MDE for Enterprise Application Systems

Alfonso Díez[1], Nga Nguyen, Fernando Díez[2] and Enrique Chavarriaga[1]
[1]*B2T Concept, Fuentes 10, Madrid, Spain*
[2]*Computer Science Dept., Universidad Autónoma de Madrid, Madrid, Spain*

Keywords: Model Driven Engineering, Domain Specific Languages, MDE Integrated Development Environment, Model Execution.

Abstract: Model Driven Engineering (MDE) has been widely researched as a solution for the complexity of software development over last decades. However, it is not widely adopted efficiently in industry. In this paper, we identify two main challenges prevent MDE from industrial adoption: the first one is capturing dynamic behaviours from real problems in human organization into formal models; the second one is the lack of an integrated development environment (IDE) which can have a fast and reliable model execution. In order to address these two challenges, we have worked during the last ten years in the area of Enterprise Application Systems based on Business Models formalisms. We have combined different technologies from the MDE context such as multilevel meta-modelling, domain specific model languages (DSML), state machines and model interpreters. The result is that we have created a large set of commercial products based on a common model based platform, which we are currently applying in many business areas. This paper describes the most representative concepts and contributions of our work to the development of MDE.

1 INTRODUCTION

This paper shows our experience in the development of Enterprise Application Systems since 2002. There are many open issues in the area of modelling business solutions, which are hot topics in the production of research papers and in scientific and business conferences. We have a wide experience owned during the last decade, and we addressed some of the most challenging problems, being allowed to suggest solutions for them.

One of the main challenges of MDE to be adopted by the industry is dealing with the software complexity (Straeten et al.,Baelen, 2009). There have been many solutions which try to solve this problem but very often they introduce more complexity to software. For instance, one way to reduce the complexity in describing problems is to raise the abstraction level. UML, which based on a two meta-level setting (Lara and Guerra, 2012), is widely used to describe systems. However, soon later people realized that it is not sufficient enough to represent complex multi-level meta-models. Therefore, number of research has produced many ad-hoc ways to overcome this problem such as clabjects (Atkinson and Kühne, 2000), METADEPTH (Lara and Guerra, 2010), deep instantiation and model to meta-to-metamodel transformation (Kainz et al., 2011), etc. but those approaches, similarly to the Ptolemaic epicycles, seem to introduce more complexity than producing practical solutions. In addition to the difficulties in representing the complexity of problem themselves are those ones arising from capturing evolution and dynamic behaviours from real problems in human organization within formal model representations. The current models are often only able to capture the static aspect of reality although the reality is the combination of both static and dynamic facets. Some approaches have been introduced but also often are very complicated, such as a transformation engine C-Saw to manipulate models based on the combination of model transformation and aspect weaving (Gray et al., 2006), or higher order model transformation (Cicchetti et al., 2008). A quite interesting approach by (Trojer et al., 2010)use state machines on model elements to support change-driven model evolution. However, as stated in the paper, this approach is still just a prototype and not yet reflects the effects of changes in model elements of meta-model.

Another challenge in MDE is the lack of simple

Diez A., Nguyen N., Díez F. and Chavarriaga E..
MDE for Enterprise Application Systems.
DOI: 10.5220/0004311502530256
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 253-256
ISBN: 978-989-8565-42-6

ways to transform models into executable solutions of any functional or technical complexity without further programming. There have been a number of approaches to solve this problem, however most of the tools only support a subset of a family languages, for instance code-generator generator which only supports a family of modelling notation (Prout et al., 2010). In addition, for defining domain specific meta-modelling, no general framework languages have been developed within integrated MDE (Lara and Guerra, 2012). Therefore, a way to generate models into business solutions, applicable in every domain, is still an open issue.

2 SCOOP - A MODEL TO DESCRIBE ENTERPRISE PROBLEMS

The problem we face daily is how to create formal models of human organizations (i.e. governments, enterprises, etc.). Any organization has to be described in three main layers: resources, processes and knowledge, each of one having static and dynamic components. In Figure 1, we describe our proposal for a **theory** on Enterprise Application Model Driven Solutions. In this sense, we consider a theory as the set of hypotheses whose consequences are applied to the problems we are dealing with. In the following we are going to define the components being part of the solution proposed.

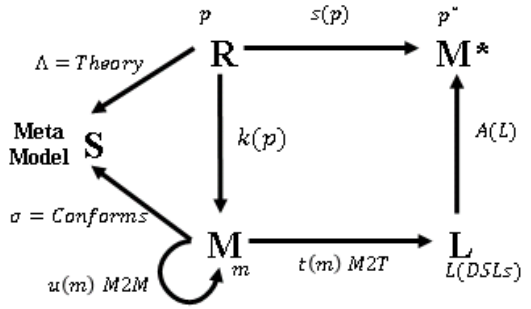

Figure 1: SCooP model.

By **R** we mean **the reality space**, which contains a set of problems *pp*. For the sake of clarity, let restrict *R*R to describe the specific domain D of problems that can be expressed in written form. Therefore, problems in D can be as large as an Enterprise, or can be narrowed to more specific ones such as Procurement, Document Management, Systems Orchestration, or even more narrower as Management of Diabetes, to name a few.

We define a **meta-model S** as a mental scheme or abstraction about the selected domain D. Because of the selection of D, any such conception will be a construct created by human beings and wrapped into semantic structures that allow them to communicate and represent it verbally or in written form. These semantic structures can be formalized by schemas of classes-*c* and relations-*r*. We call this meta-model SCooP (Standard for Cooperative Processes) which describes human organizations, both in their static and dynamic aspects.

The abstractions of the reality can be structured in layers. For example we can create a first high-level abstraction about general business concepts, and later a more specific one, such as workflow concepts. Workflow is a meta-model about sequences of tasks and decisions for human individuals or groups. Given that meta-models are described with classes and relations, the inheritance of classes produces a partial order, which gives S a structure of a partial ordered set (a poset).

Models based on SCooP have three central sets of definitions: a static description of the domain involving the model (the conceptual domain model), a procedural manipulation of the domain concepts, and a description of the dynamics (life cycle) of the model. A concept is a class in the conceptual domain model. For each concept c in the domain we define a state model that represents the life cycle of the concept. The life cycle is described by means of a finite state list, together with the transition and evaluation rules. The first ones govern state changes, and the last ones trigger the consequences of those changes. To manage state lists and rules SCooP defines **state machines** to formalize the way life cycles are understood. A consequence of the usage of state machines is the propagation of state changes along the model. This is a very powerful method for representing the dynamic aspects of the reality and managing systems with high inner complexity.

For each problem $p \in R$ there exists a model, $m \in M$,being M is the **space of models**. A model $\boldsymbol{m}$ is an instance of a meta-model S. A model is created by inheriting the basic semantic structure of the meta-model, and by extending it with new properties. The inheritance between classes creates also a partial order in the set of all *o* objects defined in $o \in m$. Every object *o* in *m* has to inherit from a concept *c* in S, because no concept can be defined without a language of reference.

Back again to the elements being part of the meta-model SCooP in Figure 1, we define now the **Space of Domain Languages L**. We have

mentioned earlier that semantic structures arise from meta-models. To these semantic structures we add syntax in order to create different languages specific for the domain (DSML). Using these specific languages, we can describe models in terms of them. Different syntaxes can be used for specific purposes: verbose class descriptions, XML structured schemas, JavaScript procedural descriptions, OCL, JSON or comma separated lists of properties, mathematical expressions, etc. For instance, we have languages to regulate state transitions, to specify the composition of an interface dialog, to bind databases, to compose diagrams, etc. This capability allows us to represent models in terms of DSMLs and, therefore, can transform Models into Languages (M2T) to create Information Systems. In this way, each model m can be transformed to a set of DSMLs that fully represent it. The opposite cannot be true, because correct syntactical expressions in language could not be allowed by meta-model rules, so cannot be reverse engineered from the Text to the Model.

A **System *m**** is an instance of a model *m* (or the union of models) plus an execution architecture and the time dimension. *m** will be composed of instances *o**in the sense that for all *o** in *m**,*o** is an instance of an object *o* in *m*. Additionally, every instance *o** has a unique state, selected from the *o* states list, in a given instant of time. The changes of an object state are defined by applying the state transition rules. The combination of time, states and object instances creates a different global state of *m** for each t.

The system M*. Let consider the problem of describing the meta-model S as an object of Reality. A meta-model can be seen as a model by itself, given that it is a description of the reality by means of a predefined semantic. So we can take S and generate models that describe S and use S as the meta-model of reference, namely m_s. Given that m_s is a model itself, it can be generated as a solution m_s^*, i.e. $s(p) = m_s^*$. This solution is able to represent any model that has been declared under the S meta-model, so it is a collection of models, and can manage the transformation of M in M* in the corresponding DSMLs. All this procedure drives us to a very interesting situation: for each S there exists a m_s^*, that can contain any model which conforms S and m_s^* is able to generate itself.

2.1 Execution Architecture

The Execution Architecture is the set of programs that are able to interpret and execute the DSMLs generated from models. Such architecture can run any model that has been serialized in L, avoiding the codification of any ad-hoc program. Therefore, we build models of the reality we want to codify; then we convert these models to different DSMLs, which can be executed without neither any additional technical effort nor development. We will call Model Engines to the components of the Architecture.

In our common practice we use three Model Engines: E3 for core structures, SABLE for web user interfaces, and an SQL engine (different providers) for relational database management. Other components can be used as needed, depending on the characteristics of the model.

In summary, only one computer system (the set of Model Engines) will solve any Enterprise Information System, composing a binary Solution: the Model Engine plus the DSMLs. The benefits of this architecture are huge: (1) all the of system's development processes are truncated, eliminating technical designs, programming and unit testing and redesigns; (2) scalability, reliability and performance of the Solution is guaranteed by the Engine, not by the Model; (3) Flexibility of the Solution, as the adaptation to new requirements, is guaranteed by the Model, not by the Engine and (4), Model definition is a problem in the scope of Knowledge Management and Representation Techniques, and no in the domain of the technologies. Therefore, any Model that can be defined can be executed, which eliminates the systemic risk in systems development.

2.2 The Scoop Meta-Model and Modelling Technique

The SCooP metamodel has been developed by our company in order to produce Enterprise Application Systems from any business sectors and functional areas. At present, SCooP is able to represent business realities in many different areas such as Banking, Insurance, Health Care, Manufacturing, Services, Utilities, etc. and in functions such as Finances, Customers Management, Operations, Resources Management, Materials Management, Project Management, Procurement, Disease Management, etc. SCooP is divided in a number of subdomains, namely: Conceptual Model, States Model, User Interface Model, Technical Interface Model, Execution Model, Security Model and Data Model. Other meta-models are under analysis.

We call Models of Reference to those models that are used in a common way in most of the Solutions we build, for instance, management of technical functions such as command line operators,

protocols and messaging (email, ftp, etc.), audit model, etc.

As previously stated, a meta-model is an abstraction of the reality. There is a natural knowledge engineering process that will turn models into meta-models, that is based on the experience of the domain analysts. It is interesting to understand how a model can be promoted into a meta-model. This progression is done as we get more and more information and experience about a given domain. The more we learn about a domain, the more able we are to create a theory about that domain.

3 CONCLUSIONS AND FUTURE WORK

The problem that we have considered in the last ten years is how to create a new generation of Enterprise Application Systems with four characteristics: a wide domain on its application, minimum costs in their specification, highest execution reliability and absolute flexibility to adapt to new conditions. Besides this overall set of desirable characteristics, the adoption of part or all of them are not widely efficiently adopted at industry due to a couple of main challenges: the first one is capturing dynamic behaviours from real problems in human organization into formal models; the second one is the lack of an integrated development environment (IDE) and an execution environment which provide a fast and reliable model description and execution. The result of our work is that we have created a conceptual background and a technical architecture that is able to generate any kind of Enterprise Application Systems based on models: the behaviour of the solution is described as a formal Business Model. The transition between models and solutions is seamless, using SCooP to specify models and generate solutions. Solutions are based in two components: a number of DSMLs that describe the model, and a number of standardized systems that interpret and execute those languages.

For the near future, we are in conditions to address, based on the SCooP model and on the tools already developed, to the development of new higher-level meta-models, more powerful IDEs, better interpreters and flexible DSMLs, and inter-model interoperability. In summary, we resolutely step forward a new paradigm of MDE effectively applied to Information Systems applications.

REFERENCES

Atkinson, C. and Kühne, T., 2000. Meta-level independent modelling. *In International Workshop on Model Engineering* at *14th European Conference on Object-Oriented Programming*. Cannes, France.

Cicchetti, A., Di Ruscio, D., Eramo, R, Pierantonio, A. 2008. Automating Co-evolution in Model-Driven Engineering, *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, p.222-231. Munich, Germany.

Gray, J., Lin, Y., Zhang, J. 2006. Automating Change Evolution in Model-Driven Engineering, *Computer*, v.39 n.2, p.51.

Kainz, G. Buckl, C. Knoll, A., 2011. Automated model-to-metamodel transformations based on the concepts of deep instantiation. *In Proceedings of the 14th international conference on Model driven engineering languages and systems*, p. 17-31. Wellington, New-Zealand.

Lara, J., Guerra, E. 2012. Domain-specific textual meta-modelling languages for model-driven engineering. *LNCS 7349, pp.: 259--274,* Springer.

Lara, J., Guerra, E. 2010. Deep meta-modelling with METADEPTH. *In Proceedings of the 48th international conference on Objects, models, components, patterns*, p.1-20. Málaga, Spain.

Mohagheghi, P., Fernandez, M. A., Martell, J. A., Fritzsche, M., Gilani, W. 2009. MDE Adoption in Industry: Challenges and Success Criteria. *In Models in Software Engineering*, pages 54–59.

Prout, A., Atlee, J. M., Day, N., Shaker, P. 2010. Code generation for a family of executable modeling notations. *Software & Systems Modeling*, *11*(2).

Straeten, R,, Mens, T., Baelen, S. 2009. Challenges in Model-Driven Software Engineering. *In Models in Software Engineering. LNCS 5421.* Springer.

Trojer, T., Breu, M., Sarah, L. 2010. Change–driven Model Evolution for Living Models. *In Proceedings of the 3rd Workshop on Model-Driven Tool & Process Integration. p. 73-84.* Paris, France.