MODEL-ORIENTED PROGRAMMING VERSUS COMPONENT-BASED SOFTWARE ENGINEERING

Ioan Jurca[1]
[1]*Politehnica University of Timisoara,*
Faculty of Automatics and Computer Science,
Bd. V. Parvan nr. 2, Timisoara, România
http://cs.utt.ro
ionel@utt.ro

Emanuel Tundrea[2]
[2]*"Emanuel" University of Oradea,*
Faculty of Management,
Str. Nufarului, nr. 87, Oradea, România
http://www.emanuel.ro
emanuel@emanuel.ro

Abstract: Nowadays, companies involved in the development of modern software face several difficulties. One of the most important ones is the continuous evolution of software platforms (C++, Java, .Net, CORBA, EJB, WebServices, XML, etc.). One interesting solution to this problem is the Model-Driven Architecture (MDA) approach from the OMG. It suggests that domain-specific knowledges should be encapsulated in platform-independent business models, apart from the applications. Beyond this answer is the failure of classical development techniques that rely on object-oriented design and programming. According to these remarks, we propose another way to develop software: Model-Oriented Programming (MOP). It is based on the Domain-Driven Development (DDD) track and introduces a macro-level on top of the classical programming entities. It intends to be used for the handling, reuse and evolution of the business know-how and its associated applications. This paper i) presents the concepts of the new paradigm MOP which relies on MDA, aspect-oriented and generative programming; ii) makes a comparison between MOP approach and component-based software engineering; iii) addresses implementation issues of a prototype (SMARTMODELS) which benefits from earlier experiences.

Keywords: software, model, component, web service

1 INTRODUCTION

The object-oriented approach does not provide all the solutions even if it represents a valuable basis for the description of further approaches. This remark can also be applied to component-based software engineering paradigm. In particular, they do not provide a correct answer to the continuous evolution of the technologies: keeping applications up-to-date according to the evolution of technologies is too much time-consuming. We believe that to provide an approach centered on models which capture the know-how, independently from both the software platform and the possible applications is promising.

In this paper, we propose the framework of Model-Oriented Programming with a set of essential entities. We consider them as a first attempt for the definition of the main principles of this approach. A second contribution is the proposal of SMARTMODELS which is one interpretation of this principles. Finally, we want to draw a parallel between our approach and component technology.

2 THE MODEL-ORIENTED PROGRAMMING APPROACH

MOP moves the accent from objects and components to more dynamic and evolving perspective: models. MOP is an original approach for software engineering because:

- it aims to provide a framework for describing models in which to encapsulate the specific knowledge of a domain according to multi-system scope development defined in domain engineering [3];
- it ensures a clear separation between the model and the technologies which makes the model executable by a software platform;
- it incorporates software factories (SMARTFACTORY) which automate as much as possible the code generation and also provides easy and clear entry points in the code for the user to change or update the code;
- it integrates new ideas from AOP [7], SOP [5], IP [12], web servicies [14] and component-based engineering [13] to build flexible, readable and easy maintanable software;
- relies on W3C (*World Wide Web Consortium*) [14] and OMG (*Object Management Group*) [9] standards like XML and DTD for serialization of the models, OCL [10] for assertions, MOF [8] and AST [1] for the meta-model [2] description;
- capture the semantics of a model and proposes a way to handle their treatment.

In order to provide the facilities presented above MOP deals with a set of entities. Many of them are familiar concepts from object-oriented programming and in the next sections we will present them together with their first validation. We will also use the context or the interpretation of our approach called SMARTMODELS.

2.1 MOP's entities interpreted by SMARTMODELS

As a preamble, we can say that the meta-model which allows the description of business models addresses i) the reification of basic entities, ii) the reification of generic entities and their generic parameters, iii) the semantics of the business model which corresponds mainly to the possible values that may be assigned to the generic parameters.

In Figure 1 we propose an overview of the architecture of the meta-model. The semantics of the business model is addressed through the specification of hypergeneric parameters [6], characteristics and actions. All of them participate to the definition of the semantics of business-model entities[1] (whether they are generic or not); but they do not address the description of their instances. Because applications are outside of the business-model, the methods that handle instances of atoms are accessors[2] only[3].

We propose to create a meta-level (*concepts*) in order to encapsulate the semantics. A concept is associated with one or several atoms[4]. This clear separation between the semantics of the business-model and the reification of its entities is very important because it favours i) the maintenance of the semantics (redefining the semantics should only deal with concepts), ii) the reuse of the semantics in other (closely-related) business-models, and iii) the transformation of model which is one of the key-points of model-oriented

[1] We call them *atoms* – see next paragraphs.
[2] They are automatically generated taking into account the type (e.g. if it is a collection or not).
[3] This is the main difference against *actions* which address entities, but not their instances.
[4] It is an approach which is quite similar to the classes and meta-classes of the Smalltalk language.

programming. The main facilities provided by MOF to describe meta-information are just class variables and class methods; from our point of view it is not sufficient.

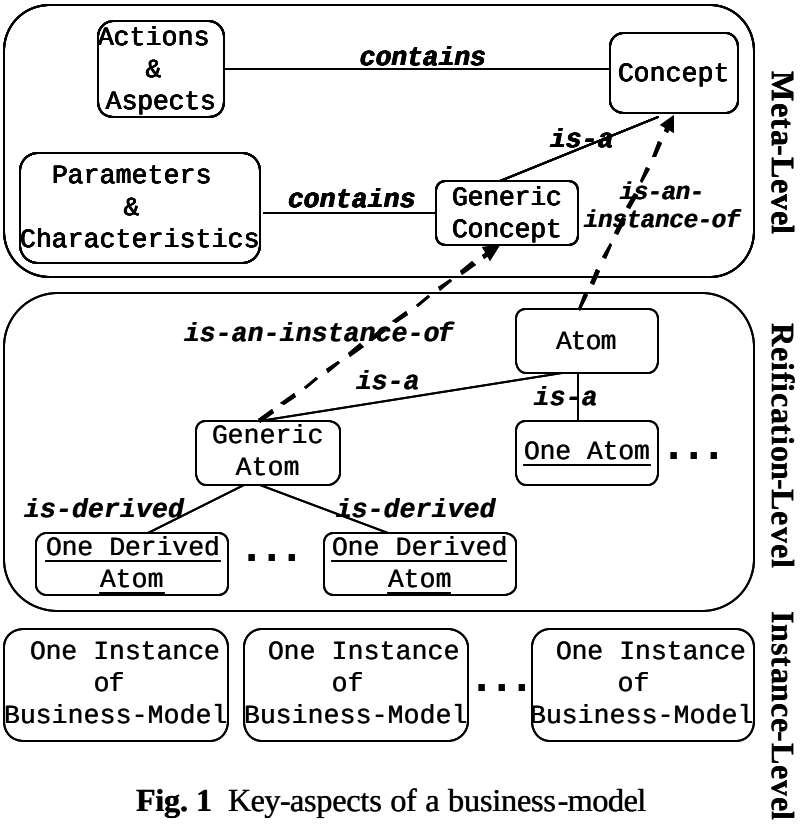

Fig. 1 Key-aspects of a business-model

The description of the business-model entities relies on well-known concepts that may be found in most programming languages. We present them briefly in the context of SMARTMODELS and with regard to MOF [8]. In our meta-model, an *atom* is the structure which supports the description of an entity. The support of generic entities (*generic atoms*) is also an important issue for business-models. 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 can be *an attribute, a method, some method parameters, some modifiers, etc.* But the most interesting ones deals with the different kinds of relationships (aggregation-like or inheritance-like). Most semantics may be encapsulated within classifiers and relationships and other atoms mentioned above may have 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 interface, extends between classes, implements between one interface and one class*) in this business-model [2]. Then it is meaningful to be able to record their definitions as generic atoms[5].

The term "*genericity*" of the "*generic atoms*" comes from a set of hyper-generic parameters and a set of characteristics which records the differences and the commonalities

[5] One generic entity for modifiers, one for inheritance-like relationships and one for aggregation-like relationships.

between all the derived entities[6] (e.g. all the Java classifiers). The definition of an hyper-generic parameter is mainly based on a basic type (it may be an integer, a boolean, an enumeration, a tuple or a collection) and on some additional information. The definition of a characteristic relies on an atom or a collection of atoms (e.g. one kind of classifier records the possible kinds of inheritance-like relationships that it may declare). Intuitively, generic atoms are quite similar to the concept of generic class in the Eiffel language. 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.

From the previous paragraphs results that a significant part of the semantics of a business-model is encapsulated in a few generic atoms. A part of the semantics is captured by parameters, characteristics and invariants[7]. It is a first step but it is still not sufficient to handle the full semantics of atoms. For example, the value of parameters used for the instantiation of generic atoms will affect the behaviour of its derived atoms. It is necessary to be able to specify this behaviour. Each atom, even if it is generic or not, has a meta-level (its concept) where it is possible to define actions.

Actions are methods defined as first-class entities. When an atom is derived from a generic one its execution is driven by the value of the parameters. An action has a signature and assertions[8] defined with respect to the reification of entities and hyper-generic parameters (in case of a generic atom). It may also accept the execution of orthogonal concerns (aspects [7]). An action must be completely independent from the application related to the business model. A typical scenario is that the behaviour of a given application relies on the semantic model: this means to call those actions or query hyper-generic parameters.

SMARTMODELS organizes applications by facets (like in SOP [5]). A *facet* represents one concern of the application with respect to the business-model. The description of a facet relies on the design pattern Visitor [11]. Depending on the requirements of applications, the behaviour related to the visit of one atom may be spread out differently. In a first approximation, one atom of the business model is associated with only one visit-entity. A facet specifies i) the business-model that it addresses, ii) how the business-model is visited, iii) the entities that are relevant according to the objectives, and possibly iv) some additional technologies.

3 MOP VERSUS COMPONENTS

MOP and SMARTMODELS offers practical solutions to a couple of very important problems found in object-oriented and component-oriented programming. In the next sections we want to draw a parallel between our approach and component technology. We will continue to addresses implementation issues of our prototype – SMARTMODELS, which benefits from earlier experiences (SMARTTOOLS) [1].

In the last decade most of the articles and trivia published on this matters seem to point to components [13] as the solution to a new maturity level of the software engineering. Building new applications by combining bought and made components improves quality and supports rapid development because of the enabling of reuse of

[6] This is the term which is quite often used in the state of the art, to refer instances of generic entities.
[7] Like in MOF or UML, it is possible also to define atom invariants.
[8] In our meta-model assertions (invariants, preconditions, postconditions) are described with OCL [10].

software. Component-based software engineering approaches plead for building reusable components that can be plugged together to create new systems.

First of all, finding the equilibrium it is again a hard thing: It is a decision between generality and specialization. General components can be used in many applications, but may be much more than needed. Specific components, on the other hand, are appropriate for a given situation, but then there may be many specific components that the client have to search through to find the one that exactly meets his needs. Some of them may need to be adapted to the particular situation and this can be another problem.

In this context MOP offers the possibility to organize all the entities defined for a business-model in a hierarchy. In this way a model can have in the same time components more general and more specific having the possibility to use or not the advantadges of inheritance for meta-information. At any time a component can be specialized to be adapted for a certain situation using the inheritance. Also, a more specific component can use or not the same semantics as its parents, or it can have different values for its meta-information.

Other problems in software engineering are linked with the systematic inter-play of components. It would be naïve to assume that we can simply select components from a well organized repertoire and after a click we have the final application. In reality many questions arise on how can the abstract interaction of components be described, how can variety and flexibility be covered in the design of a component, how can critical system properties be guaranteed when there are many vendors or how can performance be guaranteed.

MOP handles variabilities and commonalities through meta-information (see sections 1.2). The behavior of an atom is influenced by the values of its parameters and characteristics and both, the actions defined in its meta-level (concepts) and the set of assertions attached to each atom, will check the conformity and support aspects [7]. All the components provided by different vendors will have to conform to the meta-information from the business-model.

All current software component camps are not on a domain-specific standards. Components-based reuse has proved useful in some application domains. For example, libraries of mathematical functions are commonly used. However, in other domains, component-based approach poses problems that obstruct effective reuse. Notably, methods for searching, analyzing and customizing components and integration of components into a working system are not well defined, explained and understood. Our approach on MOP follows the Domain-Driven Development [4] principles and therefore offers a framework for development domain specific applications. Our concern is to define an approach which makes possible to specify any model applied to any domain: object-oriented languages, nuclear factory which produce electricity, etc..

Proliferation of new component technologies is another issue. To obtain a component-based application a developer must choose between at least three component technologies: CCM (CORBA Component Model), EJB (Enterprise Java Bean) or Web Services. In our approach technologies are defined independently from SMARTMODELS. They contain functionalities which allow to define more easily the application. For example, the DOM API is welcomed to manipulate XML representation of the business models. Other information may be added in order to generate visit-entities which fit exactly to the expectations of the programmer. Indeed the source code generation is essential to our approach because it allows him to focus only on the visit-entities that are addressed by the facets and to be assisted for the description of their behaviour.

4 CONCLUSIONS AND PERSPECTIVES

All the trends in software engineering today target the idea of developing software more efficient through reuse. We proposed a different and original way of dealing with this paradigm:"Model-Oriented Programming". MOP encapsulates the know-how of a business-domain in an abstract model and after that quickly and cheap build the applications.

Our perspectives are twofold. Firstly we want to experiment our approach for the description of various business models and their applications; currently we start to investigate the business model of a the French electricity company, EDF. The objective is to get feedbacks in order to improve the expressiveness of SMARTMODELS as well as a better automation (in SMARTFACTORY) of i) the generation of the behavior, and ii) the semantics transformation of both business models and applications when they evolve toward another model or application.

Secondly, we want to improve the expressiveness of the business models for the description of facets, aspects, applications and components, and then to implement them with SMARTMODELS. Through the definition of those business models which are dedicated to enrich SMARTMODELS itself, we aim to improve the quality and the percentage of code automatically generated.

REFERENCES

1. Isabelle Attali, Carine Courbis, Pascal Degenne, Alexandre Fau, Didier Parigot, Claude Pasquier and Claudio Sacerdoti (May 2001), SMART TOOLS: 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 at *OOPSLA'03*, `http://oopsla.acm.org/oopsla2003/files/ddd.html`.
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. George Heineman, William T. Councill (May 2001), Component-Based Software Engineering – Putting the Pieces Together, Addison-Wesley.
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. Object Management Group (March 2000), Meta Object Facility (MOF) Specification, Version 1.3, Technical Report, OMG.
9. Object Management Group, Model-Driven Architecture, `http://www.omg.org/mda`
10. Object Management Group (January 6, 2003), Object Constraint Language, RfP document ad/2000-09-03, Version 1.6, OMG.
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. Charles Simonyi (September 1995), The Death of Programming Languages, the Birth of Intentional Programming, Technical Report, Microsoft, Inc.
13. Clemens Szyperski (1998), Component Software: Beyond OOP, ACM Press and Addison-Wesley.
14. W3C Working Group (February 11, 2004), Web Services Architecture, `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf`