed up the design of our approach. Next chapter presents SmartFactory – a practical implementation of SmartModels principles.

II. 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 [3]. It is a development environment generator that provides a SmartModels entities' editor, tools and features to describe semantic information. It was built on Java and XML technologies as a research project in the I3S laboratory from Sophia-Antipolis, France. Therefore it offers support to design new software development environments for programming languages as well as domain specific languages defined with XML.

Eclipse Platform, which is the support 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 commercial quality Java programs to be used in non-Eclipse IDE, 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.

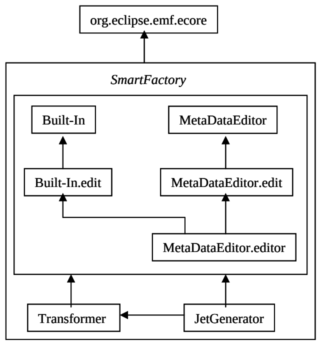


Figure 2. SmartFactory Plug-ins

There are seven plug-in Java projects which work together to implement the SmartModels MOP's principles and rules. Table 1 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:

- the Built-In and Meta-Data Editor use EMF ecore both for their design and implementation;
- the Code-Generator uses EMF.CodeGen for automation of the code-generation process for the SmartFactory transformed model;
- the Transformer deals with EMF ecore model transformation. From the design point of view it did not really need to be described using ecore, but is still used for reasons of unified development of the prototype and for conformance with the way EMF generated code for the other plug-ins.

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.

A. The Built-In Plug-In

The Built-In plug-in represents the starting point of the implementation of SmartFactory. It is the reification of all the entities of SmartModels and it can be distributed as a library jar file called *"SmartModels_BuiltIn.jar"*. In order to add value through our Meta-Object Protocol (MOP) approach a part of the code of this plug-in adapts EMF Ecore model.

MOP information is implemented as a Java static code and it automatically and recursivelly updates the SmartModels database of MOPs. Therefore, the decision was to forbid the *mop package generated* – MopFactory, to create instances of SmartModels_MOP or database objects.. By default, EMF generates a factory for each package from the business-model ecore file. In our approach the creation of *mop objects* is exclusively the job of the database and it provides a full set of methods in order that it can be queried by the meta-programmer. Other factories are also customized to support singleton classes (i.e. concepts).

It is also important to underline a major advantage of this database due to the direct access of each entity which has MOP information to its corresponding concept. This is implemented through a method which is redefined in each class in order to return the correct sub-type of concept thanks to the new feature in J2SE 5.0 that allows covariant return types. In this way, for each entity you can know exactly who is the corresponding concept and manipulate the semantic information (query the parameters/ characteristics or execute actions).

As a conclusion, the SmartModels_Built-In plug-in encapsulates the reification of the SmartModels entities and it produces a set of Java classes for our core model adapted in order to conform to the SmartModels principles and rules. Now we need to provide for our approach an editor in order to be able to ease the description of the business-models and then the creation of their applications.

B. The Meta-Data Editor Plug-In

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 enrich 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 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 in order to have a valid model. 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 (using other Eclipse platform features).

By default, the EMF generated editor provides extension points for all non-empty packages in the ecore model. In SmartFactory there is no meaning to describe some entities independent from a model. That is why the editor database – *SmartModelsEditor*, is the only valid container for a SmartModels model and all the other extension points are suppressed.

C. The Transformer Plug-In

One of the main principles of SmartFactory software factory is that it aims to maximize the use of the tools provided by Eclipse framework. The Meta-Data Editor is generated by the EMF CodeGen.Editor plug-in and we also want to reuse this code-generator for our business-model. In this way we can regenerate and reuse all the evolving features that EMF will provide and we can drive the evolution of SmartFactory at least at the speed of Eclipse Tools development.

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. Thus, the transformer performs an adaptation on our editor output to construct a new customized ecore file representing the target business-model. Then the last step in order to generate a pure Java code, but which reflects the SmartModels approach, is to use the JetGenerator plug-in also mentioned in Table 1 and described below.

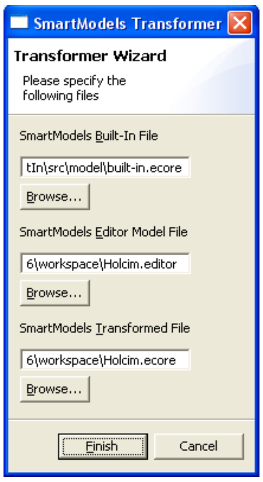
The architecture of the meta-model of the Transformer has two parts:
- a hierarchy of classes which help the transformer to handle the different types of model serialization;
- a set of six components each one of them dealing with a part of the transformation (Atom, Concept, Derived-Atom, Characteristic, Hyper-Generic Parameters, Action). It does not matter the order of launching them, but it it important to run all of them on a model in order to have a valid transformation.

That is why in the next version of the SmartFactory prototype we plan to better customize the use of the Transformer and thus we will add on top of it a "manager" which will organize the component action (now this role is taken by the plug-in default action).

The last, but very important mention is the way the Transformer deals with SmartModels features that the EMF tools do not provide (we already assumed that the ouput transformed model has to be an ecore model). The most significant limitation SmartFactory has to deal with when using EMF tools and Java programming language is that there is no support for the meta-level (the semantic information). SmartFactory makes the compromise to solve this by:
- using the Transformer plug-in to add `org.eclipse.emf.ecore.EAnnotation` in order to encapsulate the semantic information. In actual fact these comments encapsulate Java pure code generated by the Transformer (in this way we are prepared to change this approach for the day when EMF and Java will provide support for meta-level);
- using the code-generator plug-in to adapt the EMF.CodeGen in order to take into account the annotations made by the transformer while generating the code.

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.



Please specify the following files

SmartModels Built-In File: tIn\src\model\built-in.ecore

SmartModels Editor Model File: 6\workspace\Holcim.editor

SmartModels Transformed File: 6\workspace\Holcim.ecore

Figure 3. The SmartModels Transformer

D. The Code-Generator Plug-In

Afterwards, 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.

As earlier mentioned, the code-generator plug-in has the role of adding value to the EMF ecore code-generator in order to take into account the SmartModels specific annotations attached by the Transformer. It does not represent a phase in developing with SmartFactory (like writing a business-model with the Meta-Data Editor or transforming it), but it reuses the EMF.CodeGen to support particularities of our approach. In order to clearly separate the entry points where SmartFactory updated the JET templates, they are organized as distinct emitter parts, each one of them in distinct files which can be included in the EMF templates when needed. This brings the advantage of clearly identifying the parts to adapt when the platform evolves.

We hope you will understand more at the live presentation of the prototype which will accompany this article in the conference.

CONCLUSIONS AND PERSPECTIVES

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 article promotes the idea that model-oriented programming is a better approach to solve this new challenges.

These ideas have been around for a couple of years, but today there is no major vendor which gets behind OMG's MDA initiave and make it happen in software development. Our approach together with the prototype want to form a possible and feasible way to apply this principles. It can happen in JetBrains or Eclipse and based on this last experience we propose a way to address meta-modeling issues extending their know-how.

The first chapter introduces SmartModels, an approach centered on models of the framework of Model-Oriented Programming. It presents 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 second chapter which addresses 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.

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 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] I. Attali, C. Courbis, P. Degenne, A. Fau, D. Parigot, C. Pasquier and C. Sacerdoti (May 2001), SmartTools: a development environment generator based on XML technologies, *XML Technologies and Software Engineering*, Toronto, Canada, ICSE'2001 workshop.

[2] K. Czarnecki, U. Eisenecker, June 2000, Generative Programming: Methods, Techniques and Applications, Addison-Wesley.

[3] K. Czarnecki, J. Vlissides (2003), Domain-Driven Development, Special Track *OOPSLA'03*, `oopsla.acm.org/ddd.htm`

[4] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, M. Loingtier, J. 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.

[5] P. Lahire, D. Parigot, C. Courbis, P. Crescenzo, E. 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.

[6] Object Management Group, Model-Driven Architecture (MDA) and Meta Object Facility (MOF) Specification, www.omg.org/mda

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

[8] E. Ţundrea, P. Lahire, D. Parigot, C. Chirilã, 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

[9] P. Crescenzo, P. Lahire, E. Ţundrea, SmartModels: la généricité paramétrée au service des modèles métiers, LMO 2006, pages 151-166, 22-24 March 2006, Nîmes, France