# Evolving a Model Driven Software Product Line for Management Information System Applications

**Juan Sebastián Montaña Ortega**

**UNIVERSIDAD DE LOS ANDES**

**ENGINEERING FACULTY**

**SYSTEMS AND COMPUTING ENGINEERING DEPARTMENT**

**SYSTEMS AND COMPUTING MASTER**

**BOGOTÁ D.C., 2009**

# Evolving a Model Driven Software Product Line for Management Information System Applications

**Juan Sebastián Montaña Ortega**

**Master Thesis**

**Advisor**

**PhD. Rubby Casallas Gutiérrez**

**Associated Professor**

**UNIVERSIDAD DE LOS ANDES**

**ENGINEERING FACULTY**

**SYSTEMS AND COMPUTING ENGINEERING DEPARTMENT**

**SYSTEMS AND COMPUTING MASTER**

**BOGOTÁ D.C., 2009**

# Contents

# Figures

# 1. Introduction

High complexity of requirements, faster growing systems, and frequent appearance of new technologies make evolution a constant issue in software industry. To deal with software evolution, several proposals have arisen with the purpose of making maintenance tasks easier while improving team's productivity.

One of the most known approaches used to tackle previous concerns is Software Product Lines (SPL). One of the main reasons to use an SPL approach is to take advantage of commonality in a set of applications by creating reusable components. In SPL Engineering, the first task to perform is to create components, called the assets of the line. After assets are created, they are reused to produce specific products.

On the other hand, Model Driven Software Development (MDD) uses models as first class entities; meaning that they will be used during the whole development process. Stakeholders create models for a particular domain and these models are transformed into more detailed models, deriving the desired product.

Merging both SPL and MDD approaches; we obtain a Model-Driven Software Product Line approach (MD-SPL). An MD-SPL approach aims to ease the software development process, by reusing its assets (metamodels and transformations) to obtain specific products. However, software systems need to evolve and software built using MD-SPL approaches is not the exception [1].

According to [2], SPL approaches deliver software systems organized around commonalities shared by a family of products. These commonalities are supported by the assets of the SPL. Other functionalities of a system are included for a specific software product. However, if new functionalities for a product can be also used in other systems, they can be generalized and included as new assets of the SPL, forcing the assets to evolve.

The purpose of this work is to present an approach to deal with evolution of an MD-SPL, in order to reduce resources (costs, time and developers) and increase benefits, such as productivity of a development team.

Our proposal is validated using a case study for our MD-SPL [3], which faces evolution needs due to new requirements that we have identified. We have decided to include new concepts in the metamodel level to deal with this problem. We present an approach to evolve an MD-SPL, taking into account the impact of modifying it. By evolving the line, we increase the productivity in development teams, generating a considerable amount of code.

The document is structured as follows: Chapter 2 presents context for the problem and describes our MD-SPL and the scope of it (the products we are currently able to build). Chapter 3 presents the problem we are dealing with and clarifies it using an example of our MD-SPL. Chapter 4 presents our approach to evolve the MD-SPL and how other approaches attack the evolution problem. Chapter 5 shows the implementation of the solution in the MD-SPL. Chapter 6 presents the impact of dealing with changes. Finally, chapter 7 presents future work and conclusions.

# 2. Context

## 2.1 Model Driven Software Development

Model Driven Software Development (MDD) claims to improve software construction using models as first class entities; they are not used only for documentation and communication purposes, they are used during the whole development process [4]. Stakeholders create models for a particular domain; these models are transformed into lower level models, deriving the desired product.

Model Driven Architecture (MDA) [5] is a proposal by the Object Management Group (OMG) that encourages the creation of domain models, independent of technological details. These models are transformed into new models related to some specific platform implementation.

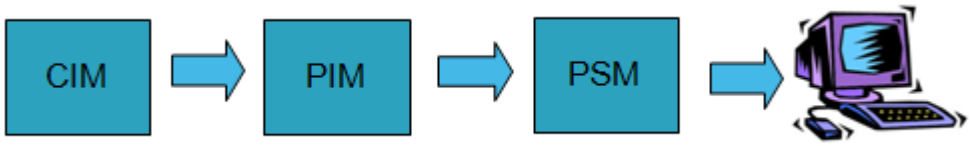


**Figure 1. MDD Process**

Figure 1 shows the MDD process. A Computation Independent Model (CIM) is the main input in an MDD approach; this model describes a particular business context and requirements. The CIM is refined to a Platform Independent Model (PIM), which specifies services that will be provided to the business. Finally, the PIM is refined to a Platform Specific Model (PSM), which includes a concrete realization of services, tied to a particular technology or platform model. The code is obtained by applying model-to-code transformations to the PSM.

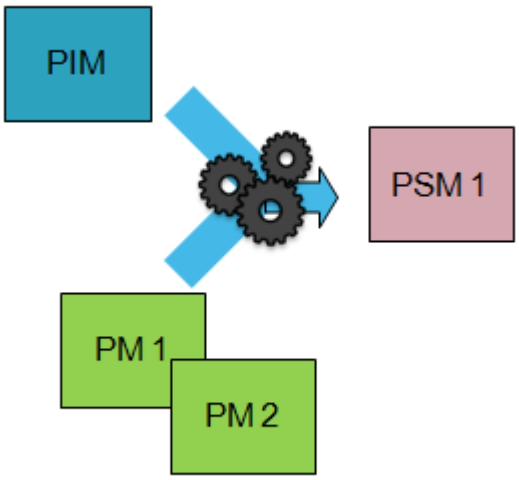


**Figure 2. Platform Models in MDA**

The main goals of MDA are portability, interoperability and reusability. A key principle to achieve these goals is to separate the specification of a system from its implementation. MDA proposes to specify a system without the platform that will support it (PIM) and to specify different platforms (PM); then, a specific platform will be chosen to execute a system and the system specification will be transformed into a model for a particular technology (PSM).

## 2.2 Software Product Lines

A Software Product Line (SPL) approach, aims to improve productivity in the software development process. An SPL is defined as “a set of software systems that have a similar purpose and share common features” [6]. Products of an SPL are created using reusable components, which are called the assets of the line. These assets may be models, documents, fragments of code or any other useful element to generate a product.

The typical development process in an SPL is composed by two phases:

**Domain Engineering**

In this stage of the process, the assets of the line are built after analyzing and deciding the type of products that will be obtained. It is important to define the scope of an SPL in order to create reusable artifacts. These assets are produced taking advantage of the commonality of the systems that will be obtained later. To be able to generate different products, variability assets can be defined in an SPL.

### Application Engineering

This phase is related to creating specific products by reusing previously defined assets [6], [7]. In order to produce different applications, an SPL should provide variability mechanisms. Creating a product includes requirements specification, assets configuration (related to variability in the SPL), reuse of assets and finally code generation.

## 2.3 Model Driven Software Product Lines (MD-SPL)

Several approaches, like to proposed in [8], [9] and [10], are SPLs based on MDD. These proposals are called MD-SPL approaches. An MD-SPL is a set of products developed departing from domain application models which conform to domain application metamodels, and that are derived using a set of model transformations. These model transformations may require various stages. At each stage, domain application models are automatically transformed to include more details. Any MD-SPL approach aims to ease the software development process, by reusing its assets (metamodels and transformations) to obtain specific products. However, software systems need to evolve and software built using the MD-SPL approach is not the exception [1].

We have developed an MD-SPL [3] whose main objective is to provide an agile and semiautomatic development process for MIS Applications using MDA concepts. Next subsection revises MIS Applications and limits the scope of the systems we aim to generate; and the last section of the chapter explains the way we manage the creation of these applications in our approach.

## 2.4 Management Information Systems (MIS)

Enterprise applications appeared to support business processes and to face enterprise challenges: requirements in applications like data volume, scalability, distribution and concurrence to handle thousands of users in several places around the world.

Management Information Systems (MIS) are enterprise applications that generate reports and consolidate information from different sources to support the decision making process in enterprises [11]. Typically, an MIS application assists in registering and processing of information, and presenting it in a consolidated way. In this sense, we could say that MIS applications basically provide Create, Retrieve, Update and Delete (CRUD) operations for business entities in any domain.

Our focus is on multi-layer MIS applications, which delegate on application containers the non functional requirements that are typical in these systems. Three-tier architecture is one of the most common architectures used when designing MIS applications.

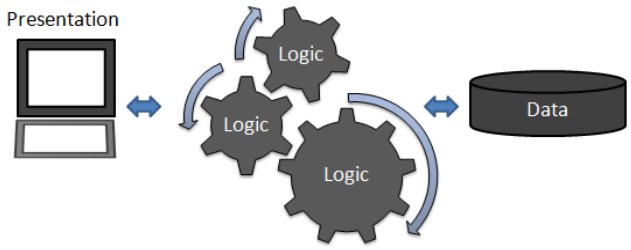


**Figure 3. Three-tier Architecture**

As Figure 3 shows, the Data tier is at the bottom of this architecture; information is stored and retrieved to the upper tier for processing. The Logic tier performs analysis of data to support business processes. Finally, the Presentation tier displays information through graphical user interfaces.

## 2.5 MD-SPL for MIS Applications

We have built an MD-SPL [3] whose main objective is to support the creation of MIS applications through an agile and semiautomatic process.

The assets of our MD-SPL are metamodels, model-to-model and model-to-code transformations. Our framework uses the Eclipse Modeling Framework (EMF) [12], ATLAS Transformation Language (ATL) [13] for model-to-model transformations and Acceleo [14] for model-to-code transformations.

Figure 4 shows the structure of our MD-SPL and the development process used to create an MIS application with our line. We have separated concerns in four different domains: Enterprise Application, Architecture, Platform and Language. Currently, our MD-SPL supports JEE5 at platform level and Java at language level.

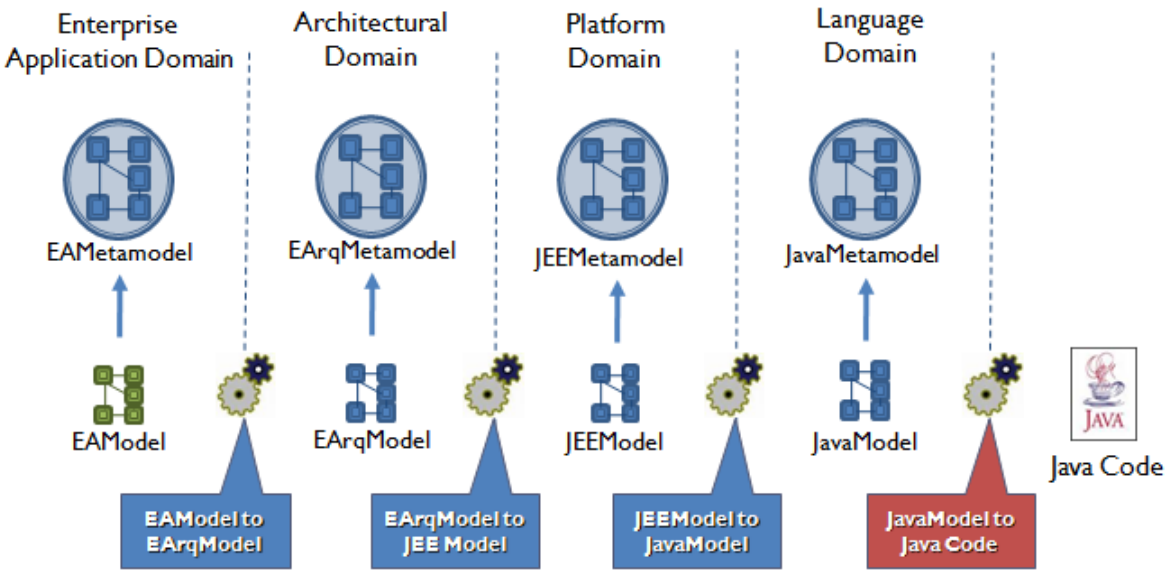


**Figure 4. MDSPL: Structure and Process**

Using our MD-SPL, the development cycle begins by specifying the business domain in a model. In this task, business analysts create the first model, which conforms to Enterprise Application metamodel. In this model, they express the requirements of the application in terms of business entities, relationships between them and CRUD operations.

The second stage of the process is the transformation of the business model into an architecture model; this transformation refines the model to include concepts like application tiers and services. In the third step, the line produces a platform model (JEE5 in our case), which includes concepts such as entity beans, session beans, interfaces and business objects. Then, we produce the last model: a Java model, which conforms to a Java metamodel, and includes concepts such as package, class, method, and attribute. All of these transformations, written in ATL, are executed automatically by the framework. We execute model-to-code transformations, written in Acceleo, to obtain Java code based on the Java model.

Currently, the code we generate using the line is limited to the data and logic tiers of an application; presentation layer was out of the scope of first versions of the MD-SPL. We have to develop by hand all the presentation pages that will be used in an application.

After generating a specific product using the MD-SPL, the development team must write additional code that could not be generated automatically, in order to satisfy all the requirements. This code should be written in specific places (Acceleo's

placeholders); otherwise, it will be lost when regenerating the code using the MD-SPL. Due to the nature of MIS applications, the code that must be written by hand is typically related to more complex versions of the basic CRUD services such as creating some business entities when another is created, updating a business entity and all its relationships or adding mechanisms that allows filtering information to facilitate some decision making process.

# 3. Problem

Evolution is a natural occurrence in software development and an inevitable part of the software lifecycle [15]. Evolution brings changes that are included to correct, to improve, or to extend the assets. Due to dependency between assets, the impact of modifying them may be very high and that is why a successful process must manage these changes effectively [16].

When we are working on the MD-SPL domain, the products of the line also have to evolve. However, and due to the fact that the products are generated from the assets of the line, the problem relies in the evolution of the line (this means, the evolution of the assets). Evolving an MD-SPL implies the modification of the assets to correct it, to improve it or to extend its scope.

To clarify the problem, we will use an example application that we are building in our development group. This application supports pedagogical processes in a business administration course called “Management Simulation”. The purpose of this application is to serve as a tool to execute a market simulation for an industry; based on decisions that corporative groups (groups of students) make. Students and teachers will use the application concurrently, so remote access, performance and availability are of high relevance and must be guaranteed.

This application is a good candidate to be developed using our MD-SPL. We need to model several business entities and relationships between them. Our business model has over 300 business entities and several associations (one-to-one, one-to-many, many-to-many) that relate them. A fragment of this model is shown in Figure 5.

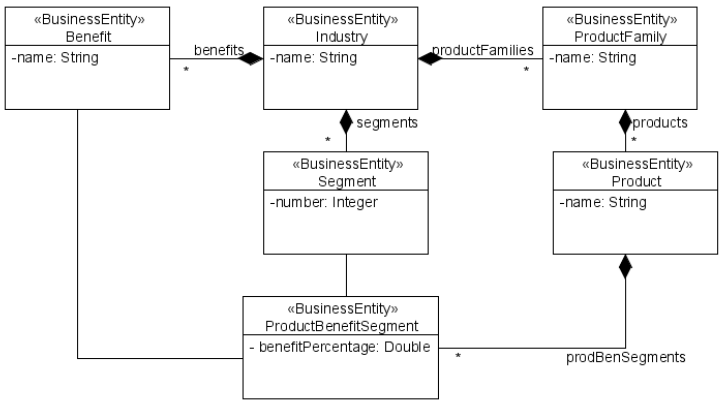


**Figure 5. Fragment of Business Model**

In a business model, we use concepts like *Business Entity, Attribute, Simple* and *Multiple* to model structure and relationships. This model exemplifies one of the complex types of services that the application has to provide; when creating a *Product* for a *ProductFamily*, we have to create also a *ProductBenefitSegment* for each *Benefit* and *Segment* that exists in the *Industry* related to that *ProductFamily*.

The current scope of our MD-SPL only supports unidirectional relationships and CRUD operations for a single business entity. Based on this, accessing one entity from another is not always possible and due to the complexity of business logic, it is easy to make mistakes when hand-coding operations related to business entities and their relationships.

Another requirement that is very common in MIS applications due to their nature is to consolidate the information to support the decision making process. Specifically, we are talking about filtering the information when creating a business entity (for example to filter ProductFamily by Industry when creating a Product) or filtering the information when visualizing it.

Finally, the presentation layer components must be created by hand to allow users interact with the application. Creating these components, given the complexity of relationships, becomes a time consuming task.

With previous example, we can clearly see the evolution needs that an MD-SPL may have to deal with. Including support to new requirements or increasing the scope of the line are very common issues when working in the MD-SPLs domain. However, modifying the assets of a line may have a huge impact because in several cases the dependency among assets is high, or one change may imply to modify a lot of assets. We propose a mechanism to deal with evolution of an MD-SPL by reducing resources (costs, time and developers) and increasing benefits, such as productivity of a development team.

# 4. Strategy

Having a layered architecture reduces the complexity of software design, by clearly defining responsibilities and concerns of each one of the tiers. One layer needs to interact with another, so for integrating new requirements, all layers must evolve together.

To be able to include a change in the MD-SPL, there are two possible strategies: modifying assets of the line (metamodels and transformations) or creating them. In the former case, we have to perform modifications throughout all of the assets of the line: if we include concepts at the business domain, we also need to include concepts in the other domains to keep trace of the new information and be able to generate code for the final model. In the latter case, we will enrich the line by including new assets, being able to express information related to new concerns. Adding these assets implies integrating them with previous ones.

Although we want to evolve an MD-SPL by adding new requirements, not all of them imply the same type of changes in the assets. We propose a categorization for changes, whit the corresponding evolution process for each one of them. At the end of this section, we compare our work with other approaches that propose different alternatives to deal with the evolution problem.

## 4.1 Adding concepts to a metamodel where domain models are created by hand.

We consider that when a change implies the addition of concepts into a metamodel, and models conform to this metamodel are created by hand; this change fits in our first category. The process we propose to follow when dealing with evolution for this type of changes is shown in Figure 6.

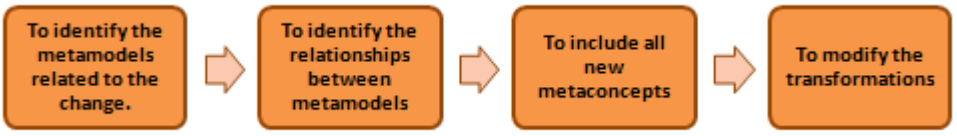


**Figure 6. Evolving an MD-SPL when domain models are created by hand**

The first step of the process is to define the metamodel where changes have to be included. It is important to identify the place where a specific concern is represented in the line to include it successfully. After the metamodel is identified, relationships with other metamodels should be identified to know which concepts will be added into each one of them. When all metamodels impacted by a change

are identified, the new concepts will be included. Finally, model-to-model and model-to-code transformations have to be modified to include new metaconcepts.

## 4.2 Adding concepts to a metamodel where domain models are generated automatically and user interaction is needed.

A change belongs to our second category when it implies the addition of concepts into a metamodel, models conform to this metamodel are automatically generated and human interaction is required to continue with the MD-SPL process. Figure 7 shows the process we propose to deal with this type of changes.

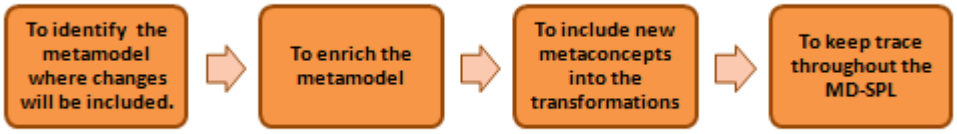


**Figure 7. Evolving an MD-SPL when domain models are generated**

We begin the evolution process by identifying the domain where the new concern will be included. Given that human interaction is needed to complete the inclusion of the requirement, the metamodel should allow expressing the concern in terms of already existing concepts, or in new concepts that enrich this domain. After the concepts are added, they should be included into the transformations. Notice that the changes in transformations are different from those proposed on previous subsection. In this case, we must create transformation rules, different to the other case where existing transformations may be impacted by the change. Adding a new concept into one metamodel implies to keep trace of the change throughout the whole line and to allow the code generation related to this change, so the change must be propagated until the end of the line. This can be done by including concepts into the model-to-model transformations.

## 4.3 Adding concepts to a metamodel when human interaction is not required.

When a change does not imply user interaction, it belongs to our third category. The first step of the evolution process for these changes is to identify once again the metamodel where changes will be included and then to include them. The model-to-model transformations have to be extended (by creating new transformations rules) or modified (by adding concepts to existing transformations). Code will be generated for these concepts, so they should be propagated until the final model of the line. This process is different from the one we showed before because we do not have to provide high abstraction levels for somebody; all the new concepts are included into a model using information from other models.

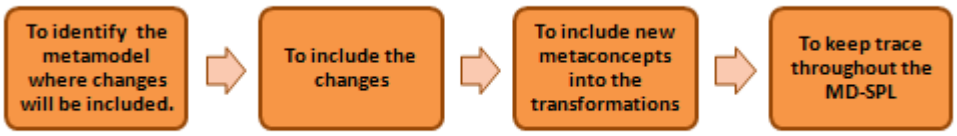


**Figure 8. Evolving an MD-SPL when human interaction is not required**

## 4.4 Evolution problem tackled using other approaches.

Evolution is a well-known problem, so several approaches have appeared to deal with it. However, each one uses a different perspective and proposes different mechanisms to solve the problem. In this subsection we show some existing approaches to deal with evolution.

One of the proposals, explained in [17], tackles the evolution of Software Architectures using MDE. This approximation is based on defining conformance models between different abstraction levels, to normalize any two metamodels. This normalization process must be done to guarantee the evolution of all the assets. After normalization, a source and target metamodels should be defined, and the evolution of assets is achieved by executing a source-to-target transformation. When the evolution process finishes, they document it using UML and ADL's. Moreover, ADL-to-UML transformations are defined, in order to obtain all the documentation in UML.

Mens et. al. developed an UML metamodel for evolution purposes. This work [18] is based on the concept *evolution contract*. An evolution contract captures (in a formal manner) the evolution behavior, by keeping track of all the modifications an asset has suffered. An advantage of this proposal is that tackles evolution problem in a generic way, being potentially useful for dealing with unanticipated evolution.

Compositional metamodeling is the approach proposed by Karsai et. al. in [19]. This aims to simplify the evolution process, by reusing existing metamodels and composing them. The main motivation of this technique is to make metamodels more scalable and easier to evolve.

Although we are interested in changes made to the assets of the line (changes on metamodels and transformations); there are different proposals which deal with the evolution of models so they keep conformance to evolving versions of a metamodel, like those presented in [20] and [21]. Evolving these models is out of the scope of our work, we only focus on changes made on assets.

# 5. Implementing the Solution

In this section, we show how we applied the strategy described on Section 4 into our MD-SPL.

## 5.1 Adding concepts to a metamodel where domain models are created by hand.

We have noticed the need of adding a mechanism in our MD-SPL to express relationships that are typical in MIS applications and that we were not considering before. Due to the complexity of business logic, it is easy to make mistakes when hand-coding operations related to business entities and their relationships. Based on this issue, it is desirable that our line provides some way to model additional characteristics of relationships and takes them into account when generating CRUD operations.

We include a new concept, the “Multirelation” concept, in our business and architecture metamodels, based on the extended entity-relationship model mapped to object oriented programming [22]. In the platform and language metamodels, it was enough with adding attributes to existing classes. To facilitate the management of relationships, we included bi-directionality in the line; in this way it is easier to access one business entity from another. To accomplish the complete task, we had to modify all our model-to-model transformations, now taking into account the new concept (Multirelation) through the whole MD-SPL. The last two transformations were very easy to modify, because we do not have to include new transformation rules; we only included new attributes in the existing rules. We also had to modify the model-to-code transformations related to CRUD operations.

The fragment of business model we showed in Figure 5 appears in Figure 9 after changes in the MD-SPL were done. We are now providing new capabilities to model a *Multirelation*, specifying how to obtain each *Destiny* using an OCL like query.

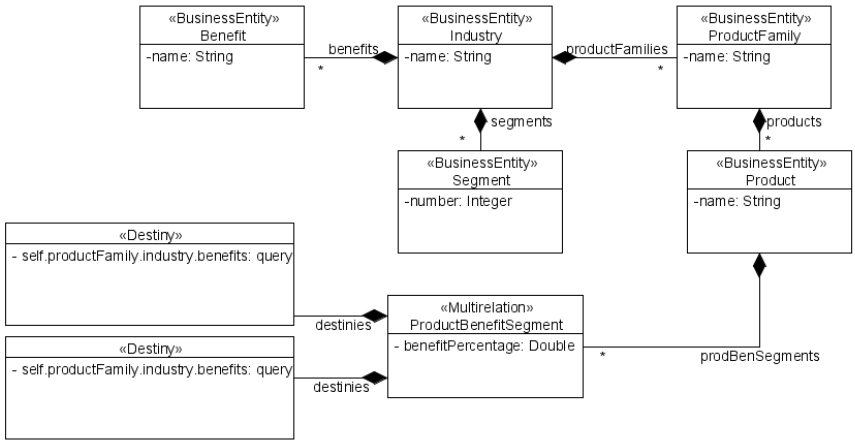


**Figure 9. Fragment of Business Model after MD-SPL Evolution**

## 5.2 Adding concepts to a metamodel where domain models are generated automatically and user interaction is needed.

We found that dealing with additional filters for information when creating the business model was a difficult task due to the lack of mechanisms to express them and we do not want to create a language to filter information. It was also a complex task to try to keep trace of a filter throughout the line, so we noticed filters were closely related to implementation details.

We opted to create a new metamodel at the same abstraction level as the platform metamodel. An example of a filter model is shown in Figure 10. This new model has to be provided as an input to the line. The fact that filters are close to code suggests that they should be expressed in a domain which is close to code. That is the reason why we decided to include the new input (filter model) in the platform domain. In this way, we propose to compose the platform model (generated from architecture model) and the filter model (the new input of the line) obtaining a complete platform model, with all the information from both inputs. We created a transformation that merges filter and platform models producing the complete platform model; we changed platform-to-language transformation and completed some model-to-code transformations to generate the filters. It is important to notice that in case of including the filters in the first metamodel, the impact would be reflected in all the assets of the line and a high level language would have to be defined to express the filters; or at least, an existing language should be maintained throughout all the assets of the MD-SPL.

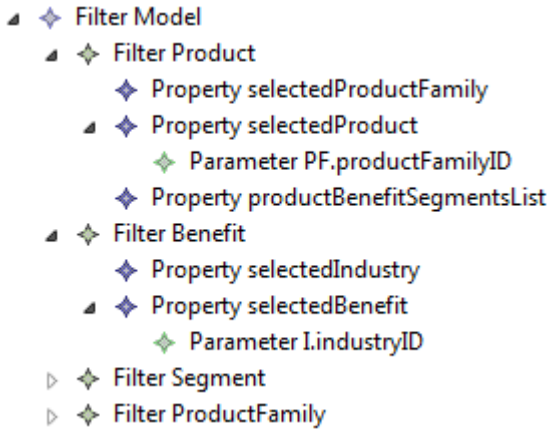


**Figure 10. Sample Filter Model**

The model in Figure 10 is composed by four different filters. The matching point to merge this model with the generated platform model is the name of the filter. Each filter has a set of properties to express the name of the property, if it is a filterable field or not, the EJBQL query to obtain the filter results and if it will be used when creating or updating a business entity.

## 5.3 Adding concepts to a metamodel when human interaction is not required.

Developing the presentation layer code for those products that the MD-SPL generates is an expensive and time consuming task. In case of changing the business model, each of these interfaces has to be updated by hand, increasing development costs. Because of this, we decided to enrich the MD-SPL, to provide facilities to generate code for the presentation layer too. To be able to generate code for presentation tier, we need to include new concepts at metamodel level and to provide a mechanism to communicate the presentation and logic layers.

In our development team, we have a lot of experience with JSF (Java Server Faces) technology, so this was the selected platform for generating presentation code. Taking into account that there are several and well studied proposals related to Model Driven Web Engineering (MDWE) like OOHDM [23], UWE [24], WebML [25], OOWS [26] or OO-H [27], we decided to add this functionality at the platform level (the place where we introduced filters) given that it is our first time to deal with presentation components. Enriching the Web domain by modeling presentation since the first metamodel is out of the scope of our work. We added new concepts to the platform metamodel, such as backing bean, fields and JSP files. We introduce these concepts in the platform model when merging the models, because with the information in each model we can create the presentation related concepts. Currently, we are able to generate the code of a three-tier application to create, edit and delete a business entity and all its multirelations.

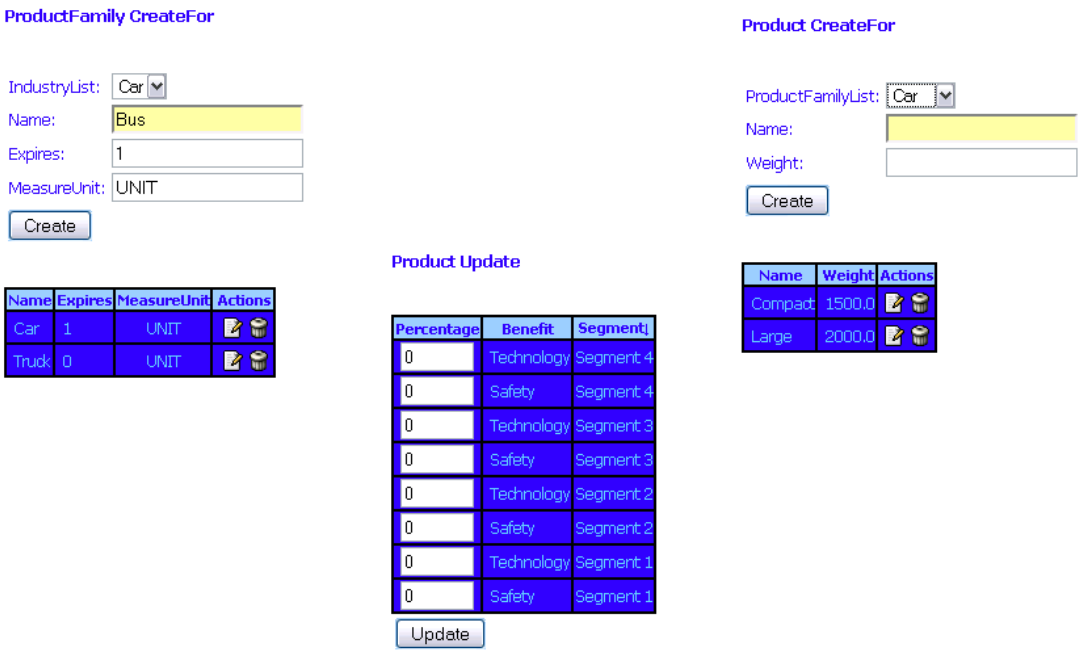


**Figure 11. Some screenshots of a sample application**

Figure 11 shows some screenshots of the "Management Simulation" application that we are able to generate by using our MD-SPL. After all the changes we made to the line, the new structure is shown in Figure 12.

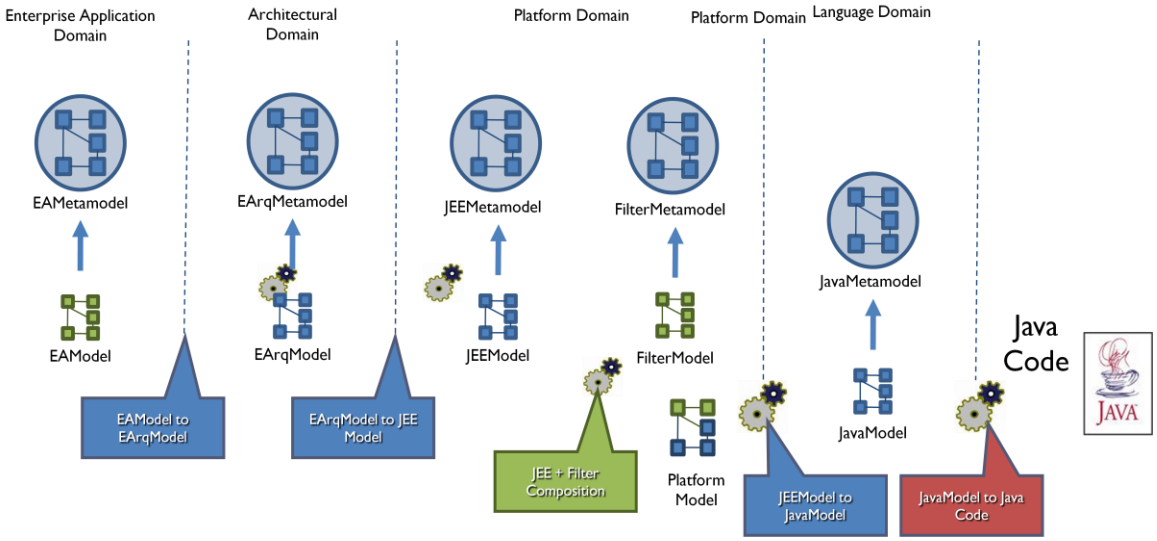


**Figure 12. MD-SPL Structure after Changes**

We keep having a business model as the main input to the line. This model is transformed into architecture model and this one into platform model. At this step of the process, we include a new input (the filter model). The modeler defines the filters that will be used in an application, expressing them in a filter model. We merge the platform and filter models, composing them and obtaining a complete platform model. When merging these models, we also introduce concepts related to presentation layer, using information that we find in both inputs. From the complete platform model, we obtain a language model and at the end we generate the code of the application.

# 6. Impact of the Solution

We measure the cost of performing the changes in the line in terms of modified classes in each of our metamodels. The report shown in Figure 13 was obtained by applying transformations to the two versions of all the metamodels from the MD-SPL based on the work proposed at [28].

A metamodels is also a model, so we execute transformations to transform these models into models conform to a Metrics metamodel. We define some metrics, like the number of attributes of a class, to identify the amount of changes that were done while evolving the line. Overall, we added twenty three attributes for existing classes in metamodels and created twenty two new classes with forty four attributes. Due to a clear definition and separation of concerns in metamodels, it was easy to modify the transformations.

We also wanted to measure the benefit obtained from evolving the line, in terms of the amount of generated code for an application. Modeling two different applications, the percentage of new generated code is shown in Figure 14.

| Enterprise Application | Version 1 | Version 2 |
|---|---|---|
| Number of classes | 13 | 16 |
| Number of associations | 9 | 21 |
| | Metaconcept | #Attrs |
| Modified Metaconcepts | BusinessEntity | 1 |
| | Association | 2 |
| | Simple | 3 |
| New Metaconcepts | MultiRelation | 7 |
| | Destiny | 2 |
| | CreateFor | 1 |

| Architecture | Version 1 | Version 2 |
|---|---|---|
| Number of classes | 22 | 15 |
| Number of associations | 26 | 11 |
| | Metaconcept | #Attrs |
| Modified Metaconcepts | Service | 1 |
| | Attribute | 3 |
| New Metaconcepts | MultiRelation | 3 |
| | CreateFor | 3 |

| JEE5 | Version 1 | Version 2 |
|---|---|---|
| Number of classes | 25 | 24 |
| Number of associations | 22 | 17 |
| | Metaconcept | #Attrs |
| Modified Metaconcepts | Method | 1 |
| | Parameter | 1 |
| | Attribute | 4 |
| New Metaconcepts | BackingBean | 0 |
| | EntityListener | 2 |
| | CreateFor | 0 |
| | Filter | 2 |
| | QueryMethod | 1 |
| | QueryParameter | 3 |
| | ComplexAttribute | 2 |
| | JSPPackage | 2 |
| | JSPFile | 2 |

| JAVA | Version 1 | Version 2 |
|---|---|---|
| Number of classes | 21 | 29 |
| Number of associations | 16 | 32 |
| | Metaconcept | #Attrs |
| Modified Metaconcepts | Field | 4 |
| | Method | 1 |
| | SessionBean | 1 |
| | EntityBean | 1 |
| New Metaconcepts | EntityListener | 0 |
| | BackingBean | 0 |
| | CreateFor | 0 |
| | ComplexAttribute | 3 |
| | QueryMethod | 1 |
| | QueryParameter | 3 |
| | JSPPackage | 3 |
| | JSPFile | 4 |

**Figure 13. Number of changes in the metamodels of our MD-SPL**

With the previous version of the MD-SPL, we had to complete the generated code to manage all relationships of an entity and create the presentation pages for any application. It took us between ten and twenty hours of work, depending on complexity of relationships, to complete the code associated to a business entity. Taking into account these numbers, we can see the advantages of evolving an MD-SPL. Although we do not achieve 100% code generation, we are able to produce functional applications that include all the code related to CRUD operations for a set of business entities and their relationships.

| Application | Generated LOCS before evolution | Generated LOCS after evolution | % of growth in generated code |
|---|---|---|---|
| University Manager | 1476 | 3352 | 227.10% |
| Management Simulation | 13220 | 39621 | 299.70% |

**Figure 14. Increase in the amount of generated code after evolution**

# 7. Conclusions and Future Work

From previous results, we can see the benefits of evolving an MD-SPL. We are increasing the productivity of a development team, generating a bigger amount of code automatically. We enrich the MD-SPL by offering users new mechanisms to model a MIS application in a more complete and detailed way. These new capabilities allow obtaining a bigger amount of code for an application. In fact, we are now able to generate functional three-tier applications. The amount of code programmers have to write has been reduced significantly as previous numbers showed.

We provided mechanisms to deal with different abstraction levels in our proposal: a business model, the main input, and the filter model, which is closer to implementation details. Based on this, we are able to generate more code of an application.

The impact of evolving an MD-SPL depends on the evolution needs it has. We have shown different alternatives to evolve a line, based on the nature of requirements: we added concepts in already existing assets and we also enrich the domains of the line; by providing new assets and extending the scope of the line. By combining these alternatives, we obtained a new version of the MD-SPL that is able to generate more complete products by taking advantage of new capabilities.

Keeping concerns well defined in their correspondent domains facilitates the process of evolving a line, because changes are easy to localize and impact in transformations is reduced if we have a clear separation of responsibilities.

As the line evolved, we noticed it would be useful to have traceability mechanisms between several domains in the MD-SPL, in this way it is easier to know the origin of each element in a specific domain. Having the traceability links simplifies the process of evolving the line, because it is easier to keep track of the results of executing any model-to-model and model-to-code transformation.

It would be desirable to define a strategy for semiautomatic evolution of the models while evolving the line, in this way the models will always conform to the latest version of a metamodel. Defining more complex metrics, not only for the metamodels but also for the transformations, would help in measuring the impact of changing the assets of the line.

# 8. Bibliography


[1] A Van Deursen, E Visser, and J Warmer, "Model-Driven Software Evolution: A Research Agenda," in *CSMR Workshop on Model-Driven Software Evolution*, Amsterdam, The Netherlands, 2007, pp. 41-49.

[2] M. Pussinem, "A survey on Software Product-Line Evolution," *Report 32, Institute of Software Systems, Tampere University of Technology.*, 2002.

[3] A Yie, J Bohórquez, and R Casallas, "Un Caso Práctico en MDA para Construir Aplicaciones JEE5 y.NET," in *VI Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento-JIISIC'07*, Lima, Perú, 2007.

[4] J Greenfield, K Short, S Cook, and S Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools*. USA: Wiley, 2004.

[5] J Mukerji and J Miller. (2003) MDA Guide Version 1.0, OMG Document: omg/2003-05-01. [Online]. http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf

[6] J Bosch, *Design and Use of Software Architectures: Adapting and Evolving a Product-Line Approach*. Boston, USA: Addison-Wesley Professional, 2000.

[7] K Pohl, G Böckle, and F Van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer, 2005.

[8] A L Santos, K Koskimies, and A Lopes, "A Model-Driven Approach to Variability Management in Product-Line Engineering," in *Nordic Journal of Computing*, vol. 13(3), 2006, p. 196–213.

[9] M Voelte and I Groher, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development," in *In Proceeding of the 11th SPLC*, 2007, pp. 233-242.

[10] P Tessier, S Gerard, F Terrier, and J M Geib, "Using variation Propagation for Model-Driven Management of a System Family," in *LNCS 3714*, 2005, p. 222–233.

[11] D Duffy, *Domain Architectures – Models and Architectures for UML Applications*. USA: Wiley, 2004.

[12] (2009) Eclipse Modeling Framework Project (EMF). [Online]. http://www.eclipse.org/modeling/emf/

[13] (2009) ATLAS Transformation Language (ATL). [Online]. http://www.eclipse.org/m2m/atl/

[14] (2009) Acceleo, MDA Generator. [Online]. http://www.acceleo.org

[15] N Chapin, J Hale, K Kham, J Ramil, and W Tan, "Types of Software Evolution and Software Maintenance," in *Journal of Software Maintenance: Research and Practice*, 2001, pp. 3-30.

[16] J McGregor, "The Evolution of Product-line Assets," in *Technical Report, CMU/SEI-2003-TR-005m*, 2003.

[17] B Graaf, "Model-driven evolution of software architectures," in *Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR 2007)*, 2007, p. 357–360.

[18] T Mens and T D'Hondt, "Automating Support for Software Evolution in UML," in *Automated Software Engineering. Volume 7, Number 1.*: Springer, 2000, pp. 39-59.

[19] G Karsai, M Maroti, A Lédeczi, J Gray, and J Sztipanovits, "Composition and Cloning in Modeling and Metamodeling," in *IEEE Transactions on Control System Technology (special issue on Computer Automated Multi-Paradigm Modeling)*, 2004, pp. 263-278.

[20] B Gruschko, D Kolovos, and R Paige, "Towards synchronizing models with evolving metamodels," in *Proceedings of the International Workshop on Model-Driven Software Evolution*, Amsterdam, Netherlands, 2007.

[21] K Garcés, F Jouault, P Cointe, and J Bézivin, "Adaptation of Models to Evolving Metamodels.," , 2008.

[22] R Cattell, D Barry, and M Berler, *The Object Data Standard ODMG 3.0*.: Morgan Kaufmann Publishers, 2000.

[23] (2009) Object-Oriented Hypermedia Design Method. [Online]. http://www.tecweb.inf.puc-rio.br/oohdm/space/start

[24] A. Kraus, A Knapp, and N Koch, "Model-Driven Generation of Web Applications in UWE," in *Proceedings of the 3rd International Workshop on Model-Driven web Engineering (MDWE 2007)*, 2007.

[25] (2009) The Web Modeling Language. [Online]. http://www.webml.org/

[26] J Fons, V Pelechano, M Albert, and O Pastor, "Development of Web Applications from Web Enhanced Conceptual Schemas," in *Conference on Conceptual Modeling (ER)*, 2003.

[27] (2009) The Object-Oriented Hypermedia Project. [Online]. http://gplsi.dlsi.ua.es/iwad/ooh_project/

[28] (2009) ATL Transformations. [Online]. http://www.eclipse.org/m2m/atl/atlTransformations/KM32Metrics/ExampleKM32Metrics[v00.01].pdf

# Annex A. Metamodels

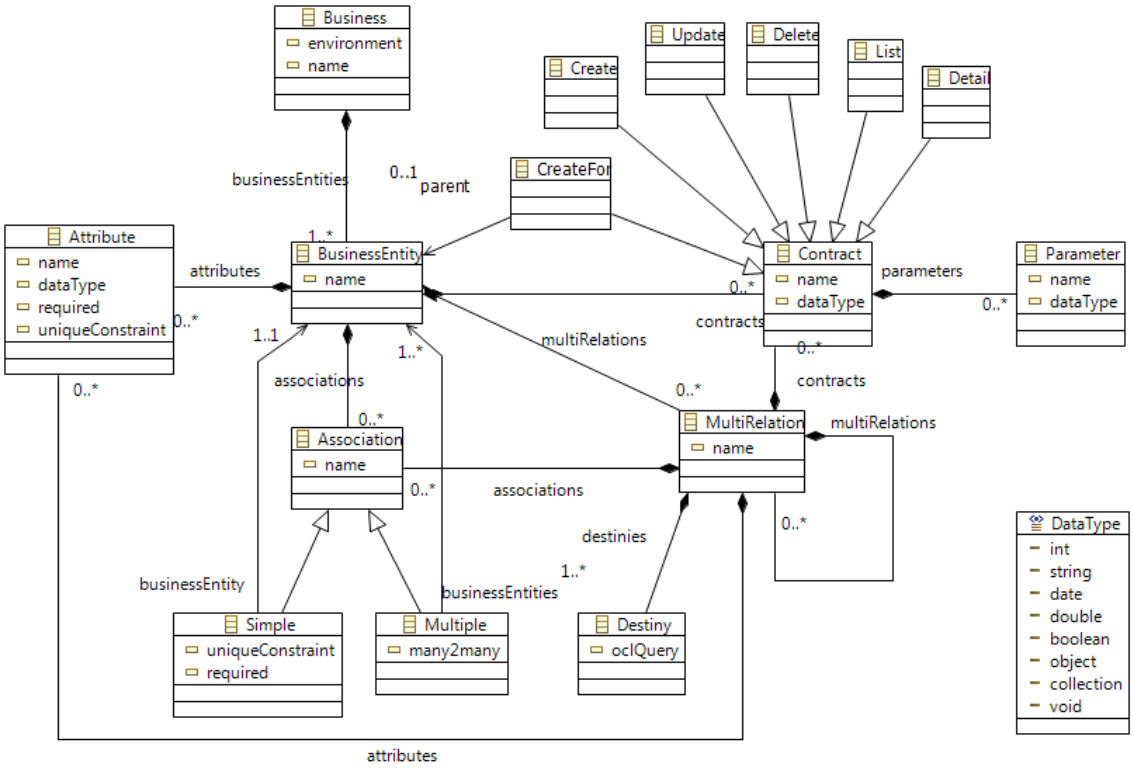


**Figure 15. Enterprise Application Metamodel**

The Enterprise Application Metamodel has the concepts that the MD-SPL supports to specify a business model. A *Business* concept is the root of the application, and it is composed by business entities. A *BusinessEntity* has a name, a set of *Attributes*, a set of *Associations*, a set of *Multirelations* and a set of *Contracts.*

An *Association* may be *Simple* or *Multiple*, *Multirelations* have a name and a set of *Destinies;* a *Contract* may be a *Create, CreateFor, Update, Delete, List* or *Delete* and may have a set of *Parameters.* A *Destiny* represents the way of obtaining information when managing the *Multirelation.*

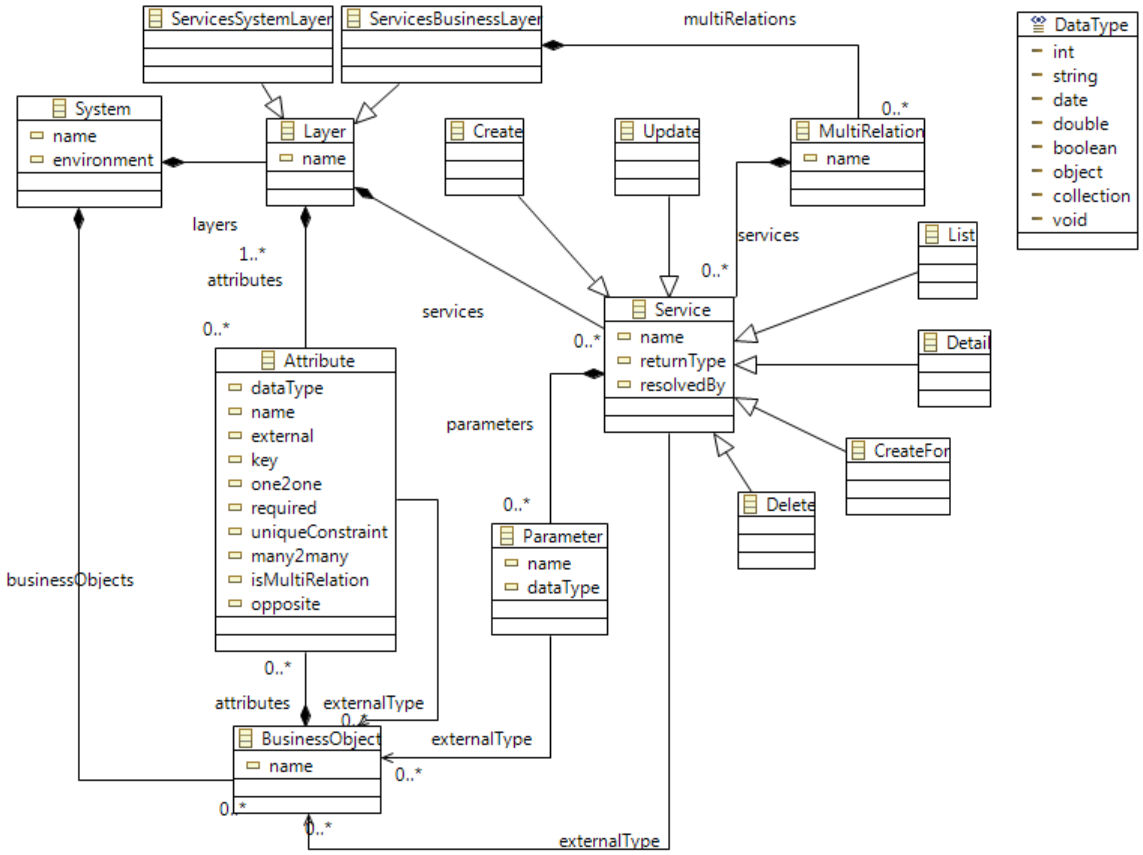


**Figure 16. Enterprise Architecture Metamodel**

The Enterprise Architecture Metamodel has concepts related to architecture domain. All concepts from a business model are transformed into one of the concepts shown in this diagram. A *System* concept is the root of the application, and it is composed by layers and business objects. A *Layer* has a name and a set of *Services* and another of *Attributes;* it may be a *ServicesSystemLayer* or a *ServicesBusinessSystem*. The latest may have a set of *Multirelations.*

A *Service* may be a *Create, CreateFor, Update, Delete, List* or *Delete* and may have a set of *Parameters.* A *BusinessObject* represents the element that will be used to communicate different layers, carrying on the information associated to a business entity.

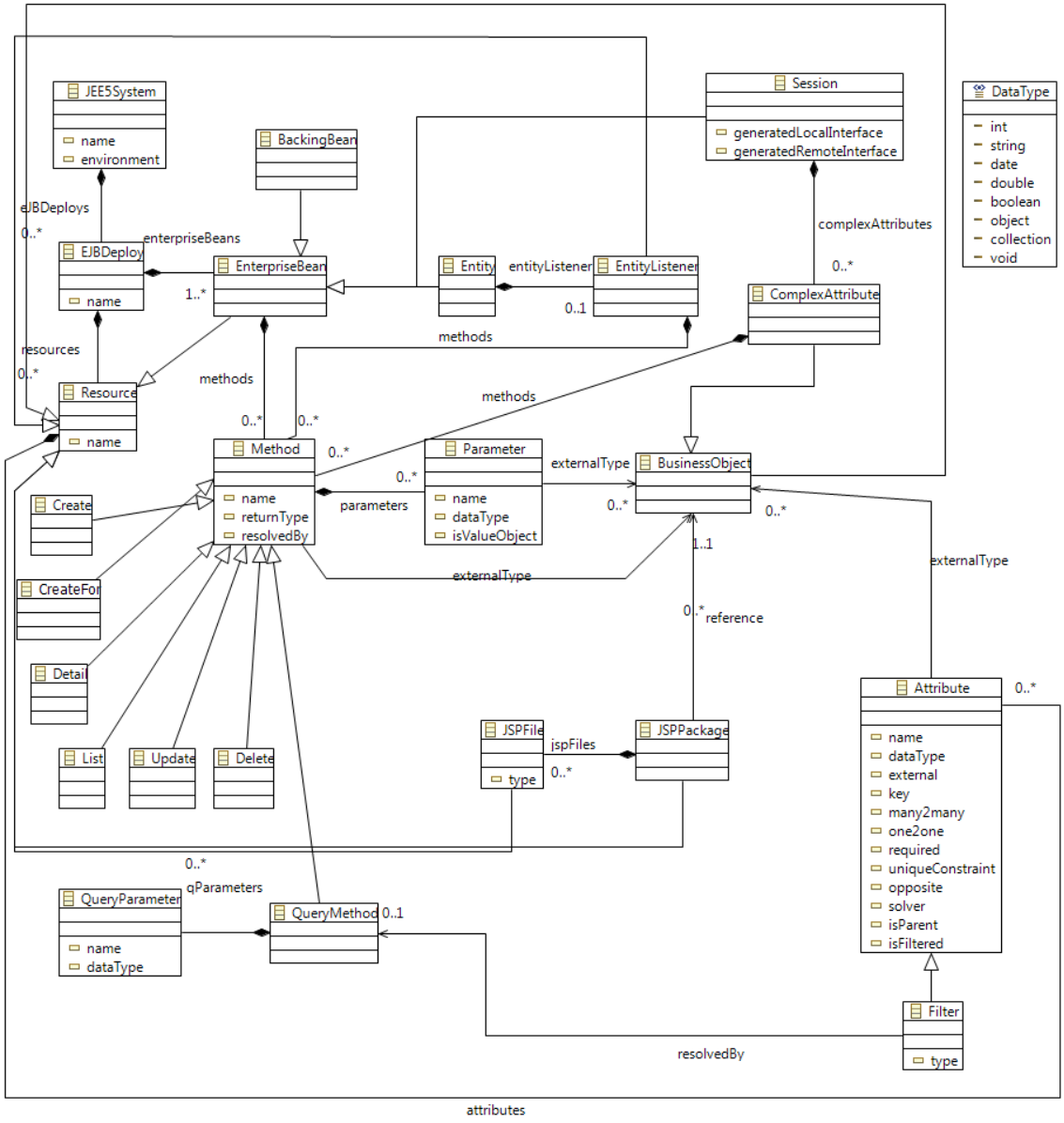


**Figure 17. Platform Metamodel**

This metamodel contains JEE5 and JSF concepts to represent the platform domain. All concepts from an architecture model are transformed into one of the concepts shown in this diagram. We have various types of *EnterpriseBean: Entity, Session* and *BackingBean*, each one related to data, business and presentation layer respectively. In this domain we talk about *Methods* that may be a *Create, CreateFor, Update, Delete, List or Delete* that may have a set of *Parameters;* or a *QueryMethod* that may have a set of *QueryParameters* and are related to *Filters* in the application. *BusinessObject* appear to keep communicating different layers of

the application. A *JSPPackage* has a set of *JSPFiles*, concepts that are related to presentation tier.

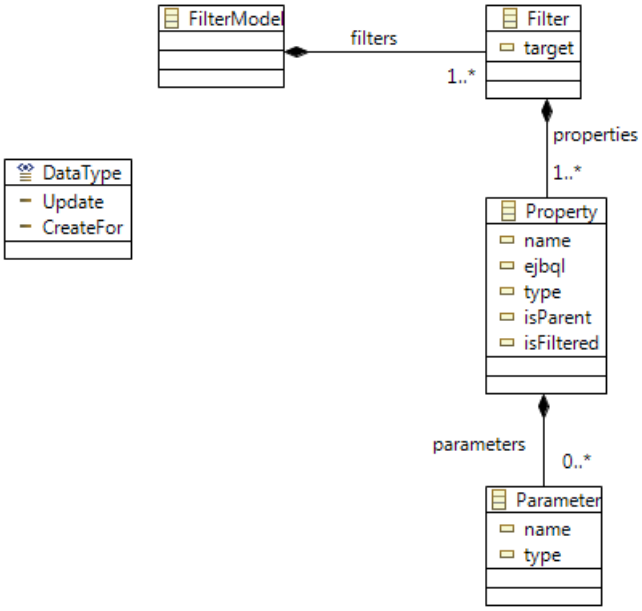


**Figure 18. Filter Metamodel**

The filter metamodel allows a modeler to express filters for the JEE5 platform. The root concept of the model is a *FilterModel* that is composed by *Filters.* Each *Filter* has a set of *Properties*; a *Property* has a name, an EJBQL query, a type (*CreateFor* or *Update)*, two fields that indicate how the property affects the filter and may include a set of *Parameters*. A *Parameter* has a name and a type.

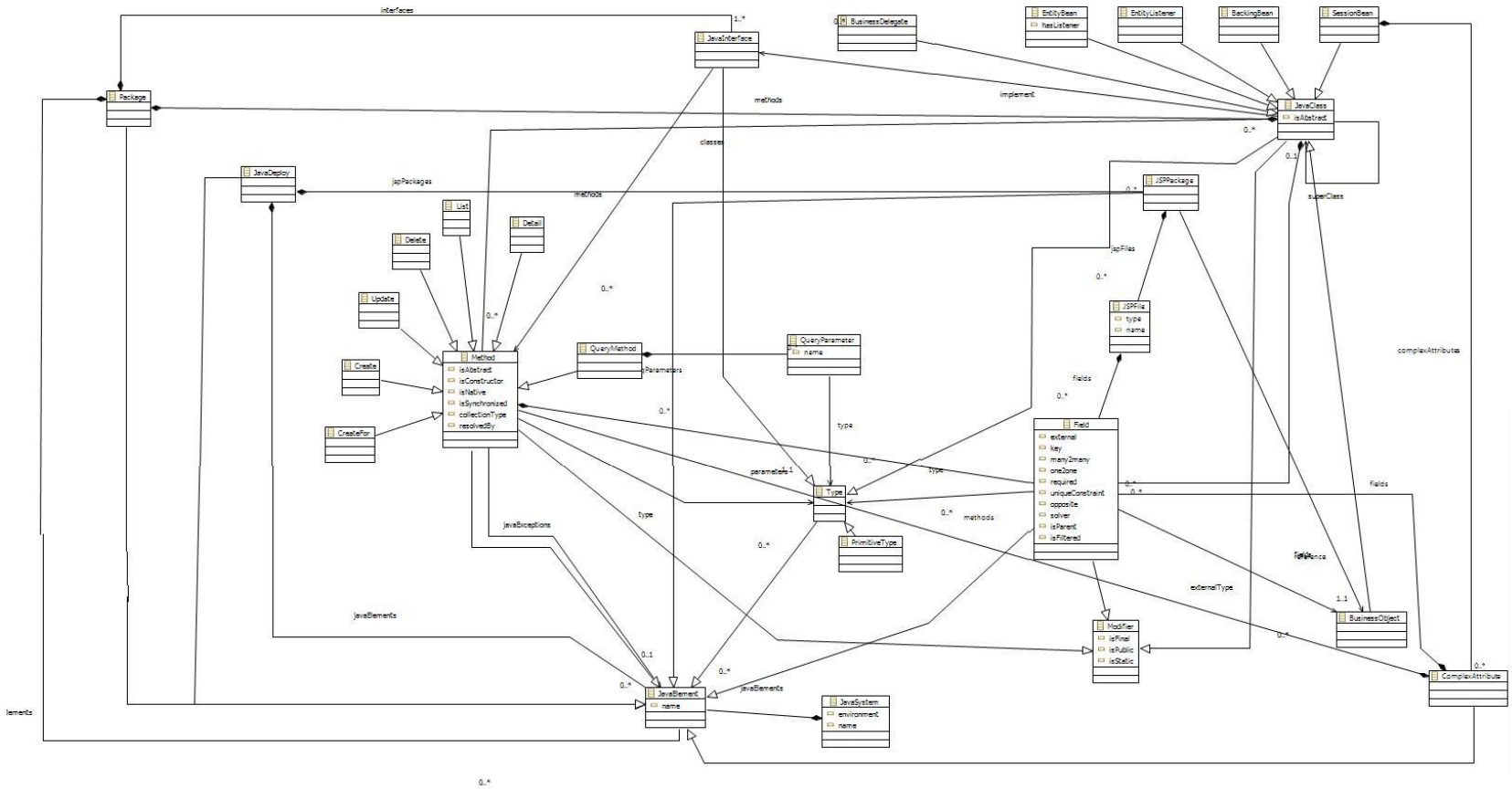


**Figure 19. JAVA Metamodel**

Java metamodel is the biggest one of the MD-SPL. *JavaSystem* is the root concept of this metamodel, and is composed by a set *JavaElements*. A *Package* contains a set of *JavaClasses* that are composed by a set of *Fields.* There are several specializations for the *JavaClass* concept to be able to generate different code based on the type of *JavaClass* they are. The metamodel contains different types of *Methods.* Presentation concepts such as *JSPPackage* and *JSPFile* also appear in this metamodel, for being able to generate presentation layer code from a Java Model.