Towards a Model Driven Architecture Approach within an Identity and Access Management Software

COLIN BONTEMPS

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2012

Towards a Model Driven Architecture Approach within an Identity and Access Management Software

C O L I N B O N T E M P S

DD221X, Master's Thesis in Computer Science (30 ECTS credits)
Degree Progr. in Computer Science and Engineering 270 credits
Royal Institute of Technology year 2012
Supervisor at CSC was Serafim Dahl
Examiner was Anders Lansner

TRITA-CSC-E 2012:098
ISRN-KTH/CSC/E--12/098--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

ABSTRACT

This project has been realized in the frame of a degree project for the school of Computer Science and Communication at the KTH: Royal Institute of Technology. The project set at the Company Arismore, near Paris (France). This project aims to explore the opportunities from Model-Driven Architecture (MDA) for designing the interfaces of an identity management middleware. The integration of such software is a long process. The hypothesis is that MDA could improve this process by providing consistency between functional needs and technical realization. For this purpose a MDA-tool was set, it is called Obeo Designer, based on Eclipse Modeling Framework. This tool allowed designing a meta-model for the configuration of target interfaces and several model edition tools. For the same model, different graphical edition tools have been designed. This allows a customer expressing its functional needs on a model, and a software integrator filling in this model with an adapted view for each actor. Generation of injectable code is then allowed through generation modules coded with Acceleo. Another method experimented was retro modeling. It allows creating model from existent configurations. This method allowed generating technical documentation and also analyzing existing projects. Conclusions are that this tools and methods allow the customer and the integrator at the company to work with the same data. A second improvement is to restrain the design of the interfaces to what is possible to be configured. Limitations come from the graphical model edition which do not provides a full implementation of all constraints. Several recommendations for functional design with models are finally proposed.

MOT EN MODELLDRIVEN ARKITEKTUR FÖR EN IDENTITETS- OCH ACCESSHANTERINGSPROGRAMVARA

Detta projekt har utförts som ett examensarbete på skolan för datavetenskap och kommunikation på Kungliga Tekniska Högskolan i Stockholm (KTH) för Arismore, ett teknikföretag nära Paris. Projektets syfte är att undersöka huruvida en modelldriven arkitektur (MDA) är att föredra vid design av gränssnitt för s.k. "middleware" för hantering av användaridentiteter. Integrering av en sådan programvara är en lång process och den grundläggande hypotesen är att MDA skulle kunna förbättra denna process genom att bidra med bättre överensstämmelse mellan funktionella behov och tekniskt genomförande. Ett MDA-verktyg, Obeo Designer, byggt på Eclipse Modeling Framework, har använts. Verktyget stödjer byggandet av en metamodell för att konfigurera målinriktade gränssnitt och ett flertal modellediteringsverktyg. För denna metamodell har flera olika grafiska verktyg tagits fram som tillåter en kund att bestämma de funktionella behoven för en viss modell och att skapa ett mjukvaruintegrerande verktyg för att komplettera modellen med gränssnitt som är anpassade för varje aktör. Exekverbar och injicerbar kod genereras med hjälp av Acceleo-moduler. En annan typ av experimentell modellering har använts för att skapa modeller från existerande konfigurationer. Denna metod stödjer generering av teknisk dokumentation och analys av befintliga projekt. En slutsats är att verktygen och metoderna tillåter att kunder och företagets integratörer arbetar med samma data. En annan förbättring är att framtagandet av gränssnitt begränsas till vad som kan konfigureras. Metodens begränsningar härrör från tillkortakommanden i editorerna som inte gav stöd för en fullständig implementation av alla restriktioner. Slutligen föreslås ett antal rekommendationer för funktionell design inom modellerna.

TABLE OF CONTENT

TABLE OF FIGURES

1 INTRODUCTION AND CONTEXT

1.1 INTRODUCTION

1.1.1 SUBJECT

The aim of this project is to explore Model-Driven Architecture opportunities and challenges in a concrete case study. This project has taken place at the Company Arismore in France. This concrete case targets the user interface part of an identity management middleware. In this section, the context which gave birth to this project will be introduced. Then the studied problem will be explained more in depth.

1.1.2 PREREQUISITES FOR UNDERSTANDING

1.1.2.1 MODEL DRIVEN ARCHITECTURE (MDA)

Model Driven Architecture (MDA) is a way to develop software, based on the use of models. It consists in using models from the very beginning to the very end of projects. In MDA models are not only a support for understanding but really the matter of development, from which code is then generated. MDA concepts will be explored in the literature review section.

1.1.2.2 IDENTITY MANAGEMENT

Identity Management is often associated with Access or Authorization Management within the field of IAM (Identity and Access Management). A definition of identity Management given by Oracle [I] is: “Identity management is a collection of processes that a company uses to manage the security life cycle of resources for its users, organizations, and entities.”

1.2 CONTEXT

1.2.1 ARISMORE: PRESENTATION

Arismore was founded in 2002 and is since lead by Eric Boulay. “Arismore is an innovative company which helps large companies and governments to transform and secure their Information Systems. Arismore proposes non classical practices and approaches. With its investment and experience, Arismore became a specialist in France of Enterprise Architecture and Information Systems Security.” from [A]

1.2.1.1 COMPANY ACTIVITY

Arismore is a services and consulting company with focus on the transformation of information systems.

Arismore helps its customers with the realization of benefits through:

- Consulting
    - Architecture of information systems
    - Program management
    - Simplification of technical infrastructure
- Achievement of security solutions of information system
    - Identity and Access Management
    - Management of service quality and performance
- Training and certification to practice
    - Enterprise Architecture and TOGAF® 9 (The Open Group Architecture Framework)
    - Service Management and ITIL
    - Identity and Access Management

Arismore represents The Open Group [A] in France through the Architecture Forum France [B], [C]. “

1.2.1.2 INNOVATION

Arismore supports several innovation programs such as:

Use of technologies as innovative and collaborative platforms

- Management of communicating objects for business industry
- Enterprise architecture portal
- Management of quality of service
- Management of resources capacities for Cloud Computing

Approaches for security and transformation of Information Systems

- Enterprise architecture and Enterprise security architecture (within international collaborative frameworks)
- Identity federation

Arismore is acknowledged by the French government for its research activities.

1.2.2 HOW THE SUBJECT CAME

Arismore is a pure player in the field of integration of identities and access management solutions.

Within the context of a project for the technical migration of an identity and access management (IAM) software for a major account, Arismore has set goals for management of consistency between functional and technical issues as well as for the optimization of developments.

Thus, a meta-model describing the software to integrate has been established and is used to formalize the technical specifications that lead the developments. This meta-model has also been defined to enable the industrialization of model-driven development (MDD).

The integration of the software through its configuration and specific developments is a long process that Arismore wishes to optimize by studying the possibilities to generate configuration files and code elements through a model-driven approach.

The IAM software that will be the subject of this concrete case study is called Identity Manager, produced by the editor Computer Associate. This software will be introduced in the next part.

1.2.3 IDENTITY MANAGEMENT AND CA IM

1.2.3.1 IDENTITY MANAGEMENT

Overview, from [I]:

“Identity management is a collection of processes and strategies that a company uses to manage the complete security life cycle of resources for its internal and external users, organizations, and entities, both within and beyond a firewall. An identity management solution can provide a mechanism for implementing the user management aspects of a corporate policy. It can also be a means to audit users and their access privileges. Identity management ensures the integrity of large application grids by enabling new levels of security and completeness to address the protection of enterprise resources and the management of the processes acting on those resources. Identity management also provides enhanced efficiency through a higher level of integration, consolidation, and automation, and increased effectiveness in terms of application-centric security, risk management, governance, and database integration. Identity management supports the full life cycle of enterprise applications, from development to deployment to full-blown production. In identity management, typical resources include Web access control applications, database servers, operating systems, legacy systems,

directory servers, and packaged applications (such as ERP and CRM applications). Entities refer to identifiable users of enterprise resources and can include employees, devices, processes, applications, or anything else that can interact in a networked computing environment. Entities managed by an identity management process can even include users outside the organization (for example, customers, trading partners, or Web services). In addition, entities are not limited to representing single users. An entity can refer to a group of users (also known as a role) or an organizational unit. ”

More details may be found in [J]

1.2.3.2 COMPUTER ASSOCIATE IDENTITY MANAGER (CAIM)

CAIM overview from [II]

“CA Identity Manager enables you to automate user provisioning and application access based on the relationship and role of each user within your organization-whether they are employees, contractors, customers or business partners. By employing CA Identity Manager, you will improve operational efficiency and reduce security risk by getting new users on board faster, reducing your help desk burden, and ensuring people cannot access certain systems. “

“CA Identity Manager provides an integrated method of managing users and their access to applications, including:

- Assignment of privileges through roles
- Delegation of the management of users and application access
- Self-service options so users can manage their own accounts
- Integration of business applications with CA Identity Manager
- Options to customize and extend CA Identity Manager “

CAIM provides two kinds of interfaces for its final users. The first one is the Administrator console used by resource managers for managing user and resources. The second one is called « self-service » and is used by common users in order to accomplish simple tasks such as changing password or asking for an authorization. Both these interfaces represent an important part of the integration process. All the references made in this document to interfaces and GUIs for CA IM will refer to both this interfaces.

1.3 PROBLEM

1.3.1 INTEGRATION OF CA IM AND FUNCTIONAL NEEDS

Why integrating CA IM is a complex process

The integration of CA IM for a customer is often more complex than it could be, because of the complexity in the expression of functional needs.

In order to deliver to what is expected by a customer, the interfaces of CA IM are customized. These customizations are accomplished through specific developments which are an important part of the integration process. Indeed the product does not natively offer all functionalities asked by most of the customers. User interfaces? play a major role in the expression of functional needs, as they are the only things that will be seen by the final user. The tool used by customers to define these interfaces is Microsoft PowerPoint in most of cases; and yet this tool allows drawing more or less anything. This freedom of design often implies several round-trips between the company and the customers, in order to find a compromise between what the customer want and what Arismore can create. Non-native configurations are used to get as close as possible to the needs.

This complexity in the expression of functional needs brings complexity in the integration process. This is translated in the use of java code, java script and html code in order to meet what is expected by the customer. The fact is that this customization of the product represents an important part of the integration process. Alhough in most cases the modifications added through code are not the most significant ones, just hiding a button at some point of the process might involve the use of java code.

This gap between the expression of functional requirements and technical integration is a source of complexity and is thus costly for Arismore and its customers.

1.3.2 Opportunities of MDA & Obeo Designer

Why MDA and Obeo could solve this problem

Arismore wishes to optimize the process of integration, and decided to explore the capabilities of Model-Driven Architecture (MDA). The point would be to allow the expression of functional needs using models which could also be used later on for the technical realization. Indeed MDA, by bringing developments to a new level of abstraction, closer to the functional concerns, and allowing automized code generation, may be a good way to improve the imperfect process.

The use of models can improve the process of integration in two ways. First, by providing more constraints than PowerPoint for the design of user interfaces by the customer (expression of functional needs), and secondly by making the data in these models accessible to allow, functional requirements to be easily transformed into a format useful for integration purposes.

At the time of the beginning of the project, Arismore has already chosen an MDA tool in order to explore the opportunities of this method. This software is called Obeo Designer and it is based on the Eclipse Modeling Framework (EMF). The Obeo designer is expected to allow easy and customized editing of models and also to integrate models in the integration process

This software will be presented with more details in the next section.

1.4 Summary

Arismore is a company of services and consulting. It wishes to improve the process of integration for a special identity management software. This process suffers of the gap between functional requirements and technical realization. MDA is a development method using models that aims to reduce this gap. One of the main parts of the functional userrequirements, provided by a customer, relates to user interfaces (GUIs). The goal of this project will be to explore MDA opportunities for the integration of these GUIs in the product CA IM.

2 LITERATURE, TOOLS AND INTERVIEWS REVIEW

This section will expose the points of view found in the related literature. First, Model Driven Engineering (MDE) will be introduced, with an historic, and three of its different approaches. Then the chosen tools for implementing MDA will be presented. Finally focus will be put on the target CA IM and its integration process. This section will not only reference research documents but also interviews with technical experts at the company Arismore and software documentations. As a conclusion for this section, the chosen approach will be presented.

2.1 MODEL DRIVEN ENGINEERING (MDE)

2.1.1 MDE AND MDA

The idea of a Model Based approach for coding emerged at the end of the nineties. The definition of the unified modeling language (UML) and its adoption by the Object Management Group (OMG) [D] in 1997 showed the interest of the programmer's community in models. The OMG's MDA initiative [3] [E] brought a lot of expectations as the Object oriented Programming (OOP) was showing its limitations in terms of optimization. Its principle was to make use of patterns in order to easily implement the same model for different versions and platforms. Not all of these expectations were fulfilled in enterprises, because of the complexity and abstraction of the concepts [9]. The scientific community was more optimistic about it and gave some suggestion for how to properly use this framework [4]. One may find in [5] or [10] a good overview of MDA capabilities and expectations at its beginning. Two main concepts in OMG's MDA are those of Platform Specific Model (PSM) and Platform Independent Model (PIM). A PIM aims to contain more conceptual information, contrary to a Platform dependent model (PDM) which is more oriented for code generation. In MOF a PIM is generally supposed to be transformed into several different PDMs, as a concept may be applied to several platforms. The concepts of PID and PDM exist at different abstraction levels and may concern as well simple models as meta models. They represent the horizontal dimension of MDA, as a complementary of the Meta concept which represents the vertical dimension. The concept of Meta model within computer science appeared at the beginning of the 21th century. It was then formalized by OMG in the Meta Object Facility (MOF) (2002) [F]. Meta means in some way to have one degree of abstraction higher. Basically Meta-Modeling is then building models for models. In its MOF specification, the OMG define a meta-model that is supposed to be of top-level. This model builds a framework for defining model, and as a top-level object, it can define itself, but also other models from OMG such as UML. This complex version was then proposed in three versions (2006): Essential MOF (EMOF), Complete MOF (CMOF) and Semantic MOF (SMOF). The one that will interest us in this document is the first one. EMOF provide a more understandable and usable framework than its siblings. The framework that will be used in this case study is called Ecore, and is aligned on EMOF. Ecore (2007) is a modeling language provided in the Eclipse Modeling Framework. Its usability and its integration in Eclipse IDE has made of it one of the most used tools for MDA nowadays.

2.1.2 DIFFERENT MDE APPROACHES

Despite being unified under the terms of Model Driven Engineering (MDE) or Model Driven Development (MDD), several different approaches are to consider. What they all have in common is the use of model, but it may take different places in the process of MDE. As presented in Figure 1, the model may be the source of the programming, and this is the way that brought the more expectations. But a model may also be a powerful visualization tool for existing code. A modern approach is to use model at runtime in order to have a continuous correspondence between code and model. These are the three approaches to be presented in this section. Before starting with these, it is interesting to consider the two extremes of the figure, which will not be developed further in this document.

One point that shall not be omitted is that even simple code always uses some kind of model. The model may be implicit or mental, but it is always here. Indeed code can itself be considered as a model, because it is the

representation of how the system will behave when executed [1]. The same action could be represented with another programmatic language, or in machine code. But in the code-only approach the model aspect is not a central object, and it is not explicitly written in a way to be easily understandable. On the other side, to the right of the figure, the model-only approach allows better understanding of some phenomena even without being linked to code as it may just be a pattern or may concern a field where code is not involved.

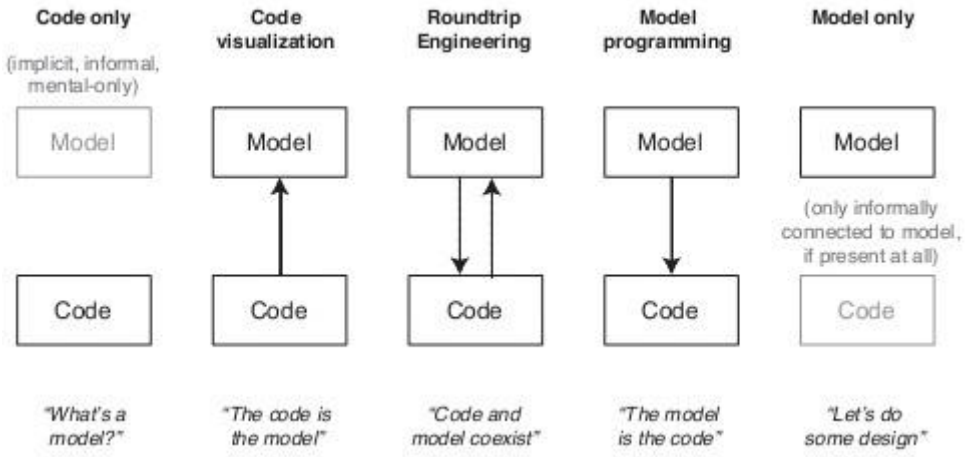

FIGURE 1 : THE MODELING SPECTRUM (FROM [2], ADAPTED FROM [8])

2.1.2.1 FORWARD MDE APPROACH

The forward MDE approach consists of taking the model as a basis for designing software. It is represented in Figure 1 at the fourth position. Besides giving a clearer view of the structure, it aims at improving the quality and the speed of code realization. This approach was expected to become a new layer in programming activity designed to hide the rising complexity in modern software realizations. Indeed many main advances in computer sciences have been made this way; such as having programmatic languages as an upper layer of a complex machine language. MDE could hide the complexity of programmatic languages, being an upper layer of abstraction. Ideally, it could bridge the gap between functional and technical aspects of a system, allowing avoiding most of the coding and using patterns for generating code, in the way that an assembler does at a lower level. Therefore it has to be compared with software factories (such as authors do in [1]).

2.1.2.2 REVERSE MDE APPROACH

The reverse MDE approach creates the model from existing code. It is represented at the second position in Figure 1. Having models coming from code may have an interest for improving the understanding of the code, its structure or detecting structural flaws. Moreover,, being able to extract the information of a system in a functional way allow using forward MDE in order to generate it for other platforms or for other versions of the same platform. Indeed, during the migration of a system, the functional aspect often stays unchanged.

More on reverse engineering in [6]

2.1.2.3 RUNTIME MDR APPROACH

Run time model is a modern approach of MDE which interests more and more researchers [K]. As the name suggests, it consists in having a continuous correspondence between model and code, even at runtime. In [1], Robert France and Bernhard Rumpe explain why they think this kind of MDE should become increasingly important. They focus particularly on its advantages for monitoring, adapting and correcting design errors of running systems.

More on runtime engineering in [6]

2.2 EMF & OBEO DESIGNER

The Obeo Designer is an MDE tool based on the Eclipse Modeling Framework (EMF) [III], which offers domain specific modeler (DSM) creation and editing without coding [12]. The concept of DSM is strongly related to the one of PSM as defined in the MOF specification for MDA. EMF alone does allow creating a meta-model in different formats. One of these is Ecore, which is a direct simplification of the complex MOF-defined language for meta-modeling. It is then possible to generate java classes from this meta-model, and to export them to an independent plug-in. In another instance of eclipse it will be possible to instantiate the model corresponding to the meta model. EMF provides a graphical editor for custom models as a tree diagram view.

What Obeo does that Eclipse MF does not is to allow easily defining one's own graphical view for custom models. Where this could be done programmatically with Eclipse Graphical Modeling/Editing Framework (GMF and GEF), Obeo Designer introduce an upper layer allowing doing this without expertise and nearly without any single piece of code. The principle is to synchronize a style sheet defined over the meta-model with instance-model data. This style sheet allows defining several kinds of diagrams and tables as points of view of the model data. The Obeo style sheet for defining views is called an Odesign file. Each point of view may include several kinds of self-defined diagrams and tables, showing more or less complexity, or with focus on different concerns. The idea of points of view for Obeo is one point of view for one concern.

This vision of MDA could look like restricted because model transformation are not part of the process. In facts EMF provides some tools for model and meta-model transformations, so that it is possible to work with several meta-models over each of which an Obeo style sheet is defined. This way of using MDA is then compatible with the three kind of MDE evoked earlier. EMF provides as well a Model2Text language which is a strict implementation of MOF specification for its kind. This language is called Acceleo and has been developed by the Obeo team. Acceleo provide easy access to model date in order to generate all kinds of files, as used in forward engineering.

2.3 THE TARGET : CA IDENTITY MANAGER

2.3.1 CAIM: IN-DEPTH VIEW

All information presented further comes from several different manuals from Computer Associate and from interview and discussion with people working at Arismore. For more details see [G], [V] & [VI]

The area within CAIM that will interest us in this study is the one involving Graphical User Interfaces (GUIs). In fact Arismore develop and customize functionalities of CAIM so that its clients can benefit from features and GUIs that meet their specific needs.

These GUIs (see Figure 2 & 3) proposes services such as creating, viewing, modifying, or deleting entities. An **Entity** may be a user, as well as a role or an organization in the customer company. **Roles** have for purpose to correspond to one profession. One might cumulate roles for a single job. A role gives access to the appropriate features and applications, and only to them. The central concept of these features in CAIM is called Task. **A Task** describes the whole process of a single feature. It can be used for viewing, creating, modifying or selecting entities. A Task is visible only to certain users. This is done by associating a Task to one or several Roles. These roles may then be attributed to concerned users.

In order to handle entities such as users or organizations, CAIM is connected to a data base (LDAP) called User Store. Within the environment defined for CAIM, each type of entities within the User Store has its own attributes and properties. These attributes accessible through the User Store are called **Physical Attributes**. It might be the first name of a user. In order to be displayed or stored, physical attributes sometimes need to be formatted or modified. For instance, a date might have different storage and display formats. In the database the date might be stored as "DDMMYYYY" but have to be displayed as "the DDth month YYYY". In order to

handle this treatment, CAIM uses **Logical Attributes Handler (LAH).** A LAH calls to (is) a Java class that converts or formats an attribute, for both storing it and accessing it. Each LAH define one or several **Logical Attributes,** which are the formatted version of one or more physical attributes. Logical Attributes can then be accessible in the same manner as the physical attributes when defining the GUIs. These GUIs to be defined are always displayed in the frame of the use of a particular Task.

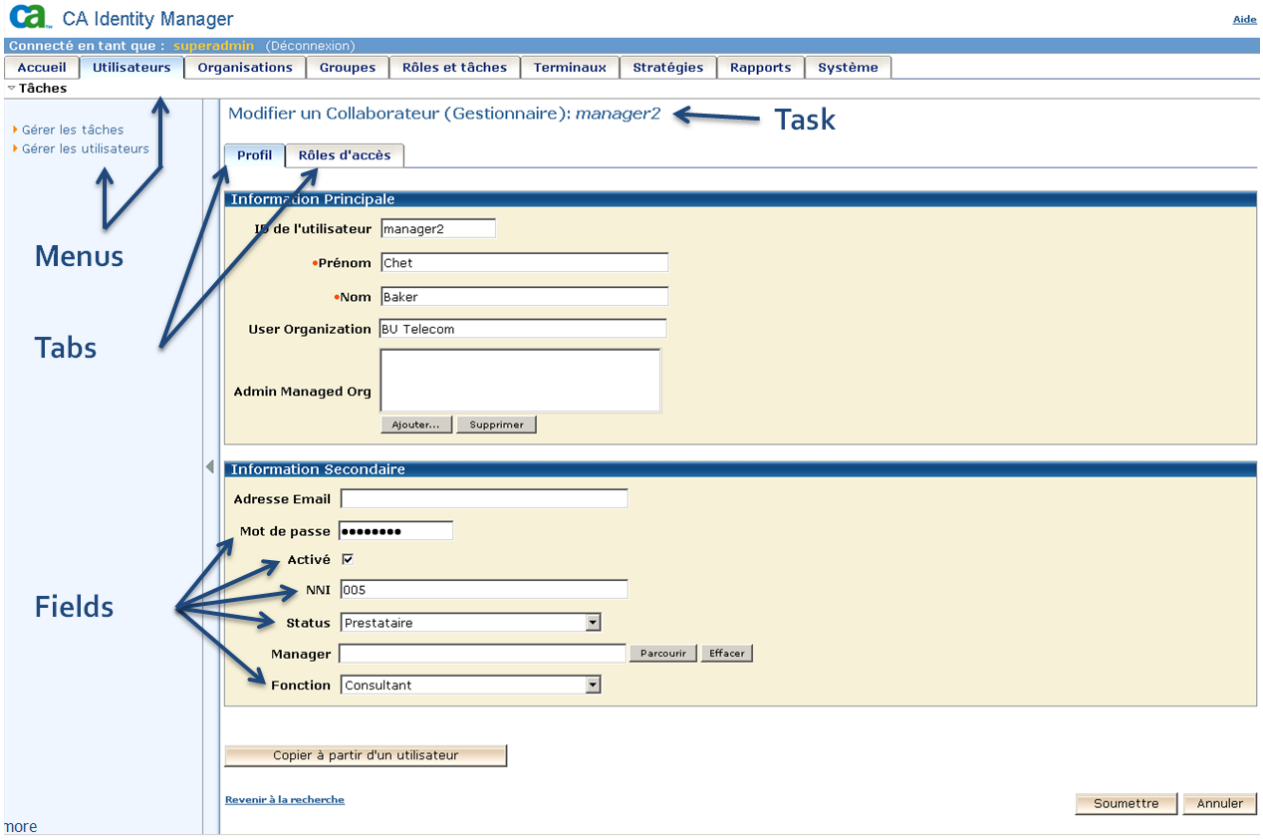

FIGURE 2 : SCREENSHOT OF A GUI FROM CAIM

A Task may include one or several **Tabs** each of these points to a **Screen.** The concept of Tab in CAIM is almostthe same as in a web browser. The main difference is that tabs are here handled by a tab controller, which specifies if they are to be accessed simultaneously or one by one in the right order. A Screen is the interface for reading and modifying attributes from the User Store. It is composed of **Screen Field** and **Separators.** Each of the Screen Field is bound to a single attribute; it is Physical, Logical or Internal to the Screen (**Screen Logical Attribute**). A field can be of several display format styles such as Text, Check Box, Option List, User Selector, etc. Each field has permission on its attribute, such as Read Only or Write Once. A selector field may be bound to a **Search Screen.** A Search Screen is a special screen for searching in the database. It is composed of **Search Fields** and **Result Fields** which are similar to Screen Fields, with less widget styles available. A search screen is defined for on single type of entity (User, Group, etc.) Separators in Screens are disposed like Fields but are not associated to an attribute. It can be layout information (page section), content such as html or pictures, or also pointer to another task (Nested Task).

A Task can include one or several **Business Logic Task Handler (BLTH)** which allows calling java code at different step of the execution of the task, as before or after validation of the task. An **Event Listener** allows calling Java code at different approbation steps of some tasks. An event may be user creation, and an Event Listener may be called before and after approbation of the user creation by a manager. A **Category** is a container for Tasks. There are three levels of categories that form a menu from where the tasks are accessible in the user interface of CAIM.

All these objects are the ones that will interest us in this study. In order to configure this objects, CA IM allow importing and exporting them. The file format used is then XML.

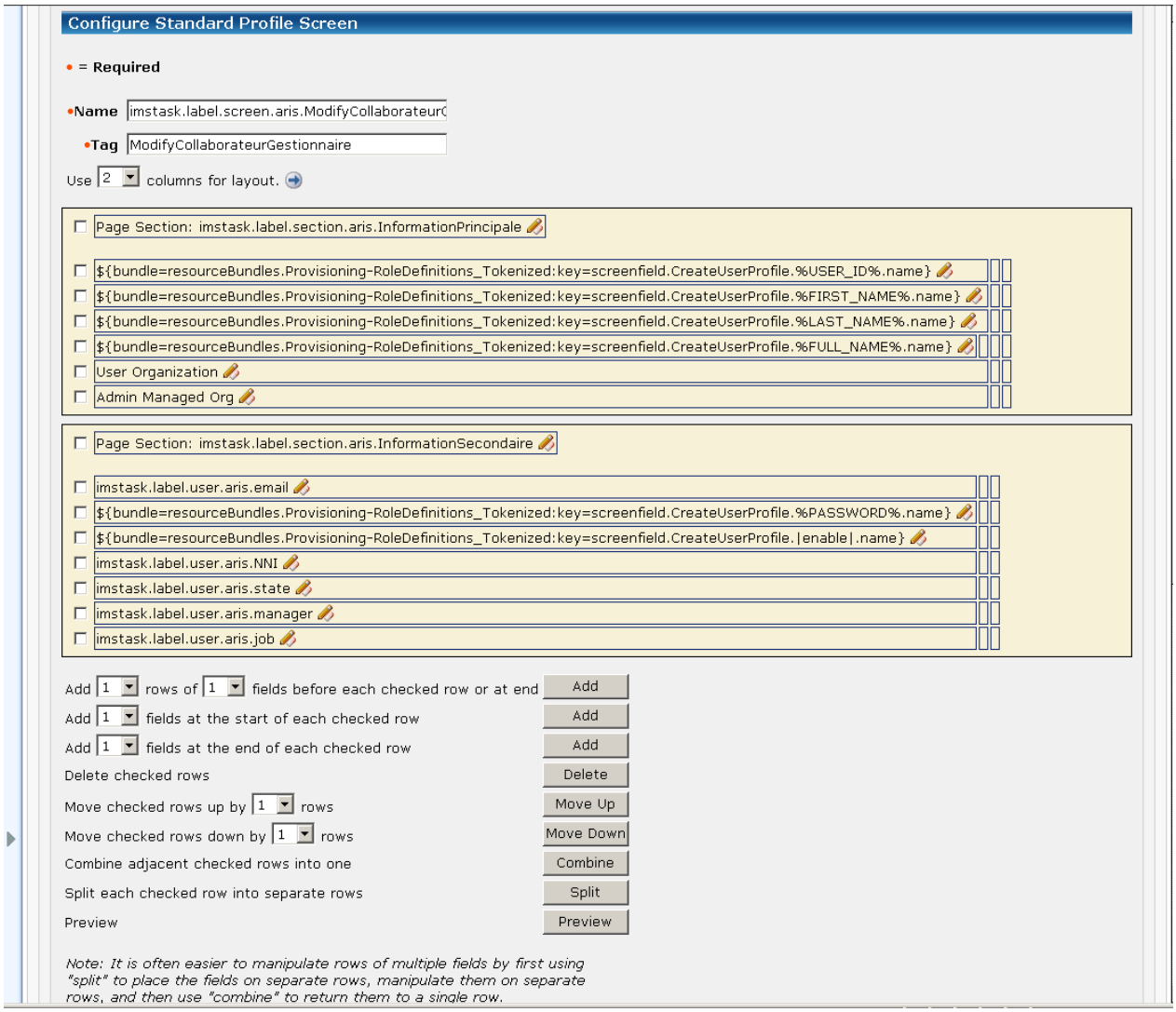

FIGURE 3 : NATIVE CONFIGURATION OF A SCREEN

2.3.2 Structure of Configuration Files

Now that all the objects to model are defined, the data structure will be presented. All this objects, with some others are exported and imported with two XML files into CAIM solution.

- The first one, “Environment Settings” includes LAHs, global BLTHs and Event Listeners.
- The second one “Environment Objects” includes Screens and Tasks

A Task may contain: Tabs and BLTHs.

A Screen may contain:

- One “Field Group” named Field or
- Two “Field Groups” named Result Fields and Search Fields, if the screen is a Search Screen.

Field Groups may contain Screen Field and Screen Field Separators

Handler objects such as LAH, BLTH and Event Listener must make use of java code. Each of these objects refers to a single class. This class implements the corresponding methods provided by the CA API.

In order to support multi-languages GUIs, the label of every displayed object is stored in several properties files. There is one properties file for each language

2.3.3 The View from Integrators

At the beginning of this degree project, it was important to know precisely which were the problems faced by the people working with the integration of CA IM. After several short discussions with different "integrators" and an interview with Benjamin Petitprez an expert in this domain at the company, several problem areas had been identified.

What is referred to as "integration" is an activity wider than developing or coding, although those activities is probably also part of the process. Integration is a reference to "integration of the functional needs". This process starts from helping customers write down their functional specifications, and ends in performing the tests on the final identity management system delivered.

This process is stated as long and complex, and the company wishes to optimize it. Therefore they want to explore industrialization automation through MDA. The first problem for the integrator is that it will not be possible to get a completely automated process that creates the technical configuration based on the functional needs. Indeed there are choices that must be made at some point and there might also be complex java methods to write. This clearly implies that the process will always require human intervention.

When asked about the impact of complex functional needs, Benjamin answered that java enhanced formula-fields in GUIs, may represent up to 80% of the total fields. The use of JavaScript for these fields may concern between 10% and 70% of the fields, depending on the complexity of the need. Then when asked if this complexity could be avoided, he answered that probably not in most cases, because the product does not propose enough native customizations (without coding) in order to answer the functional needs of most customers.

Besides, the definition of functional needs still suffers from many misunderstanding, and brings many round-trips between the company and its customers.

2.4 Chosen Approaches

The different MDE approaches than can be used have been presented. The capabilities of the chosen tool have been explained. This brings us to the choosing of which approach to start with. The main focus will be on forward MDE because it is the one that allows developing from models. Indeed the goal of the project is to simplify developments for projects to come. The possibility for reverse engineering will be explored in a second iteration, paving the way for a more attractive runtime MDE.

3 How to Implement MDA for CA IM integration

The problem identified at the company is the complexity appearing in the process of integration of CA IM, because of inappropriate methods and tools for expressing a customer's functional needs. This complexity is particularly expressed within the definition of user interfaces. Indeed this complexity is costly for both the company and the customer. The way studied in order to solve this problem is Model-Driven-Architecture. After introducing the matter, the related literature and the tools chosen, this solution can be detailed and be given precise objectives. In this part, detailed objectives and planning will be specified from the literature and the context.

3.1 Several Points of View

From the previous sections, it is stated that we seek to improve the integration process. This improvement is expected to come from the MDA method. There are several improvements that can be made in the given context. One is adding constraints during the design of the GUIs by the Product Manager in the functional needs expression. This wouldrestrict the customer's freedom to design complex GUIs, where it is not possible to do so natively in the product. As Obeo proposes to define several graphical model editors, it is interesting to experiment what can be done by designing user interfaces with models. Models have much more constraint than a PowerPoint sheet and have the potential to represent advanced interfaces. This will then be a part of the experimentation to perform.

Then another point for optimizing the process of integration would be to be able to reuse the data from the GUIs-models to perform integration. This would be possible by generating code from the model, and it is also one of the possibilities offered by Obeo Designer with its plug-in Acceleo.

Another improvement that can be made is to take into consideration different actors in the process. Actors include at least the Customer's Product Manager on one side and the Integrator of the software on the other side. Both of them have different points of view and different concerns during the integration process. The Product manager knows what kind of functionality he expects and what he would like the GUIs to look like. The integrator know presicely how to configure the product to produce the expected results. He is more focused on the parameters of this configuration than on the functionnal design. This highlights the Obeo Designer principle from the last section: one concern = one view point. So there will be at least two viewpoints.

3.2 Target Process

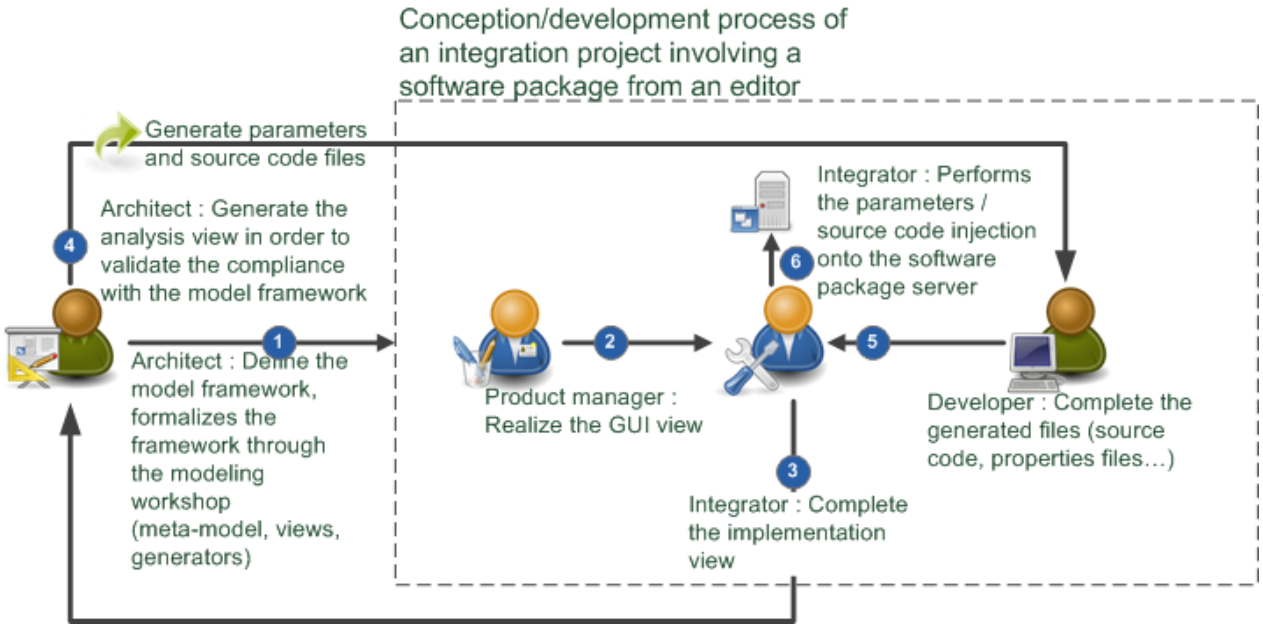

FIGURE 4: TARGET PROCESS FOR INTEGRATION OF CA IM

For organizing the work around this idea of MDA tool, a collaborative process has been defined by the company. It involves the two actors already evoked: Product Manager and Integrator. Two new actors appear in the diagram (Figure 4). The developer is the person writing the code for the non-native configuration, because, as explained before, it is not possible to entirely automate the process, and also code will always be needed in order to satisfy a customer's need. So the developer needs to be there for coding purpose. The Architect is an important stakeholder in this process since he is defining the work frame and tools. It is more or less the work of the architect that will be experimented in this project. This work allows the other actors to work together by defining an architecture framework and a framework reference using Obeo Designer.

3.3 OBJECTIVES

1 - Using the tool Obeo Designer, create and integrate a meta-model with all elements enabling the integration of the Computer Associate Identity Manager (CAIM) software GUIs:

- Define a meta-model for the GUI's configuration
- Establish the representation of elements from the meta-model in the tool

2 - Define asymmetric generation modules with usage scenarios, coming from integration projects of the IAM solution:

- Implement generation modules from elements of the meta-model.
- To generate all the elements of code (Java class) and configuration files (XML files, ldif files, files. properties).
- Validate their relevance to the project by injecting the generated elements and comparing these with those made by a developer on the same usage scenario.

3 - Develop two compatible user oriented modeler tools in order to have different actors working together from different points of view:

- By specifying two Obeo "viewpoints" specific to a functional and a technical point of view of the same use case.

4 – (Two options)

- Test the possibilities of reverse engineering of the MDD tools against the elements and configuration of existing code for the project.
- Or, develop a third viewpoint enabling analysis and validation of the model prior to files/source code generation.

3.4 SCOPE

The only part of the configuration that will be treated in the project is the one concerning GUIs. Meta-model, generation modules and modelers to define are thus limited to what is needed for the configuration of the CAIM GUIs. Retro modeling-tools and analysis view, if done, are to be in this same scope.

Two viewpoints are expected to be designed within the modeling tools. The first one is an accessible viewpoint restricted to the functional aspects of the GUI's. The second viewpoint is expected to allow the integrator to use all available meta-objects and thus will include more technical information. For this, the view will handle deeper layers such as java classes and handlers. This is required in order to generate as much content as possible for the code skeleton and the configuration files.

3.5 ORGANIZATION OF WORK

3.5.1 TIME SCHEDULE

Task	Time allowed	End date
Step 1 : Information gathering and education	**4 weeks**	**07/10/2011**
Sub goal: Be able to deal with the MDA and IM contexts, to specify the range of the Obeo tools.		
Information gathering on Model Driven Architecture	1 week	
Information gathering on Identity Management	1 week	
Obeo Designer, discovery through an example	2 weeks	
Step 2 : Establishment of a meta-model and code generation	**7 weeks**	**25/11/2011**
Goal: Have working generators modules, validated with: *1 - comparison with "hand-made" ones* *2 - integration in the software*		
Information gathering about CAIM and its files to generate	2 weeks	
Analysis of the configuration files to generate	1 week	
Establish a first meta-model draft for CAIM's GUIs	1 week	
Create Acceleo generator modules	1 week	
Comparison, test and validation of generated files	1 week	
Report writing	1 week	
Step 3 : Define Modeling tools for Product owner's and Integrator's viewpoints	**9 weeks**	**27/01/2012**
Goals: *1 - Have a comprehensive and operational meta-model for the viewpoint modeling and the generators.* *2 - Have two working viewpoints, with appropriate diagrams and tables, adapted to corresponding final users.*		
Specify the range for each viewpoint	1 week	
Refine meta-model according to expected views	1 week	
Define both viewpoints	5 weeks	
Validation and improvement of views with usability concerns	1 week	
Report writing	1 week	
Step 4 : Test model retro-conception possibilities	**4 weeks**	**24/02/2012**
Goal: Have explored possibilities to generate a model from the configuration files.		
Extract the useful information from configuration files	1 weeks	
Deploy a model from files	2 weeks	
Report writing	1 week	
Step 5 : Conclusion	**2 weeks**	**09/03/2012**
Goals: *1 - Have an exhaustive report about the work done, the results obtained and the methods used along the degree project.* *2 – Build a presentation, support to return the finding of the work term project to the company.*		
Look back on the work and prepare the presentation	1 week	
Finalize the report	1 week	
Total	**26 weeks**	**09/03/2012**

4 METHODS AND RESULTS

4.1 METAMODELING

4.1.1 METHODS

4.1.1.1 INTRODUCTION

Within the frame of this degree project, a Platform Dependent Meta model has been defined. Its concern is the configuration of Graphical User Interfaces (GUIs) for identity and authentication management. The platform targeted is Computer Associate Identity Manager (CAIM).

In order to create this meta-model, the first step was to make a deep analysis of the product functions that were target of the modeling. The first input for the meta model were example models from existing projects at the company. They brought a first constraint for the Metamodel, because they had to be compliant with it. From this has been defined a precise scope for the meta model, to know exactly which object were to be included. This area had to be coherent and also suited for the generation purpose. A part of this work had been done before the beginning of this study project at the company, so there was an idea of what was the global scope of the task. Still, much work was left to do in order to define exactly which objects to create.

What would have made the task easier in the realization of this meta model was an exhaustive set of constraints for it. This was nearly impossible to obtain as it was not even present in the software documentation. Constraints came from different sources: the xml files to generate, the native configuration of the GUIs, the software documentation and also discussions with some integrators within the company. Also these sources were not always as precise and coherent as whished.

From all these sources came a first version for the meta-model. This version evolved with time, as the metamodel was used for new applications. That has been a bit complex because every time the meta model changes, these changes must be applied for all its applications already made.

The format used for the meta-model was the Ecore Model, the native format of EMF, as it provides a good graphical representation and a graphical editor. This was really useful because the metamodel quickly grew big. The graphical representation allows easily focusing on what you are modifying and gives a clear view of its implications and bounds.

4.1.1.2 ANALYSIS METHOD

The meta model has been designed simultaneously with the generation templates. Indeed, the fact that the meta model allows generating the target files properly is a first quality criterion. The process is illustrated in figure 6.

4.1.2 RESULTS

As seen in Figure 5 the whole meta-model is designed as a base for the target functionalities. It includes classes of which some are abstract and enumerations. An enumeration is a list of values that defines a data type. They are useful for all class properties which can only take a limited set of values. In this section several problems and matters encountered while designing this meta-model will be discussed. When considering these problems in hindsight, they may be considered as design patterns, or good exercises for future meta-modeling works.

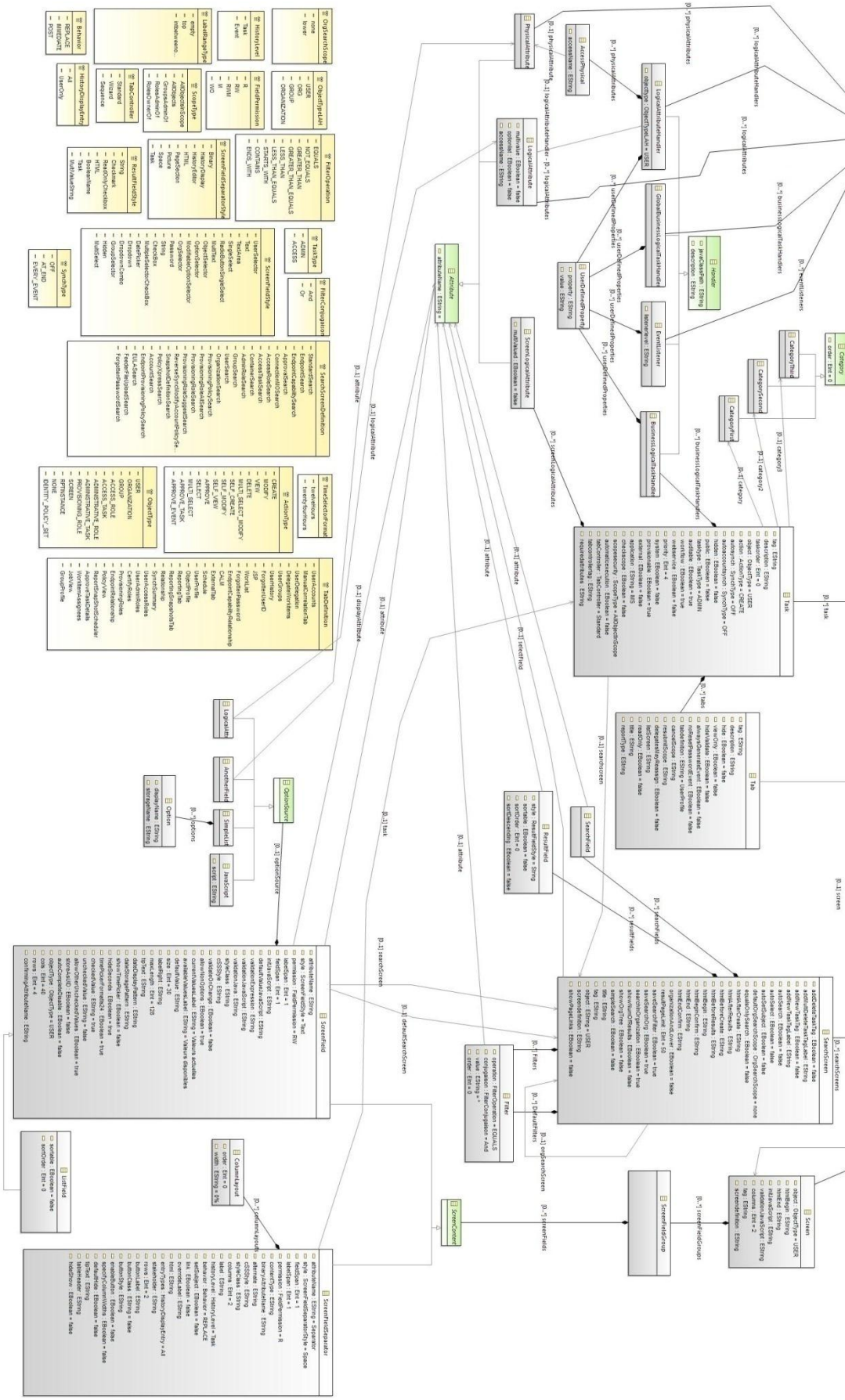

FIGURE 5 : FULL META MODEL OF CAIM GUIS

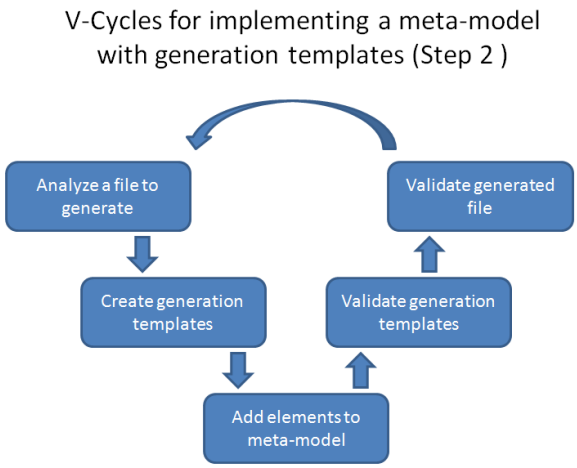

FIGURE 6: V-CYCLES FOR DESIGNING METAMODEL

4.1.2.1 METAMODEL DESIGN PATTERNS

4.1.2.1.1 RELATION'S ATTRIBUTE

This problem was encountered when modelling the relation between a Logical Attribute Handler (LAH) and a Physical Attribute. For each Physical Attribute referenced by a LAH, an access name for it is given; this access name defines how the attribute will be called in the associated java class. As this access name may be different from that of another LAH using the same attribute, it had to be defined for neither of these objects, but rather for the relation between them. To link an attribute to a relation between two classes is something that is possible with UML but not with Ecore. In order to reproduce this pattern with Ecore you must create a class for the relation.

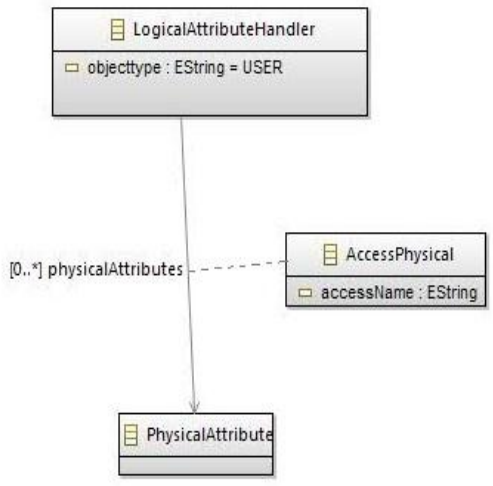

FIGURE 7 : RELATION'S ATTRIBUTE WITH UML

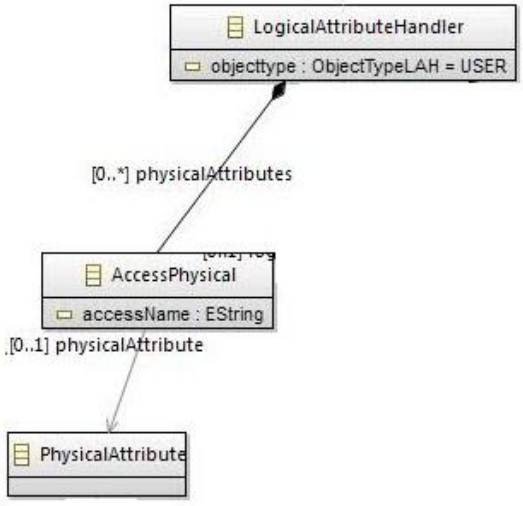

FIGURE 8 : RELATION'S ATTRIBUTE WITH ECORE

4.1.2.1.2 ONE CLASS FOR ONE CONCEPT

The object "Screen Field" in CAIM has an attribute "style" which may take different values ("Text", "Dropdown", "Checkbox"…) Depending on this value the object Screen Field have different properties which are relevant to it (length, label …). As an example a field of style "Text" has an attribute "maxlength" while a

field of style “Checkbox” is not concerned by it. Another example, a field of style “GroupSelector” has a search screen associated to it while a field of style “Hidden” does not.

In order to model this structure, the first choice made was to use an abstract class “Screen Field” with a different implementation class for each of the “style” values. This had the advantage of having only the relevant properties for each kind of Screen Field. Then the second choice made was to use a multiple-heritage approach so that properties which are shared by several fields are only present once in the model. This allowed easy modification and also unified access to a property. With that approach it was then possible to generate the Screen Field code with one single template. This template included or excluded a property for a field depending only on the classes it inherits from.

On the other hand, this approach had a big weakness. As a Screen Field represents a single concept, one might want to change its “style” attribute. This modification implies creating a new instance of Screen Field in another implementation class and then to copy into it the shared attributes. This is complicated but possible. For instance, supposed that the style change was a mistake, if the user wants to come back to the old value, all properties, which were not shared between the two Field styles, are then lost. This feature was, however, important for usability and it brought us to adopt another, simpler structure.

The new and definitive structure was a single Class that owns all the attributes for all kind of fields. This choice came after the confirmation that it was possible to choose the editable attributes for a class within the model editor. Then, all attributes are stored anyway, but only the relevant ones are displayed in the model editor. The main drawback of this new approach is the handling of default values depending on the field style. This is easy to handle at the creation of the field, but not on a style change.

The latter solution adopted is finally closer to the reality than the former, indeed screen fields are implemented by a single object in the .xlm files.

4.1.2.1.3 PROPERTIES AS CLASSES

Some objects had some properties that were not suited to be represented as a class attribute. For example, a Search Screen may have filters applied to the result showed. In the XML file this is represented like in Figure 9 : Filters for Search Screens in the XML

```
<Property name="Filter.0.Field">%USER_TYPE%</Property>
<Property name="Filter.0.Op">EQUALS</Property>
<Property name="Filter.0.Value">gaia.type.generic</Property>
<Property name="Filter.1.Conj">Or</Property>
<Property name="Filter.1.Field">%USER_TYPE%</Property>
<Property name="Filter.1.Op">STARTS_WITH</Property>
<Property name="Filter.1.Value">gaia.type.SIRH*</Property>
```

FIGURE 9 : FILTERS FOR SEARCH SCREENS IN THE XML

As we do not know how many filters there will be beforehand, a class Filter has been created in the metamodel. This class allows handling as many Filters as desired without having to take care of these attributes in the class Search Screen.

This pattern has been used several times in our metamodel. As it increases metamodel complexity, it should be avoided when unnecessary.

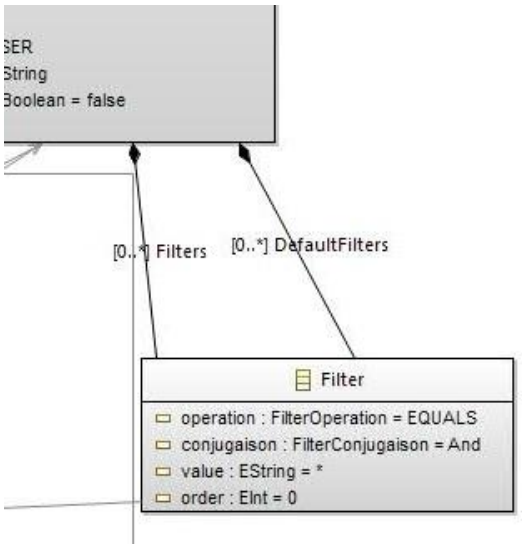

FIGURE 10 : HANDLING OF FILTERS IN THE METAMODEL

4.1.2.1.4 METAMODEL STRUCTURE: THE CONTAINMENT TREE

In order to access easily the elements from one to another, it is useful to have a structure of "containment tree". All elements must be contained in a Root element. For example the Class "Tab" is contained in the class "Task" itself contained in the class "Tasks" itself contained in the root class "Super Root". To Exclude a Class from this tree is not a good practice as a non-referenced element might then not be accessible from any element but itself. Using some Classes only for structural containment is good practice, as it allows loading only one portion of the model, for generating or editing purposes.

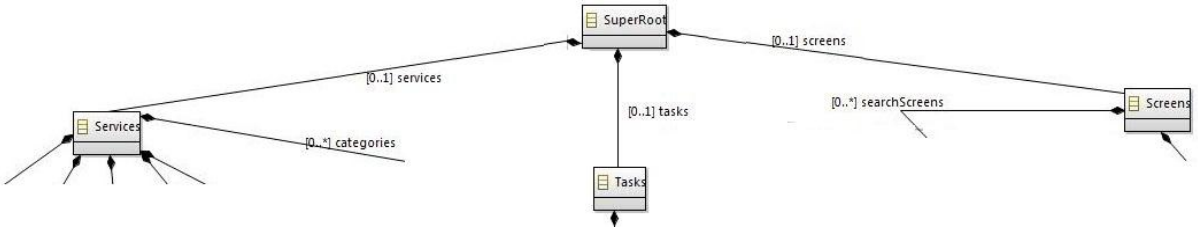

FIGURE 11 : CONTAINMENT TREE IN THE METAMODEL

4.1.2.1.5 OBJECT CONSTRAINT LANGUAGE (OCL)

For some cases, constraints must be written using the Object Constraint Language OCL. For instance the class "Screen Field" points to one single "Attribute". "Attribute" is an abstract class than is implemented by three classes: "Physical Attribute", "Logical Attribute" and "Screen Logical Attribute". A "Screen Logical Attribute" belongs to a Task, so that in order to use it a Field must be within this Task. In concrete terms only a field in a screen pointed to by a tab belonging to this task may use this attribute. For this pattern, there was no other choice than to use OCL, because the Screen Field can only point to one single "Attribute".

4.2 CODE GENERATION

4.2.1 THEORY

4.2.1.1 MODEL TO TEXT TRANSFORMATION LANGUAGE

A model to text language allows easy access to the model-data and conversion of it to character-strings. As defined by the Object Management Group (OMG) it may be used to transform the model into code or documentation. In [11] the OMG defines a standard for Model to Text (M2T) Languages, with syntax and

functionalities to be found in such a language. Most of this information may be found in the MOF's metamodel for M2T Languages.

4.2.1.2 ACCELEO AS MOFM2T LANGUAGE

The language used within this degree project is called Acceleo. It is a strict implementation of the OMG Standard. Acceleo has several advantages compared to some other M2T languages; especially it is integrated in Eclipse IDE and proposes quick fixes, code folding and an advanced traceability for the files generated. For more details see [H] & [IV]

4.2.2 METHODS

4.2.2.1 WRITING

In this project we aimed to produce three kinds of files with code generation:

1. XML files for import in CAIM
2. Java code Skeletons for implementation classes
3. Properties file for labels (in both the French and English language)

For all these three cases the process is the same. It must start with an example target file taken from a project and includes programmatic parts into it. With Acceleo these programmatic parts must be surrounded by [and].

It is then possible to organize the generation into several generation files, one for each concern, which each can be imported by an other. Within a generation file (also called "generation module") functionalities may be organized in templates and queries. The M2T language allows making full use of loops and conditions in written code.

A good practice is to create a "launching module" for each "generation module" so that the generation can be easily launched.

4.2.2.2 VALIDATION

The validation of the generated code may seem to be quite easy for one who disposes of an example target file; as the generated file just has to be the same as this target file. For practical purposes, this method may not be best suited. In fact the goal is to have every file valid as long as the model is valid. This question arose essentially for the xml files as they are the most complete and the most complicated of the three types of files. Full validation has not been accomplished yet and several methods are to be explored. The reason why a full validation was not possible is that a precise list of all constraints for the xml files does not exist. CAIM provides us with error detection when xml files are imported. So a first validation can be done by using all functionality available in a model, generating its files and seeing if no errors occur when importing them.

4.2.3 RESULTS

The java code skeletons are really easy to generate and validate as the methods to be implemented are already known depending on the object class. As the generation module does not include a lot of programmatic parts, this has been easy to create and to validate.

For the xml files:

The two main files are EnvironmentSettings and EnvironmentObjects, and for the EnvironmentObjects? we aim to generate only the Screens and Tasks parts.

For the first one only one module is used with 4 templates:

- One for generating the file, that calls loops in order to generate code for each LAH and each BLTH, called over the model root.
- One for generating code for a LAH called over a LAH.
- One for generating code for a BLTH called over a BLTH
- One for replacing non-standard characters by HMTL within the objects description

For the second file two modules are used with many templates:

- The first module includes :
    - A template for generating the files that calls loops over tasks and screens.
    - A template for Task code that calls loops over Tabs and BLTHs included in a Task
    - A template for Tabs
    - A template for BLTHs
    - A template for Screens that calls a loop over Screen Fields
    - A template for Search Screens that call loops over Search Fields and Result Fields
    - A template for Search Fields
    - A template for Result Fields
    - Two templates for Search Screens filters
    - Several utility templates
- The second module includes :
    - A template for each kind of Screen Field and Screen Field Separator
    - A template that redirect to the appropriate template depending of the Field kind

4.3 Documentation Generation

4.3.1 Theory (see 4.3.1)

4.3.2 Methods

The most appropriate style for displaying clear information on the model was tables that show properties and relations of the different objects. Obviously it was also easier to generate tables than written text. We started our work from an existing technical specification from a project at the Company. The MDA approach allows accessing easily the links between objects and thus is particularly suited for generating tables of dependency, allowing impact analysis.

Then a choice had to be made of the file format to use. The criteria for this choice were:

1. Allow generation of complex tables
2. Quality of page layout
3. Not too hard to generate
4. Easily editable
5. Respect standard of the company

The usual format used by the employees for this kind of document is .doc so that they can edit it and then they convert it to .pdf, before they send the final version to the customer. Both these formats are really complex and not appropriated for automatic generation. We settled on the HMTL format, which, when properly designed, may be converted into a .doc document with Microsoft Word. Also complex tables are quite easy to code with HTML and the layout may be customized with CSS.

Another way that has been suggested to us by the Obeo Company was to use a software called “gendoc” in order to include generated text directly into a word document. It however has several problem of compatibility and the new release was not available at the time of the project.

4.3.3 RESULTS

The generation of the documentation gives good results. Combined with a retro model from an existing project, it allowed generating important pieces of documentation. Even if some style effects proved to be quite hard to accomplish with HTML (especially vertical column headers for some tables) the page layout looks good and is easy to modify. The model approach, allows generating all kinds of table that were needed without too much complexity.

The CSS file has been designed beforehand and is common to all projects. To allow more flexibility it could have been generated depending on the model. That would have permitted e.g. to decide precisely the columns widths depending on the lengths of the titles. HTML however already does this quite well, so it was not critical for the project.

The documentation is organized in seven sections, each of which contains several tables.

1. Tasks Tables
2. Screen Tables
3. Search Screen Tables
4. Logical Attribute Handlers Tables
5. Business Logic Task Handlers Tables
6. Event Listener Tables
7. One single table of impact between Tasks and Physical Attributes

In the generation module there is one template for generating the file, called over the root of the model, then one template for each kind of table, called on the relevant objects, and finally several utility templates for formatting or allowing generation into several languages.

4.4 RETRO MODELING

4.4.1 THEORY

Retro-modeling is the action of building the model from an existing project, as opposed to building the project from a model designed beforehand. Both these actions are parts of Model Driven Development (MDD).

Retro Modeling may be used for creating analysis views and documentation of an existing project, but its real usefulness lies in migrating a project to another platform or to another version of the same platform.

4.4.2 METHODS

The implementation of a retro modeler depends a lot on its input and output, i.e. the format of the input files and the format in which the models are stored. Whatever the formats of the files given as input to the retro modeler, a parser will probably have to be developed for them. For Ecore, the instantiation of the model may be done by using the edit methods provided by EMF and instantiating the object with Java code, or by writing the model directly in xml.

In our case, the retro modeling is done from three .xml files and one label properties file. As the target model xml files are quite close to the source xml files, the choice was made to directly write the xml file of the model.

The parsing is done in Java using the library JDOM [I] which is based on the SAX API (Simple API for XML). The .properties file is used by only one method that gets the appropriate label if it exists in the file.

A good practice after the retro modeling process is to make sure that the instantiated model is valid, by running a validation step. Especially concerning the attributes that take enumerated values, because giving them a value which is not present in the meta-model enumeration would cause an error at the loading of the model in the MDA tool.

The main difference between the source xml files and the model are the relationships between objects. In the source xml, a pointer is only represented by the name of the target object, without any clue about its existence in the file. In the model, a pointer points to the “address” of the target object, which has to exist at this address and to belong to the right class.

For this matter, a solution is to make two iterations over the objects: one for instantiation, and one for creating relations. A third one would be for validation. Considering the size of the source file (may include more than 40000 lines), optimizing this process was a matter of importance. In this project it was possible to create most of the relations during the first reading of the file, just by instantiating the objects in a well-thought order. Then there are only a few relationships left to implement, which does not require reading the whole file again. This makes the code less complexand allows a faster execution. It can be listed as a good practice.

The combination of retro modeling and code generation allows a simple first validation of the process by comparingan original file with its parsed-then-generated homolog.

4.4.3 RESULTS

The retro modeling has been tested on large scale projects and allows instantiating models that would need months of work to be made by hand, in only a few seconds.

4.5 VIEWPOINT DEFINITION

4.5.1 METHODS

4.5.1.1 PRINCIPLE

The viewpoints are supported by the meta-model for defining several views of the same model data. As represented in the figure 12, the point of view is like the projection of a system under a particular angle.

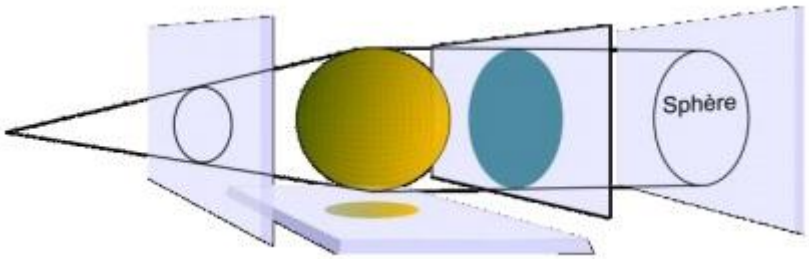

FIGURE 12 : POINT OF VIEW APPLIED TO A SPHERE REPRESENTING THE SYSTEM

As defined in the previous part, three points of view have been considered for this project. The actors for who they are designed are Product Manager, Integrator and Architect. What is most representative of a view point is a concern, in facts a view point is not limited to one kind of representation. For each view point (or point of view) several kinds of representations have been tested. This allows drawing a map of the diagrams, such as the one present for UML diagrams in [13]. The figure 13 includes only the diagram representations, the table representations have been omitted on purpose for not overloading the figure.

4.5.1.2 MAP OF DIAGRAMS

All these diagrams (Figure 13) are defined in the same .Odesign style sheet. The representation of each object can be precisely controlled for each of them.

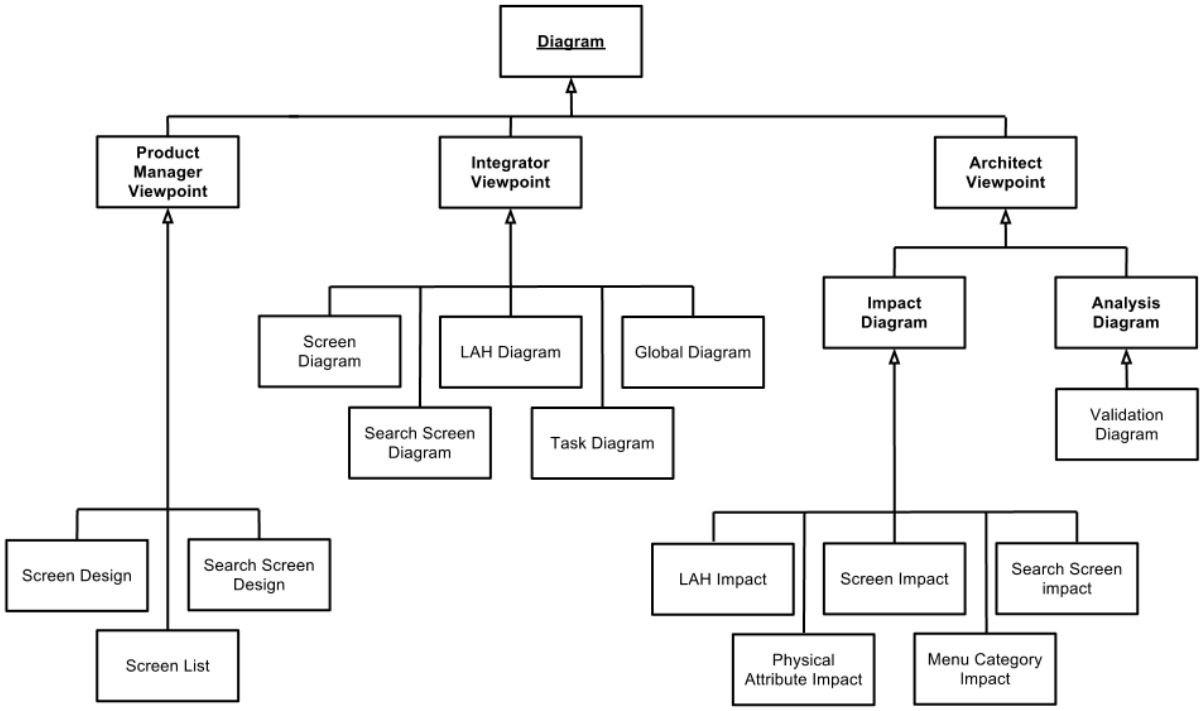

FIGURE 13 : MAP OF DIAGRAMS DEFINED WITHIN THE PROJECT

4.5.2 RESULTS

4.5.2.1 PRODUCT MANAGER POINT OF VIEW

The first point of view defined is the Product Manager's one. Indeed it is the first to be used in the target integration process involving MDA. It is also the point from which the efficiency gain is expected (compared to the traditional integration process using CA native configuration tool).

The Product Manager (PM)'s point of view is composed of three diagrams. The first two are very similar and allow designing Screens and Search Screens. The third one is of less importance and allows navigation by listing the screens already created. From now focus will be given on the Screen Design Interface as it is the most important and representative of this point of view.

The entire trick was to make a screen design interface out of a graphical model editor. This has been accomplished to some extent and the result will be discussed in the next part. The kind of representation used is a diagram, representing the screen as a container. A tool bar on the right allows adding field to the screen. This diagram uses all the most advanced tools provided by Obeo Designer in order to improve usability. In factthe Product Manager is not someone who is familiar with eclipse, sothe Eclipse Environment had to be hidden as far as possible. A possibility which has not been studied in the scope of this project was to include this interface in a standalone application hiding even more eclipse details. In the same interest of hiding complexity, the only modification possible on objects is changing their names and also the permission for fields (R/RW …). The name of objects is editable by directly typing it after a double click on the label. The name and permission are also editable through a wizard which can be opened by double clicking on a field.

In the diagram, the container Screen is given a modified background from a screen shot of the interface. This screen shot may be adapted to the graphic chart of the customer. Fields have been sorted by widgets, even though they are represented by a single object in the meta model. So the interface allows placing fields in the screen. The widgets available do not represent the full range of possibilities of CA IM but others could be easily added. Available widgets are: Text; Message; Calendar; Checkbox; Scroll List. The scroll list field is also a container to which options may be added. The main weakness of this interface is the positioning of the fields.

Indeed Obeo does not permit any mechanisms for setting an object's position. Also it is not possible to define other constraints for positioning an object than "an object in its container". Developing an interface programmatically would be possible but it would take a long time and was not in the scope of this project.

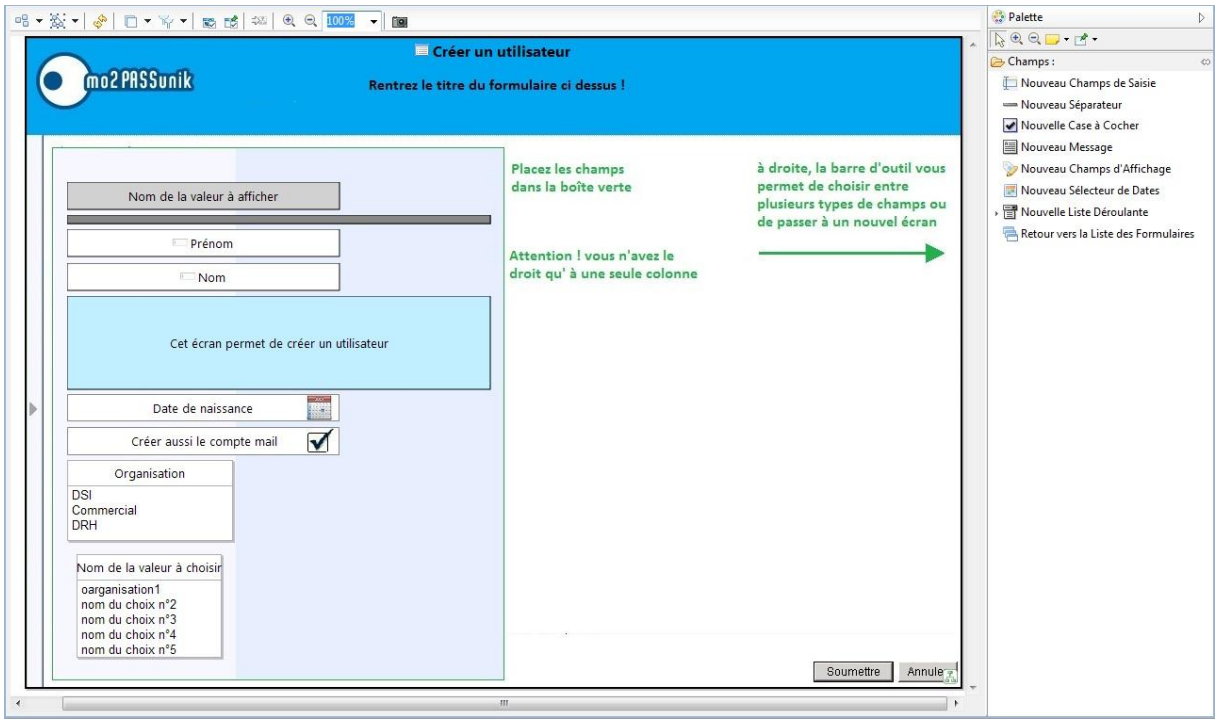

FIGURE 14 : PRODUCT MANAGER'S VIEWPOINT: SCREEN DESIGN

4.5.2.2 INTEGRATOR'S POINT OF VIEW

The Integrator's point of view allows creating and modifying all the objects defined in the metamodel. This is possible through several kinds of diagram and table. These many diagrams are necessary because the model may be really important, so that a global diagram is not useable in most cases. It is possible to navigate easily from one diagram to another so that a feeling that there is only one model is preserved. For the same reasons, each kind of diagram allows the use of layers, in order for the integrator to be able to display only the objects he is interested in for each diagram. On every diagram it is possible to create and link objects to each other by using a toolbar. This toolbar and the action of each tool are also defined in the .Odesign file.

These diagrams are in the style of a UML 2.0 Object diagram. Each kind of object is represented with a different color. This color chart is kept the same for every diagram. Unlike for the uml object diagram, the attributes of each object are not displayed directly inside boxes. It it possible to edit the bject. The name of the object can be set by editing the box name. For the other attributes it is possible either to edit with Eclipse Property View or with a wizard opened on double click. These wizards and properties views may be customized with a plug-in called EEF (Extended Editing Framework). This plug-in is also developed by Obeo, but it is a common open source plug-in for eclipse.

For the integrator's point of view, there are in all five kinds of diagrams and three kinds of tables. In the following figure – figure 14 - a linear vision of the integration process is presented. It is the vision from an Integrator's point a view. The number and scope of integrator's diagrams have been based on this process. The navigation is possible thanks to the object common to several diagrams, such as screens, search screens and logical attribute handlers. The tables defined concern Screens; one is for managing screen fields; the second for managing searchscreen fields; the third one for managing search filters for search screens. By giving many ways for editing the same thing, it allows the integrator to choose his favorite.

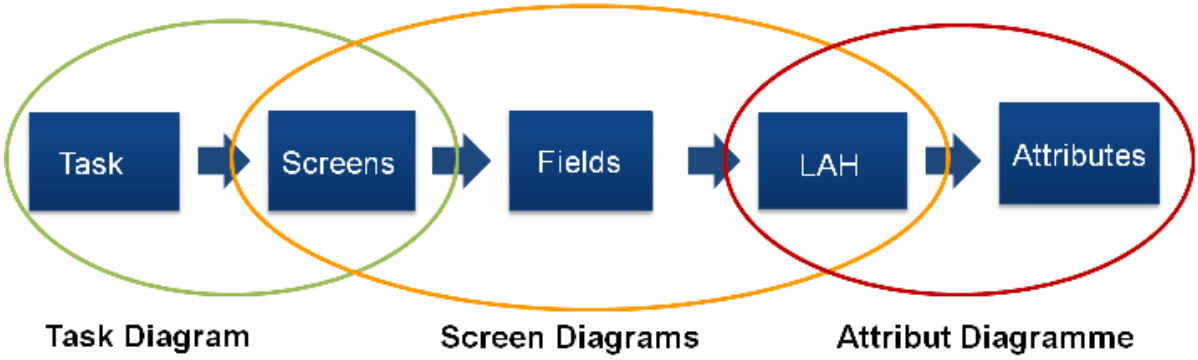

FIGURE 15 : LINEAR VISION OF AN INTEGRATOR

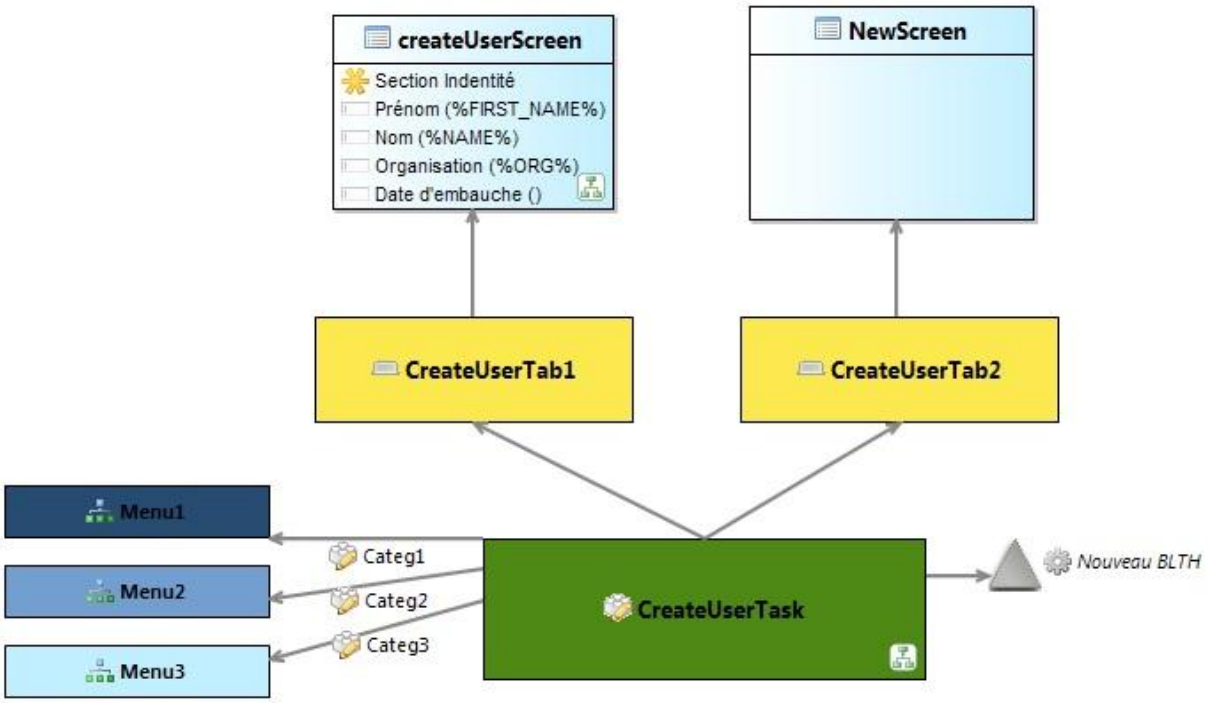

FIGURE 16 : INTEGRATOR'S VIEWPOINT: TASK DESIGN

4.5.2.3 ARCHITECT POINT OF VIEW

The Architect's point of view has been designed with two main purposes in mind. The first to provide help for validating the work done by the integrator and ensuring that the MDA tool has been well used. The second is to allow maintenance, especially allowing analysis of a project. Only the second has been developed in this project, but the analysis diagrams may be used for validation as well.

Analysis diagrams have been discussed with integrators and software architects, and they pointed out that one of the most interesting things would be to know the impact of a modification of a particular object. For instance "If I modify this Screen, which Tasks will then be impacted?" The answer to this question will determine which tests must be performed after the modification of the target object. As this feature is not present in the native configuration tool from CA, it was interesting to try it. In fact, the data in the XML configuration file of the product references label of objects it points to. This reference does thus not contain any information about its location or even if it exists. In the model a reference is an address; and the target object of the expected type must exist at the indicated address, in order for the model to be valid. This data

structure is particularly adapted for tracing all the objects used by other objects. As OCL provides methods for using inverse references it is rather easy to see the impact of a modification on the model.

One kind of diagram has been designed for each object of the meta-model which is referenced by some others. These objects are:

- Physical Attributes, referenced by LAHs and ScreenFields, SearchFields and ResultFields (kind of Field in a Search Screen)
- Logical attribut handler (LAH), referenced by ScreenFields and ResultFields
- Screen, referenced by Tabs
- SearchScreen referenced by Tasks and ScreenFields
- Category, referenced by Tasks

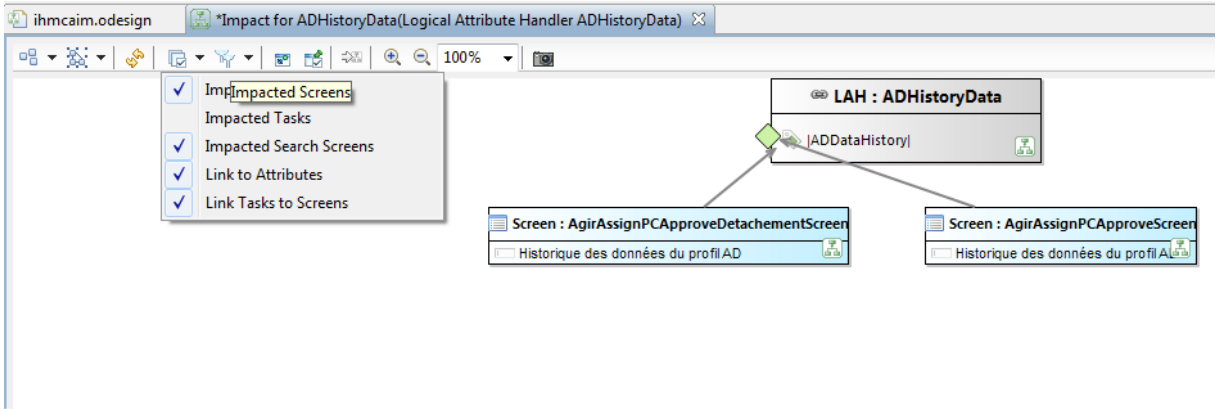

FIGURE 17 : ARCHITECT'S VIEWPOINT : LAH IMPACT

Each of the impact diagrams has several displayable layers for showing different types of objects. Also the impact diagram of an object may display several degrees of references. For instance, on a LAHs impact diagram, it is possible to display : 1 Screens with ScreenFields using the target LAH ;2 SearchScreens with ResultFields using the target LAH ;3 Task and Tabs using those Screens ; 4 The link between all these objects. Each layer may be displayed or hidden.

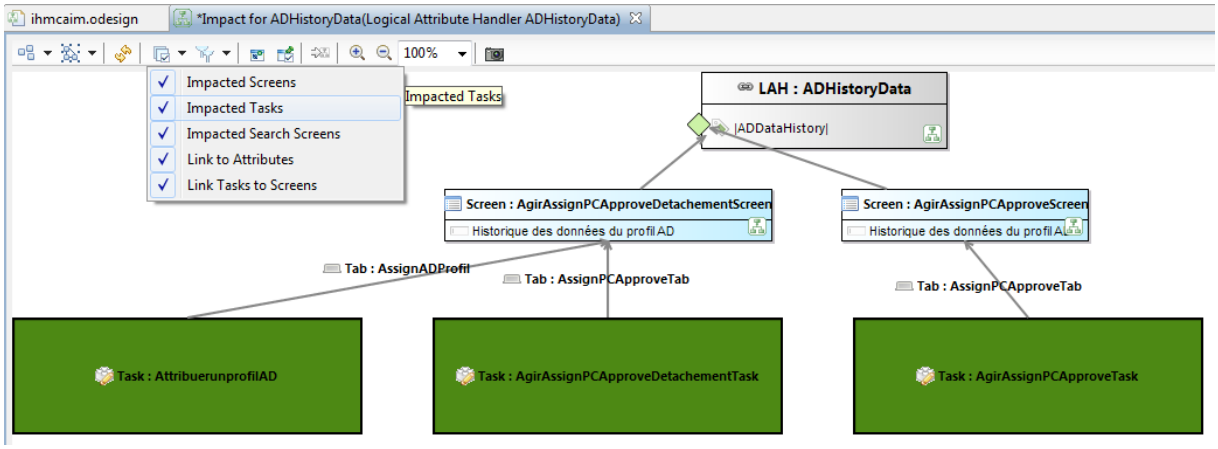

FIGURE 18 : ARCHITECT'S VIEWPOINT : LAH IMPACT (WITH TASKS)

Most of the diagrams also offer tools to perform the modification on an object, once knowing the impact, such as in the integrator's diagrams does. These diagrams proved very useful. Their main weakness is that they may grow too big in some cases. Indeed although layers by object types can be hiddenfor some important objects a

single layer may include more than 200 objects. This is too much to be displayed in a single diagram, and a table would be more appropriate.

5 DISCUSSION

The objective? of this project is to improve the integration process of the GUIs in CA IM. For this a collaborative process has been designed. This process involves MDA methods implemented in the tool Obeo Designer. There were two points in this process where improvements were expected. The first was by constraining the GUIs design where PowerPoint leaves a total freedom. The Second was by reusing in an automated way the data from this GUIs designs.

To set up this process a meta model has been developed. This meta-model allowed defining several viewpoints for the instance-models. Each of these view points includes several kinds of diagrams and tables. The link between these models and the product CA IM is established in both directions by generators and retro-modeling modules. Documentation generators have also been implemented.

5.1 ANALYSIS OF RESULTS

5.1.1 PROPOSED SOLUTION

5.1.1.1 CONSTRAIN THE DESIGN OF THE GUIS IN THE FUNCTIONAL SPECIFICATIONS

A point where improvements where expected in the integration process was for the expression of functional needs. The diagrams developed for the Product Manager allow defining a screen only by placing fields of different widget types. The main weakness of this interface is not to put constraints on the positioning of fields. Its main advantages are to limit the objects one can use for designing a screen and to give a first impression of what the screen will really look like after integration.

The positioning is not available because it was not possible to handle without hard coding it. Indeed the bet taken with this approach was that it was possible to make a GUI design interface out of a model Editor, and Obeo Designer which in this case is the editor of the model editor does not support the handling of position in the editors defined. Even so, the result is quite convincing. The advantag of having a customer know the result in advance is that it makes the process go faster. This is otherwise not often the case in many computer science industrial projects.

5.1.1.2 REUSE THE DATA FROM GUIS FOR INTEGRATION

The Second point that was expected to improve efficiency was the use of GUIdesign data directly for integration. The integration process with the tools defined allows it. Indeed the objects created by the Product Manager are Screen and Screen Fields instantiated from the meta model. The Integrator may then reuse them for integrating the software; the data may also be directly generated and injected in the product for further developments. The weakness of this approach also concerns positioning. Indeed the position and order of fields on the screen are not part of the model data. Fields must then be manually organised by the integrator.

5.1.1.3 EXPLORE OTHER OPPORTUNITIES FROM MDA

Besides forward engineering, the reverse process has also been tested. It allows building the model automatically from an existing project. Combined with impact analysis views and documentation generator modules, it brings a completely automated analysis process for projects at the company.

5.1.2 PROBLEMS ENCOUNTERED

5.1.2.1 LACK OF SUITABLE DOCUMENTATION

First the lack of documentation concerns the part of the software to model. Doing a meta model for configuring a product (CA IM) demands an important knowledge on the product. CA IM GUIs are a complex matter. There is an important documentation for how to configure them, but the product was not developed for being configured another way. For this reason there is no important document explaining the links between the

object data in XML files and the configuration of the software. Even if most of the data was explicit enough, this activity required such a level of detail in the configuration process that some aspects of the meta model were completely uncertain. Even expert integrators could not be sure of their answers for these aspects.

Another area where more documentation would have been desirable concerns the modeling tools. For Obeo Designer, the documentation becomes quite vague when tackling complex matters. Several plugins for eclipse had nearly no documentation available, such as EEF for adding nice properties views and gendoc2 for inserting generated documentation in .doc documents. For these two open source plugins, the lack of documentation was such that the only help available in order to use them was to ask their developers directly. Luckily both Obeo and gendoc2 development teams are very helpful to answer questions.

5.1.2.2 OTHER PROBLEMS ENCOUNTERED

As evoked before, one of the challenge in this project was to make a screen design interface from a graphical model editor. Here appeared the limitation of the tool used as Obeo Designer is not made for that purpose. The Screen Design interface is interesting but the lack of a handling of the position data was a limitation in the project.

Also the subject would benefit from being treated by an expert of the CA IM product. As defined in the target process, the architect who designed the MDA tools must have expert knowledge on the target product and on MDA methods.

Another problem which is related to the previous one is the maintenance of the MDA tools. Any modification done on the meta model implies an update of the view-model, the generators modules and the retro modeling module. There are many ways of modeling the same thing and some may be more appropriated than others at some time. This principle applies for the meta model and may bring much complexity in the maintenance of the tools.

The last problem identified in this project is the unsolved question of the location of constraints. For instance, a task with property “object = User” can only use screen where “object=user”. This constraint may be represented in many ways. The first and the strongest would be to separate classes in the meta model: make class task abstract and implement a UserTask; do the same for the class Screen; create link only between implementation. This method is inappropriate in most cases, because it brings complexity to the meta-model, because it is not appropriate for all kind of constraints and finally because it makes the treatment of the instance-models much more complex. A second way may be to include OCL in the meta model. This way may look the best but it is only understood as a validation rule. A Task will still be possibly connected to any kind of Screen, but a validation step would notify it. This way, when editing the model, there is still the choice between all Screens of all kinds when trying to modify a Task. As Obeo Designer proposes to include validation tools on diagram, the same work that OCL does could be done with this framework. This would be a third way, quite similar to the second but allowing the use of constraints without having to move out of a point of view. This could be useful for the Product Manager who is supposed to fear the Eclipse environment. A fourth and last way may be to place this constraint awhile editing through a point of view, since Obeo Designer allows the use of constraints during graphic edition (but not during edition with forms).

The best way would be to use several levels of validation. The only way that allows avoiding a screen to be used by a task with a different object is the first one. It’s why the first way was said to be the strongest. As this way has also many disadvantages, solutions to this problem had to be found on a case to case basis, and sometime no solution was found.

5.1.3 COMPARISON WITH LITERATURE

It is interesting to see which of the three kind of MDE mentioned before have actually delivered results. The integration process defined in the objectives was clearly of the forward MDE type. Despite of that, retro MDE

has finally been successfully performed through a retro-modeling module. This allows working alternatively on the model or on the product. In this, obtained result is close from the runtime MDE, because it is possible to construct a project successively with the MDA tools and with the native configuration tools. The only requirement is to generate and import in order to bring what has been done with models on the configuration tool in one way. The other way around it is an export and the use of retro-modeling module on the xml file that allows going from the native tool to the MDA tool. This use allows monitoring a project and correcting any mistakes by the use of a model view which is easier to comprehend and may show thing from different angles.

Obeo designer and EMF have been challenged through this project. It will now be discussed if this recent tools were answering their promises and if they were appropriated for this project. The overall impression is quite good. Indeed they allowed building an MDA process with all the necessary bricks. The Product Manager's point of view and particularly the screen design interface could nevertheless not be implemented as far as wished. Obeo Designer was indeed not intended for such a use and did remarkably well in the end. The definition of viewpoints through the .Odesign style sheet is really easy and does not require a lot of skill in programming. It may take some time to become familiar with it, but once the first steps are overcome,designing new diagram types can be done quite fast.

5.2 PERSPECTIVES FOR FUNCTIONAL DESIGN WITH MODELS

5.2.1 DEVELOP AN INTERFACE USING GEF

In order to improve the Product Manager's point of view, it could be possible to hardcode it. In facts Obeo Designer allow building model editors without much coding and is thus limited. Using the open source frameworks behind Obeo Designer, it could be possible to implement your own model editor. These frameworks are called GEF that stands for Graphical Editing Framework, and GMF, Graphical Modeling Framework. According to the Obeo Designer team, this framework is complex to use and it would involve a lot of programming, which is why it has not been tried in this project.

It could still be really interesting to further with the MDA approach. One of the main challenges was to make a screen design interface out of a graphical model editor. Indeed the constraints on the GUIs in the target product lets us imagine such a thing. By hard coding it, it would allow getting more information in the model, and also the interface really looking like the final result, in order the customer to know in advance the result. This would avoid many round-trips in the design of the CA IM GUIs.

Particularly some improvements could be:

- Handle the position and order of fields in a column or on a grid
- Handle several columns
- Handle the real menu and tabs
- Make the screen design dynamic by allowing clicking on a tab in order to design another screen for the same task
- Provide a more realistic result by using the real skins of the product for the different kinds of fields

5.2.2 TO MAKE USE OF MORE OF MDA CAPABILITIES

One important aspect of MDA which has not been used here are the concepts of platform and model transformation. The MDA proposes the concept of platform dependent and platform independent models and meta models. Such architectures could be of a great use at the company. It is nevertheless complex to set up and operate. As explained in the literature part, a platform independent meta model is built in order to represent a concept independently of the platform used. In facts there are at Arismore several platforms that are used in the Identity Management domain. These are other identity management softwares such as Oracle IM, and also other versions of CA IM (R8 and R12). In this case, three Platform Dependent Meta models (PDMM) have to be considered. One for CAIM R8, one for CAIM R12, and one for OIM. All of these PDMMs

would correspond to the same functions. It would allow having an IM platform independent meta-model that represents the concepts of these three PDMMs. Having a runtime environment with transformations between the PIMM and each of the PDMMs could be of a great use. Indeed it could then be easy for the customer to decide on which software he wants to use after the functional specification. It would also be easy by the use of several transformations to migrate a system from a platform to another. (i.e.: another software or a newer version). This kind of system is the Rolls Royce of MDA and could look like brilliant. It is nevertheless really hard to build and to operate. There is no guarantee than it would allow as much precision in the expression of functional need as a simple PDMM. Also in such a complex system any modification in one of the meta models implies changes everywhere. Thus this approach is interesting to evoke but should be considered with much care.

5.2.3 Construct a Functional Platform Dependent Meta model

Another suggestion for improving the current system could be to build a functional meta model. MDA proposes an architecture with a Plateform Independent Model (PIM) and a Plateform Dependent Model (PDM) for going from functional to technical matters. In my opinion, two PDMMs of which, one is functional, and the other one technical, may be more efficient. In fact a PDIMM might be too far from the subject, and reduce the precision in the expression of functional needs. Even if the use of several meta models is possible such as a chain of PDIMM more and more concrete followed by a chain of PDMM, such a chain would imply many transformation to deal with and much complexity for operation.

The meta model tested in this project has been built in order to generate and be exhaustive for the data. Building a simpler meta model, adapted to the functional use might be a solution to consider. This Meta model has been thought for a new project at Arismore, and some ideas have already been written on paper. It could use a new class of attributes called functional attributes. This attribute would point to a logical or physical attribute. As an intermediate between a field and an attribute, it could be designed to be useable by the product Manager. Another feature would be to separate tasks by their action such as to create different classes for Creation Tasks and Modification Tasks. In practice, most of the tasks are of one of these two kinds, but there is in facts 10 types. For a Product Manager, this could be sufficient to work with. In this respect the scope of the Product Manager could be enlarged.

Data from this functional platform dependent model would still allow much precision, and could then be transposed to the meta model developed for this project. This approach looks like a good compromise between what has already been done and a full and complex MDA architecture, involving many meta models and transformations.

5.2.4 Extend the scope

In this project, the scope has been set on GUIs of CA IM. CA IM includes other objects that may be modeled. There are particularly the task approval workflows, the organizations hierarchy and the access roles which are part of Identity Management. These objects would surely be easy to model and design through a view in Obeo Designer. Indeed workflows look like an UML state diagram, and in CA IM the tool that allows designing them uses a graphical model editor. Organization and roles could be easily represented as a tree structure. Extend the scope to all object in IM could maybe even avoid having to use the native tool at all, for more consistency.

6 CONCLUSION

The company Arismore faced the problem of complexity in the definition of functional needs. In order to bring improvement on that matter, it was decided to explore the opportunities of model driven architecture. The scope of this study has been set on the definition of GUIs for identity management software. Indeed these interfaces have enough constraints in their configuration to allow representing them through a model. The tool Obeo Designer was chosen beforhand for this study.

The question was to know how much and in which way MDA could improve the integration process. For this purpose a specific integration process had been designed. There were two points where improvements where expected. The first one was by simplifying and constraining the expression of needs by a customer through the design of interfaces with a graphical model editor. The second one was by reusing the data from this editor directly for the integration of the software.

In order to solve the problem, a meta model has been designed within the scope of the project. This meta model allowed defining through the tools some code generators in order to set up a forward MDA process. From the experience of building this meta-model, several patterns have been identified for further uses by the company. The model editing has been designed with separate views for each actor. The integrator's view permits to see the whole complexity while the product manager's point of view allows defining the interface through a WYSIWYG interface. In order to explore what other opportunities could come from MDA, retro modelling has been tried. This approach allowed defining impact analysis views and technical documentation generation modules for the existing projects.

Through this study an evaluation of the tools and the methods could bemade. One of the main challenges was to create a GUI design interface out of a graphical model editor. On this point the objectives have been achieved, but the tool does not allow having precise information on the order or positioning of the fields on a form. This doesn't make the interface usable by a customer as it is, but suggests some ways to improve it. The results showed that it was also possible to reuse the information from a point a view to another, in order to provide continuity in the integration process. The reverse engineering methods proved to be quite useful for existing projects and are also a good way to validate that the meta model includes all the information.

For further uses at Arismore, it has been suggested to consider building a functional specific meta model for better handling of constraints. Another approach to explore may be to develop a graphical model editor programmatically, in order it to include the positioning constraints for the fields in the screens.

On the whole, MDA kept its promises to reduce the gap between functional need and technical realization. It is still a complex process, with set-up and operation costs. Its main interest lies in industrializable developments where it may bring many improvements, such as an easier expression of the need, and a better control on the code and its structure.

7 GLOSSARY

7.1 MODEL DRIVEN RELATED TERMS

7.1.1 MDA (OMG): MODEL DRIVEN ARCHITECTURE

EN: **Model-driven architecture** (MDA) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems. It was launched by the Object Management Group (OMG) in 2001. (Wikipedia)

7.1.2 MDD - MDE: MODEL DRIVEN DEVELOPMENT/ ENGINEERING

EN: **Model-driven engineering (MDE)** is a software development methodology which focuses on creating and exploiting domain models, rather than on the computing concepts. (Wikipedia)

7.1.3 MOF (OMG): META OBJECT FACILITY

EN: The **Meta-Object Facility** (MOF) is an Object Management Group (OMG) standard for model-driven engineering. (Wikipedia) MOF's model is its own metamodel (it is self-defined).

7.1.4 OMG: OBJECT MANAGEMENT GROUP

EN**: Object Management Group** (OMG) is a consortium, originally aimed at setting standards for distributed object-oriented systems, and is now focused on modeling (programs, systems and business processes) and model-based standards. (Wikipedia)

7.1.5 TEMPLATE (MOF)

EN: A **template** specifies a text template with placeholders for data to be extracted from models. These placeholders are expressions specified in terms of metamodel entities and are evaluated over instances of these metamodel entities. Template is a specialization of block. A template can have a guard (inherited from block) that specifies when it can be invoked. A template can override one or more other templates. An overriding template should have the same number of parameters as the overridden template with compatible types. The overriding template is invoked in place of the overridden template when the parameter types match and the guard condition of the overriding template evaluates to true. (MOFM2T Specifications)

7.1.6 QUERY (MOF)

EN: A **query** is a side-effect-free operation. It is owned by a module. A query is required to produce the same result each time it is invoked with the same arguments. A query is specified by an OCL expression. (MOFM2T Specifications)

7.1.7 METAMODEL

EN: A **meta-model** typically defines the language and processes from which to form a model. (Wikipedia)

7.1.8 VIEW POINT (OBEO)

EN: Viewpoint is a tool which allows one to define easily several representations (diagrams, tables , treeviews, …) of the same model. The representation descriptions are grouped by viewpoint. Main advantage of viewpoint is that it allows one to separate concerns. You should create viewpoint with this quite simply rule: one concern = one viewpoint (Viewpoint Developer Guide)

8 REFERENCES

8.1 PAPERS

[1] France, R. & Rumpe, B. (2007). Model-driven Development of Complex Software : A Research Roadmap.
[2] Liddle, S. W. (2010). Model-Driven Software Development.
[3] OMG(2003). MDA Guide Version 1.0.1
[4] (2003).Metamodelling for MDA
[5] Bézivin, J., Gérard, G., Muller, P.-A. & Rioux, L. (2002). MDA components : Challenges and Opportunities.
[6] Jouault, F., Bézivin, J. & Barbero, M. (2009). Towards an Advanced Model Driven Engineering Toolbox.
[7] Bézivin, J. & Gerbé, O. (2001). Towards a precise definition of OMG/MDA Framework.
[8] Brown, A.W. (2004). Model driven architecture: Principles and practice. Software and System Modeling, 3, 314-327.
[9] Hailpern, B. & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly.
[10] Promesses et Interrogations de l'Approche MDA, Jean Bezivin & Xavier Blanc, 2002, France
[11] MOF Model to Text Transformation Language (MOFM2T), 1.0, January2008
[12] Juliot, E. & Benois, J. (2009). Viewpoints creation using Obeo Designer or how to build Eclipse DSM without being an expert developer?
[13] OMG. (2005). Unified Modeling Language 2.0, Superstructure
http://www.omg.org/spec/MOFM2T/1.0/

8.2 WEB (AS OF JANUARY 2012)

[A] Arismore http://www.arismore.fr/?page_id=301
[B] The Open Group Architecture http://www.opengroup.org/architecture/
[C] Architecture Forum France http://www.architecture-forum.org/
[D] OMG http://www.omg.org/
[E] OMG mda http://www.omg.org/mda/
[F] OMG MOF http://www.omg.org/mof/
[G] CA IM website http://www.ca.com/us/user-provisioning.aspx
[H] Acceleo Website http://eclipse.org/acceleo/
[I] JDOM Library http://jdom.org/
[J] Waters, J.K., *The ABCs of Identity Management*
http://www.csoonline.com/article/205053/the-abcs-of-identity-management
[K] 6th international workshop models@run.time (october 2011)
http://www.comp.lancs.ac.uk/~bencomo/MRT11/

8.3 SOFTWARE DOCUMENTATION

[I] Oracle Identity Manager 11g *Essentials, Student Guide,* Volume 1.
[II] Obeo Designer Architect Edition Documentation.
[III] Eclipse Modeling Framework Documentation.
[IV] Acceleo Model to Text Transformation Language Documentation.
[V] Computer Associate Identity Manager r12.5 Developer's and User's Guides.
[VI] Computer Associate Identity Manager r12.5 Student Guide.

TRITA-CSC-E 2012:098
ISRN-KTH/CSC/E--12/098-SE
ISSN-1653-5715

www.kth.se