

Baseline-Oriented Modeling: an MDA approach based on Software Product Lines for the Expert Systems development¹

María Eugenia Cabello, Isidro Ramos, Abel Gómez, Rogelio Limón
Universitat Politècnica de València
{mcabello, iramos, agomez, rlimon}@dsic.upv.es

Abstract

This paper presents our Baseline Oriented Modeling (BOM) approach. BOM is a framework that automatically generates software applications as PRISMA architectural models using model transformations and Software Product Line techniques. We follow the Model-Driven Architecture initiative building domain models which are automatically transformed into Platform Independent Models, and then compiled to an executable application (i.e. Platform Specific Models). In order to illustrate BOM, we focus on a specific domain: the diagnostic Expert Systems.

1. Introduction

Nowadays Software Engineering promotes automation in software development and open mechanisms to cope with the complexity of software systems. However, the automatic generation of such systems is only possible if there is a framework that supports the process. The Model-Driven Architecture (MDA) [1] strategy advocates the use of models and platform independence in the software development process as a new way of generating applications automatically.

Furthermore the development of Expert Systems (ES) [2] applications has acquired great importance in recent years and there is a need to provide proper support for them. The Expert System captures the knowledge of experts and attempts to imitate the reasoning processes of such experts when they solve problems in a specific domain. Such systems can be designed as a generic architecture with components that are independent and separate units. However, these

systems are complex because their architectural elements vary.

In order to cope with the variability problem, Software Product Lines (SPL) [3] have emerged in an effort to control and minimize the high costs of the software development process and to reduce the time to market of a family of products. This approach is based on having a generic design that is shared by all the product family members.

In this context, we have developed the Baseline Oriented Modeling (BOM) framework. BOM automatically generates Expert System (as PRISMA architectural models) based on SPL, by using the MDA approach.

In BOM the target platform is PRISMA [4]. The PRISMA architectural model integrates two approaches: Component-Based Software Development, and Aspect-Oriented Software Development. This integration is obtained by defining the architectural elements through their aspects. The PRISMA Architectural Description Language provides expressivity for the specification of software architectures with aspects at a design level. The architectural models can be automatically compiled to .NET executable applications.

Our work integrates these technological spaces: MDA, SPL, the PRISMA framework and Expert System as our case study. The Expert System design separates the inference process from the knowledge domain, and can incorporate several reasoning strategies in order to solve a problem by applying the most efficient of them. We use the MDA approach in order to automatically generate code from models by means of transformation rules and SPL to minimize the variability impact on the cost of the software production.

However not everything can be automated; user interaction is required to provide information to the

¹ **Acknowledgments.** This work has been funded under the Models, Environments, Transformations, and Applications: META project TIN20006-15175-605-01, and the scholarship of the Training University Faculty Program, ref. AP-2006-00690.

system. BOM's user produces a specific Expert System when he/she inputs the variability information of a particular domain into the system by means of domain conceptual models.

The objective of this paper is to present the design and use of BOM at a high abstraction level, by means of Expert System development.

The paper is structured as follows: Section 2 presents our field study (diagnostic Expert Systems). Section 3 comments on variability management in BOM. Section 4 describes the software views, metamodels, models and transformations used in BOM. Section 5 presents the related works. Section 6 presents our conclusions and provides our ideas for future work.

2. Our field study: diagnostic Expert Systems

We have carried out a field study to learn about the variability in the Expert System and we have selected diagnosis as our case study. A number of cases have been studied: medical diagnosis, educational diagnosis, video diagnosis, and disaster victim diagnosis.

2.1 The Diagnosis: domain variability

Based on the analysis carried out in the diagnostic process we can conclude the following:

Diagnosis consists of an interpretation of the states of the entities involved (viewed as a set of properties), followed by the identification of the problems using rules.

There is variability in the diagnostic process (system behavior) and such variability can be described in terms of its features as follows: i) Entity views: an entity can be considered to participate with the same properties (the same view) or have different properties (different views) during the diagnostic process. ii) Property levels: the properties of the entities can have n different abstraction levels. iii) Number of hypotheses: the goal of the diagnosis is a single validated hypothesis. There can be one or several candidate diagnostic hypotheses which must be evaluated in order to select the valid one. iv) Reasoning types: show the ways in which the rules are applied by the inference motor in order to infer a final diagnosis.

There is variability in user requirements too. We have elicited and modeled as UML-Use Case diagrams [5] the user's interaction requirements because they impact in the software architecture structure. We can describe this variability in terms of its features as follows: i) Number of use cases of the system and how the system interacts with the environment (final users).

ii) Number of actors: number of final users of the system. iii) Use cases per actor: an actor can access different use cases.

However, the features of the application domain need to be considered too. Therefore other variability emerges: the application domain variability. The features that correspond to a specific application domain are: i) Name and type of the entities' properties by abstraction level. ii) Rules by abstraction levels (the rules describe how the entity properties are related inter-levels). iii) Level, name and type of the hypotheses used in the diagnostic process. These features will be used to decorate the base architecture skeletons in order to obtain the final products.

2.2 The Expert Systems: domain functionality

Based on the field study carried out in the Expert System domain, we have detected that a unique generic architecture captures the functionality of these systems. This generic architecture has three basic modules [2]: i) the module that contains the inference process that resolves a problem in a specific domain, ii) the module that contains the knowledge about the domain, and iii) the module that allows communication between the user(s) and the system.

But as we implement the SPL's assets using PRISMA software architectures, the modules of the Expert System generic architecture must be mapped into the following architectural elements: the Inference Process Component which establishes system control and provides the general resolution strategy for taking a decision; the Knowledge Base Component which contains the domain knowledge of the case study using application domain rules (Horn clauses) and facts (information unchanged); and the User Interface Component which establishes the man-machine interaction. Therefore there is a correspondence among the modules and their respective components. However, to be consistent with the PRISMA metamodel, it is necessary to incorporate a new architectural element (the connector), to establish the communication among the components; therefore the other architectural element is the Connector, which contains the choreography of the decision process.

However, we have detected that the base architectural model implementing the generic modular model is not unique; it varies in its structure and behavior. This implies the creation of several base architecture models for each generic model in a product line.

We have developed a modeling technique to design Expert System using PRISMA software architectures [6]. This modeling technique takes into account that an Expert System varies both in the structure and in the architectural elements of its architectural model. In our

modeling technique the end user requirements are modeled using UML use case diagrams.

Furthermore, we have concluded that the variability in the structure of the Expert System is related to the features associated with the user requirements, and the variability in the behavior of the Expert System is related to the associated diagnostic process.

3. BOM: The framework

Following the results of our field study, variability management in BOM includes two types of orthogonal variabilities. The first variability (V1) has as variants the domain features and the end user requirements, which determine a specific base architecture. The second variability (V2) corresponds to the application domain features, which determine the final product of our SPL.

Since BOM follows a modeling oriented approach, we manage the variabilities as follows: The features of the first variability are represented in a Feature Model (see Figure 1). These features are represented as variability points and their variants.

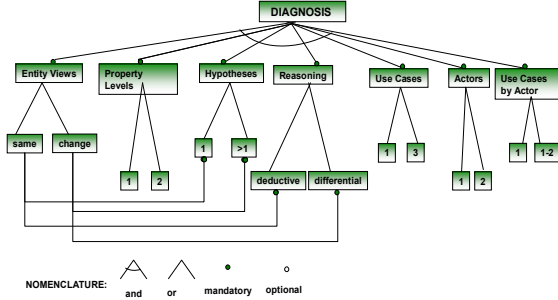


Figure 1. The Diagnostic Feature Model

The Feature Model is translated to an equivalent UML-class diagram with OCL constraints (see Figure 2): the Domain Conceptual Model. The variants are captured as instances of this Conceptual Model. The instances capturing the domain features in the medical diagnosis are: *entity view=same*, *property levels=2*, *hypotheses=14*, *reasoning=differential*, *use cases=3*, *actors=2*, *use cases by actor=2* by actor A & 1 by actor B.

The features of the second variability, i.e. variability of the application domain, are captured as instances of the Application Domain Conceptual Model (see Figure 3). We present an example of these instances in the medical diagnosis case study: *properties of level 0=cough, fever*; *properties of level 1=dry_cough, constant fever*; *hypotheses of level 1=warth, parotiditis*; *hypotheses of level 2=pneumonia, bronchitis*; *rules=IF (cough=true and fever=true and respiratory_difficulty=true) THEN syndrome=warth*.

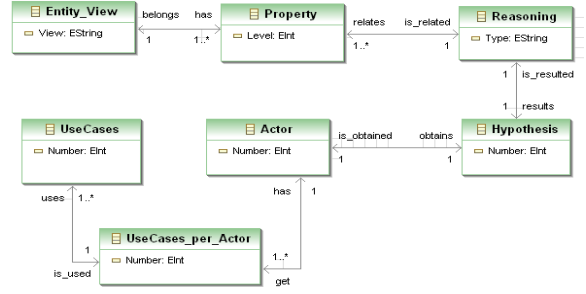


Figure 2. The domain conceptual model

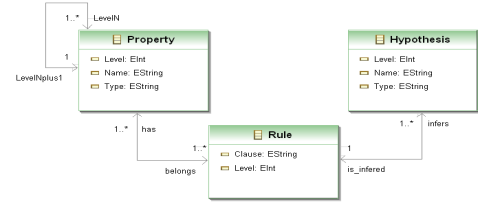


Figure 3. The application domain conceptual model

For this reason, the modeling of the system is separated into two stages:

Stage 1: Constructing the Domain Conceptual Model (capturing in its instances the variants of the domain: V1) and the model of the functionality of the system (i.e. a unique generic architecture associated with the Expert System). This information, by means of Query/Views/Transformations OMG standard [7]: QVT-Relations, produces the corresponding base architectural model (as a template or skeleton of the system in the PRISMA Architectural Description Language). A unique generic architecture is shared by several base architectures (skeletons of base architectures), which represent the SPL1.

Stage 2: Constructing the Application Domain Conceptual Model capturing in its instances the application variants. The Application Domain Conceptual Model's instances together with the functional model of the system (i.e. the base architecture produced in stage 1) are transformed into a PRISMA architectural model, i.e. the final product of the SPL, i.e. the Expert System, which is compliant with the two variabilities and the functionality selected. The second type of variability involves the SPL of the application in a specific field, i.e. SPL2. This variability causes the base architectures to be decorated with the application domain features, in order to generate the types of the PRISMA software artifacts. Therefore, a base architecture can be mapped into one or more PRISMA architectures, because in each base architecture can be inserted different properties of the application domain.

4. Expert Systems views in BOM

In BOM we have considered two kinds of views of the Expert System: the System Variability View and the System Functional View.

The System Variability View is described using the two variability conceptual models: the Domain Conceptual Model which captures the domain and user variabilities (V1) and the Application Domain Conceptual Model capturing the application domain variability (V2). The Domain Conceptual Model conforms to the V1 Metamodel (MM V1) and the Application Domain Conceptual Model conforms to the V2 Metamodel (MM V2). Both metamodels are the UML 2.0 Class Diagram Metamodel [5], but other Domain Specific Metamodels could be used giving rise to Domain Specific Models to capture the System Variability View.

The System Functional View is given, at different stages of the process, by means of two views, which are described using three architectural models.

- the modular view: for the Generic Architecture Model which conforms to the Modular Metamodel (MM MODULAR),
- the component-connector view: for the Base Architecture Models which conform to the Skeleton Metamodel (MM SKELETON), and the PRISMA Architecture Models which conform to the PRISMA Metamodel (MM PRISMA) for the final product. The Skeleton Metamodel is similar to the PRISMA Metamodel [4] but it allows feature holders.

4.1 The system's views metamodels

We have used the Meta Object Facility (MOF) [8] graphic notation for the specification of these metamodels. The Modular and Component-Connector view types proposed in [9] have been chosen as the most appropriate views in our approach.

Figure 4 shows the Modular view metamodel. The main element considered for this view is the module. The relation class represents the styles (*decomposition*, *uses* and *layer*). These styles are treated as relations since what distinguishes one style from another is the type of relation. The labels in the links are useful for indicating how the relation is made.

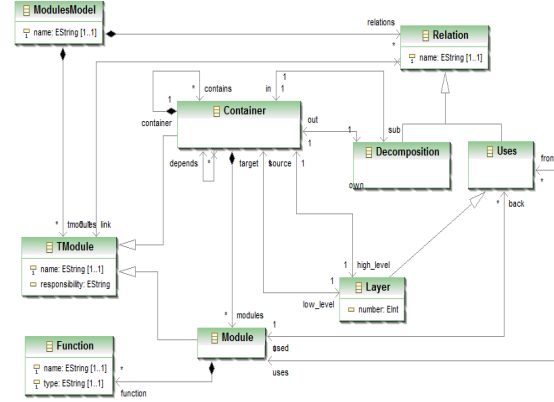


Figure 4. Modular view metamodel

Figure 5 shows the Component-Connector view metamodel. This Figure shows the component class and connector class as the main elements. Both are derived from a more general component class (TComponent). The way in which these elements are related corresponds to their relations (*Pipe-Filter*, *Client-Server*, *PeerToPeer*, *Publish-Suscribe*, *Shared-Data*). These relations are inherited from the Relation class, which links the components by means of the connector.

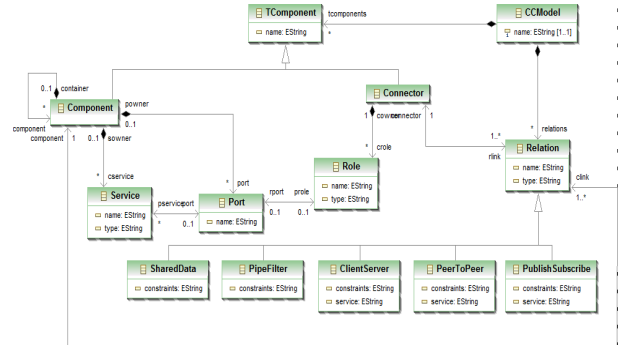


Figure 5. Component-Connector view metamodel

4.2 The relations among metamodels

We used QVT transformations to establish the relations among metamodels in MOF. The QVT-Relations language is used to describe the relations.

The source and target metamodels are identified first. The correspondence among each element of the metamodels must be defined taking into account the way in which their rules are represented through QVT. In this case, the source metamodels are the Modular metamodel and the V1 variability metamodel. The target metamodel is the Skeleton metamodel. This means that the rules considered in the relations for the elements of the modular and V1 variability metamodels, can only be of *check only* type (to verify the elements) and of *enforce* type (to create the elements of the skeleton metamodel).

Two types of relations between the System Functional View have been identified: i) the relations between the Modular view and the Component-Connector view, ii) the relations between the Component-Connector view and the PRISMA view together with the views that define the variability of the SPL. Their respective profiles are:

$R1 \subseteq \text{MM MODULAR} \times \text{MM V1} \rightarrow \text{MM SKELETON};$
def
 $R2 \subseteq \text{MM SKELETON} \times \text{MM V2} \rightarrow \text{MM PRISMA};$
def

4.3 The transformations among models

Figure 6 shows the transformations involved in the construction of our SPL architectures. This figure illustrates how the transformations performed at the model level are applied in the transformation processes. The modular model is made up of three modules: *InferenceMotor*, *KnowledgeBase*, and

UserInterface. These are all linked with each other through dependence relations. Next, the transformation task for producing the skeleton model (target) is performed. To do this, the Domain Conceptual Model is used. A skeleton component is produced for each module, and each dependence relation generates a skeleton connector.

The level M1 is where the transformations T1 and T2 are carried out. A first model (skeleton model) is obtained by the transformation T1, using the modular model and the Domain Conceptual Model as sources. This skeleton model is refined by means of transformation T2 using the Application Domain Conceptual Model which allows the PRISMA model to be obtained as a refinement.

In BOM, the model transformations T1 and T2 are executed by means of our tool using QVT-Relations model transformations (these are Platform Independent Model to Platform Independent Model transformations).

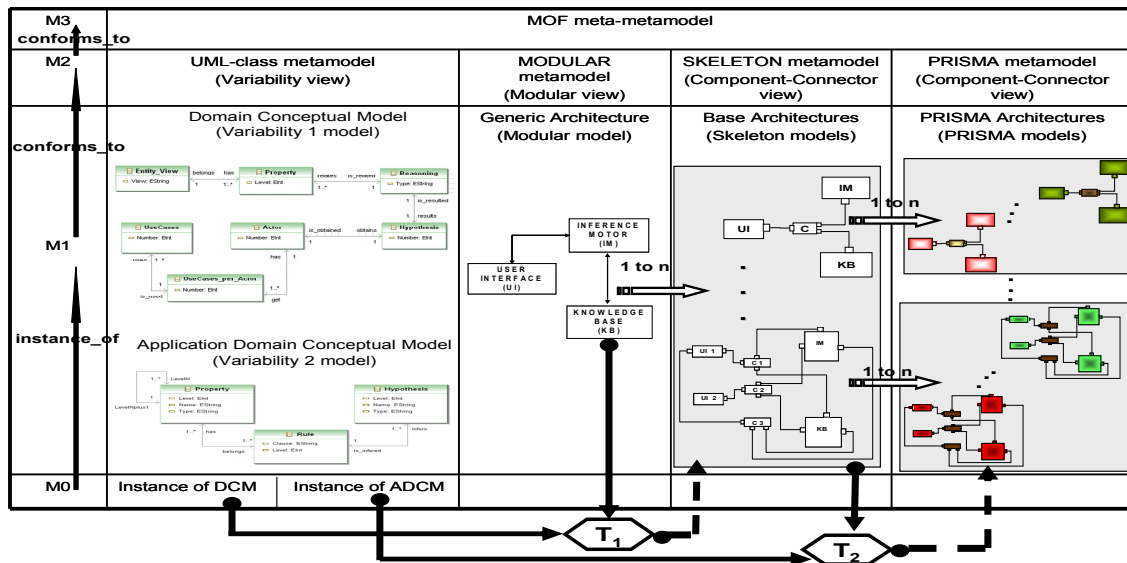


Figure 6. The model transformations T1 and T2 in BOM (expressed in the MOF levels)

The QVT-Relations code (partial) is as follows:

```
transformation modules2components(mdomain:
mview, dcmDomain: dcm, ccdomain: ccview) {
  //... Keys declaration
  top relation ModulesModel2ComponentsModel
  { checkonly domain mdomain modulesModel:
mview::ModulesModel { };
  checkonly domain dcmDomain varModel:
dcm::DomainConceptualModel { };
  enforce domain ccdomain componentsModel:
ccview::CCModel {name=modulesModel.name};
  }
  where {
    UseCase2Connector(modulesModel,varModel,
componentsModel); }
  //... Other relations
}
```

At the end, BOM uses the PRISMA-MODEL-COMPILER tool [10] to automatically generate the executable code (in C#.NET) from the PRISMA Platform Independent Model generated model. BOM executes the object code over the PRISMA-NET Middleware [10] (our Platform Specific Model).

An example of the generated C# code which corresponds to the Knowledge Base component of the educational program diagnosis case study is as follows:

```
namespace KBE
{ [Serializable]
  public class KBase: ComponentBase
  {public class KBase string name:base(name)
  {AddAspect (new FBaseE ( ) );
```

```

InPorts.Add("KnowPort", "IDomainE", "KNOW");
OutPorts.Add("KnowPort", "IDomainE", "KNOW");
} } }

```

In BOM, the final user (application engineer) builds an Expert System (product of SPL) by giving as input the features of the variabilities V1 and V2 using instances of the Domain Conceptual Model and Application Domain Conceptual Model respectively.

5. Related works

SPL is nowadays a hot topic and there are a wide variety of research results offering suggestions and solutions in specific domains. Our research is related to the following: i) Batory et al. [11] capture the domain features in the Feature Model. In BOM we capture the Feature Model's features in conceptual models. ii) Trujillo [12] uses Feature Oriented Programming as a technique for inserting features. In BOM we use this technique but at the model level, i.e. we use Feature Oriented Modeling. The features are inserted, by means of QVT transformations, in the skeleton aspects of the architectural elements that make up the PRISMA models. iii) Bachman et al. [13] propose to separate the variability and the functionality description of the software artifacts. In BOM, variability and functionality are specified in separate conceptual models.

6. Conclusions and Future Work

In this paper we describe how we automatically generate Expert Systems in a specific domain, by means of our BOM framework.

In order to cope with the complexity of the Expert Systems development, we deal with the variability in two stages: in the first stage, the variability is reflected in the base architectures of the SPL sharing a generic architecture; in the second stage the variability is reflected in different application features used to produce the PRISMA architectures (i.e. our SPL) in a specific domain sharing a base architecture. BOM manages the variabilities at the model level.

BOM improves the development of Expert Systems in the following ways: i) By using the characteristics of Expert Systems to separate the inference process from the knowledge domain, and to incorporate several reasoning. ii) By applying SPL techniques. In this way cost, time, effort, and complexity are reduced. iii) By developing product line architectures as PRISMA models, reusing and integrating software components and aspects. iv) By applying MDA techniques in order to implement systems on different platforms, and to automatically obtain executable applications.

The main contributions of BOM are: application development by automatically generating code, and the management of two kinds of variability described in two independent conceptual models separated from the functionality models.

In this paper we have presented the diagnostic Expert Systems domain. In the future we will apply BOM in different domains. Additionally, we will validate BOM performances in real life situations by means of our prototype [14]. We also plan to use benchmarks in order to compare BOM results with other approaches.

References

- [1] MDA: <http://www.omg.org/mda>.
- [2] J. Giarratano, and G. Riley, *Expert Systems: Principles and Programming*. 4th Edition: (Hardcover), 2004.
- [3] P. Clements, and L.M. Northrop, *Software Product Lines: Practices and Patterns*, SEI Series in Software Engineering, Addison Wesley, 2001.
- [4] J. Pérez, *PRISMA: Aspect-Oriented Software Architectures*, PhD. Thesis in Computer Science, Universitat Politècnica de València, Spain, 2006.
- [5] UML: <http://www.omg.org/docs/formal/05-07-04.pdf>.
- [6] M.E. Cabello, *Análisis y diseño de un generador automático de sistemas de diagnóstico basado en LPS*, Tesis de Master, UPV, España, 2007.
- [7] QVT: <http://www.omg.org/docs/ptc/05-11-01.pdf>
- [8] MOF: <http://www.omg.org/mda/specs.htm#MOF>.
- [9] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architecture, Views and Beyond*, Addison-Wesley, 2002.
- [10] J. Pérez, N. Ali, C. Costa, J.A. and I. Ramos, "Executing Aspect-Oriented Component-Based Software Architectures on .NET Technology", *3rd International Conference on .NET Technologies*, pp. 97-108. Czech Republic, 2005. ISBN 80-86943-01-1.
- [11] D. Batory, D. Benavides, and A. Ruiz-Cortés, "Automated Analyses of Feature Models: Challenges Ahead", *ACM on Software Product Lines*, 2006.
- [12] S. Trujillo: *Feature-Oriented Model Driven Product Lines*, PhD. Thesis, The University of the Basque Country, San Sebastian, Spain, 2007.
- [13] F. Bachman, et al. "A metamodel for representing variability in product family development", *the 5th International Workshop on Product Family Engineering*, 2003, pp. 66-80.
- [14] M.E. Cabello, M. Gómez, M. LLavador, and I. Ramos. *ProtoBOM: a framework that semi-automatically generates Decision Support Systems based on SPL*, Technical report: DSIC II/02/08, UPV, Spain, 2008.