UNIVERSIDAD DE EXTREMADURA

PHD THESIS:

ARQUITECTURAS MULTICAPA: ACERCANDO EL DISEÑO A LA IMPLEMENTACIÓN

JOSE GARCIA-ALONSO

DEPARTMENT OF COMPUTER AND TELEMATIC SYSTEMS ENGINEERING

Conformity of the Supervisor:

Signed: D. Juan Manuel Murillo Rodríguez
Associate Professor
Department of Computer and Telematic Systems Engineering
University of Extremadura

2014

To Isabel, for her unconditional support.

To Adrián, who makes it all worth it.

Acknowledgement

The journey to become PhD is not an easy one. Fortunately, I can say that I have enjoyed every little step of it, mostly thanks to the people who have accompanied me along it. I hope these paragraphs serve to show my gratitude to everyone who has supported me over this years.

I still remember my first years as a computer science student at the University of Extremadura. I already loved computers and software back then, but I was fortunate to meet a fantastic group of teachers, passionate about their work, that started to teach me the wonders of software engineering. These teacher had something in common, all of them belonged to the Quercus Software Engineering Group. In these early years I decided that, if the opportunity arose, I would become a member of the Quercus group. After being a member for seven years, I have not regretted that decision and I can only show my gratitude to all of the Quercus members for their continued support, and especially to its director Juan Hernandez.

One of the best experiences I have had over these years has been the creation of Gloin. What started as a crazy idea has become a reality that has allowed me to develop new facets of my profession, to live a multitude of new experiences and, especially, to meet and work with a lot of wonderful people. Thanks guys for sharing part of your lives with me.

Usually, the development of a thesis is a lonely job. However, in my case I have been fortunate to have someone that have shared this journey with me from the beginning. If anyone knows all aspects of this journey, all the ups and downs, that's Javier Berrocal. Without his support and company it would have been impossible to get here.

I have no words to describe the gratitude to my supervisor Juan Manuel Murillo. We have been working together for almost ten years and in all this time I have not stopped learning from him. Working with him, not only allowed me to get here, but has taught me a more positive way of coping with challenges and difficulties and has made me a better professional and a better person. Thank you.

Finally, none of this would have been possible without my family. My mother, who taught me to always have my feet on the ground and instilled in me the responsibility needed to finish a PhD. My father, who taught me to have my mind on the stars and gave me the hunger for knowledge and the love for science needed to start a PhD. Isabel, who has understood and encouraged me in every decision, even in the most difficult for her, and has suffered all the negative aspects of this journey without complaining. Finally Adrian, who has suffered my constant absences, but has also given me the strength I needed to get here Infinite thanks.

Abstract

In the last years Extremadura has become a privileged location for software development companies. The low cost of living and the abundance of qualified workers have led several of the country's leading development companies to create important distributed development centers in the region. Several collaborations made with these centers during the last years have allowed the authors of this thesis to work on the common problems they face. Specifically, this thesis started with a collaboration project with the regional government of Extremadura and the company Indra Software Labs.

The main goal of this project was to solve the problems affecting the regional government related with the architectural and technological variability that can be found in multi-layer applications. These problems were also aggravated by the high staff rotation suffered by the regional government. Companies like Indra, with distributed development center, shared these same problems.

After analyzing the causes of these problems, related with the ever increasing complexity of software architectures and the fast pace of evolution of the development technologies, different approaches were studied to try to solve them. None of these proposals solved all the problems faced by the regional government, especially when taking into account the high staff rotation suffered by this organization. This lack of solutions led to the start of this thesis with the following goals:

- To identify and organize the most common architectural decisions in the development of framework based multi-layer applications.
- To simplify the use of design pattern and frameworks in the development of

such applications.

- To automatically generate an specific design of the applications tailored to the to the architectural decisions taken by the software architect starting from the requirements of the application.
- To automatically generate a significant amount of the application source code.

These goals have been achieved in this thesis by presenting ArchLayer, a development process designed to help software architects and developers of these companies. The proposed process allows software architects to convert an initial design of an application, completely agnostic to the architecture or technology with which it will be implemented, into a specific design for a given multi-layer architecture and a set of development frameworks. ArchLayer is supported by an architectural decisions repository that contains the architectural knowledge needed to develop this kind of applications. Additionally, a framework information meta-model is used to gather the technical knowledge needed to correctly use development framewoks. Finally, a set of model-to-model and model-to-text transformation is provided to help architects and developers during the development process.

To validate the proposed process, it has been used by a software development company in the development of two commercial projects. This validation has proved the feasibility, completeness and the effort required to apply the contributions presented in this thesis.

Resumen

En los últimos años Extremadura se ha convertido en un enclave privilegiado para las empresas de desarrollo de software. El bajo coste de la vida y la abundancia de personal cualificado han llevado a varias de las empresas de desarrollo más importantes del pais a instalar en la región centros de desarrollo distribuidos. Diversas colaboraciones realizadas con estos centros durante los ultimos años han permitido a los autores de esta tesis trabajar en los problemas habituales en este contexto. En concreto, esta tesis comenzó como un proyecto de colaboración entre el gobierno autonómico de Extremadura y la empresa Indra Software Labs.

El objetivo principal de este proyecto consistía en solucionar los problemas que afectaban al gobierno autonómico relacionados con la variabilidad arquitectónica y tecnológica presente en las aplicaciones multicapa. Estos problemas eran empeorados por la elevada rotación de personal sufrida por el gobierno regional. Empresas como Indra sufrían los mismos problemas en sus centros de desarrollo distribuidos.

Tras analizar las causas de estos problemas, relacionadas con la siempre creciente complejidad de las arquitecturas software y el elevado ritmo de evolución de las tecnologías de desarrollo, se estudiaron varias propuestas para tratar de resolverlos. Ninguna de estas propuestas permitían solventar todos los problemas abordadods, especialmente si se tenía en cuenta la elevada rotación de personal sufrida por dicha organización. Esta falta de soluciones llevó al inicio de esta tesis con los siguientes objetivos:

- Identificar y organizar las decisiones arquitectónicas más comunes para el desarrollo de aplicaciones multicapa basadas en frameworks.

- Simplificar el uso de patrones de diseño y frameworks de desarrollo en el desarrollo de este tipo de aplicaciones.
- Generar automáticamente un diseño especifico de la aplicación adaptado a las decisiones arquitectónicas tomadas por el arquitecto a partir de los requisitos de la aplicación.
- Generar automaticamente una parte significativa del código fuente de la aplicación.

Estos objetivos se cumplen en esta tesis con la presentación de ArchLayer, un proceso de desarrollo diseñado para ayudar a los arquitectos y desarrolladores de este tipo de compañías. El proceso propuesto permite convertir un diseño inical de una aplicación, completamente agnóstico de la arquitectura o tecnologías con la que se va a implementar, en un diseño especifico para una arquitectura concreta y un conjunto de frameworks de desarrollo. ArchLayer se basa en un repositorio de decisiones arquitectónicas que contiene el conocimiento arquitectónico necesario para desarrollar este tipo de aplicaciones. Adicionalmente, un metamodelo de información de los frameworks es utilizado para recopilar el conocimiento técnico necesario para utilizar correctamente los frameworks de desarrollo. Por último, se proporciona un conjunto de transformaciones modelo a modelo y modelo a texto para ayudar al arquitecto y a los desarrolladores durante el proceso de desarrollo.

Para validar el proceso propuesto, este ha sido utilizado por una compañía de desarrollo software en el desarrollo de dos proyectos comerciales. Esta validación ha permitido comprobar la viabilidad, la completitud y el esfuerzo requerido para aplicar las contribuciones presentadas en esta tesis.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

I don't care if I pass your test, I don't care if I follow your rules. If you can cheat, so can I. I won't let you beat me unfairly - I'll beat you unfairly first.

Ender's Game, Orson Scott Card.

The layer architectural pattern allows software architects to decompose a system into decoupled components called layers. Each layer provides services to the layer above and uses the services of the layer below. The use of this pattern benefits the modifiability, portability, and reusability of the final system (Avgeriou & Zdun, 2005). Therefore, multi-layer architectures are those in which the system has been decomposed into two or more decoupled components in a vertical manner.

This thesis focuses in the industrial development of systems using such architectures. Specifically, this work is centered in the gap between the initial design of a system and the detailed design that is needed in order for the system to be implemented. To explain this situation in detail, this introduction is organized as follows. Section 1.1 reports the origins of this thesis and the reasons why it was started. Section 1.2 presents the research context in which this work was conducted. Section 1.3 details the problems that this work aims to solve. Section 1.4 specifies the concrete objectives that have been addressed in this thesis. Section 1.5 provides a general vision of the proposed solution for the addressed problems. In section 1.6 the information related to the development of this work is stated. Finally, section

1.7 presents the structure of this thesis.

1.1 Thesis origins

The research work leading to the thesis presented here has a clear industrial motivation. This thesis originated from a contact between the regional government of Extremadura and the supervisor of this thesis.

The regional government was trying to modernize their development processes focusing on multi-layer architectures and Java development frameworks. However, they were facing significant problems.

First, the regional government has a high rate of staff rotation. Due to the type of contract made by the government, its staff has great mobility. This leads to its staff making numerous department changes and even city changes within the region every year. This staff rotation in itself does not pose a problem. However, when combined with a modernization process as the one being conducted by the regional government, it poses significant challenges. Mainly, the difficulty of training its staff in the new processes and technologies that were being implemented.

And second, most of the software used by the regional government is not developed by the government itself. Usually, the software is developed by software development companies hired by the regional government and the it only handles its operation and maintenance. Again, this in itself does not pose a problem. However, the government was realizing that each hired development was using a very different architecture and development technologies, complicating its maintenance.

This were the problems that led the regional government to seek help from the University of Extremadura, and specifically from the supervisor of this thesis due to his previous experience in framework based multi-layer applications and the management of distributed development centers. The efforts to solve these problems have resulted in the thesis presented here.

1.2 Research context

The search for talent anywhere in the world and the attempt to reduce labor costs has caused the rise of distributed software development. This kind of development involves development teams whose members work together but are geographically distributed (Prikladnicki, Audy, & Shull, 2010). Particularly relevant to this work is the nearshore model. With this model companies obtain some of the benefits of the distributed software development, such as the cost reduction, while mitigating the difficulties imposed by distance and cultural differences (Carmel & Abbott, 2007). Similarly to the nearshore model, the rural outsourcing model proposes building development centers in remote domestic locations for several reasons, including to access a more stable, lower-cost workforce (Lacity, Carmel, & Rottman, 2011).

In this context, Spain has established itself as one of the stronger nearshore locations in Europe. The large labor pool in the IT market, the abundance of industrialized software factories and the potential to scale in Latin America make Spain an ideal candidate to use this model (Davis, Parker, & Shanahan, 2009).

Specifically, in recent years Extremadura has become a key destination for the rural outsourcing model. Both, national and international companies looking for lower costs and skilled staff have established development centers here.

- **Indra**. In 2004 Indra, the largest Spanish company in the IT sector, opens a Software Lab in Extremadura which currently employs more than 300 qualified workers, 70% of them coming from the local university (I. P. Release, 2012). Since 2007 this center has a branch within the University of Extremadura. This collaboration is managed from the university by this thesis supervisor.
- **Teseo**. In 2005 Teseo Software Factory opens as a development center inside the University of Extremadura in collaboration with SGAE, the main management body of musical copyright in Spain (of Extremadura Press Release, 2005). This development center was directed by this thesis supervisor and the thesis author was one of the center developers for more than a year.
- **IBM**. In 2007 Insa-IBM opens a center of technological innovation in Ex-

tremadura where, currently, more than 250 employees are working (I.-I. P. Release, 2007)

- **Accenture**. In 2008 Accenture establishes a collaboration with some regional companies sets its own development center in Extremadura (de Extremadura Press Release, 2008)
- **Ibermática**. In 2013 Ibermática opens a development center in Extremadura with more than 60 software developers (I. P. Release, 2013).

Collaboration with these entities, especially with Teseo Software Factory and Indra Software Lab, with which several research projects were conducted, have help the development of this work.

The key features of these development centers make the use of multi-layer architectures particularly suitable. Dividing a system into decoupled components that communicate only through predefined interfaces perfectly fits the business model, the distribution and industrialization of the developments of these organizations.

Another important event to be considered in the context of this work is the rise of the development frameworks. Development frameworks meaning "large, abstract applications in a particular domain that can be tailored for individual applications. A framework consists of a large structure that can be reused as a whole for the construction of a new system" (Bosch, Molin, Mattsson, & Bengtsson, 1997).

The rise of the Internet led to the creation of numerous web frameworks to simplify the development of web applications. These frameworks aim to provide reusable solutions to several common problems in the development of web applications. To do this, they provide the implementation of different design patterns, general reusable solutions to commonly occurring problems within a given context (Gamma, Helm, Johnson, & Vlissides, 1995).

The early web frameworks focused on simplifying the implementation of the presentation logic in web applications, Struts [1] being the first to have a broad success. This was followed by frameworks to manage persistent information, like Hibernate

[1] http://struts.apache.org

[2] and to provide Inversion of Control (IoC, also known as dependency injection), like Spring [3]. The industrial success of these frameworks was quickly followed by the appearance of numerous alternatives and frameworks to cover different aspects of web applications (Johnson, 2005). The increasing complexity of web applications and the wide acceptance of frameworks by the developer community has led to the appearance of dozens of frameworks covering every aspect of these applications (Vosloo & Kourie, 2008).

This kind of frameworks fits perfectly with the use of multi-layer architectures in distributed development centers. One or more frameworks are used in the implementation of each individual layer and development centers, or even teams within them, can specialize in the use of certain frameworks advancing in the industrialization of their developments.

This work specifically focuses on such developments. The development of framework based, multi-layered web applications taking place in distributed development centers. The results of this thesis are transferable to any technology and framework of this type. However, due to its wide acceptance the remainder of this work focuses on Java development frameworks.

1.3 Problem statement

Given the context described in the previous section, the development of frameworks based multi-layer applications has significant benefits but it also has a number of drawbacks. These drawbacks are derived from three different sources. Problems derived from the use of multi-layer architectures, problems derived from the use of development frameworks and problem derived from the distributed development model. Each of these problems are described in the following subsections respectively.

[2] `http://hibernate.org`
[3] `http://projects.spring.io/spring-framework/`

1.3.1 Multi-layer architectures problems

A considerable number of the applications being developed today, specially in distributed development center as the ones mentioned above, are enterprise applications. These are large software systems that deal with a wide range of processes in an organization. They are usually complex, scalable, distributed systems with multiple user interfaces or access methods (web, mobile, APIs, etc). Such applications are characterized by a fairly strict set of non-functional requirements regarding reliability, performance, integration, security, etc., reflecting their function in supporting critical processes within the organization (Fowler, 2002).

Multi-layer architectures are one of the most common solutions to develop such projects since they allow developers to focus on the application's business logic instead of its structural details. However, the responsibility for the effective use of these architectures lies with each individual development team (Pressman, 2000). Specifically, the figure of the software architect takes on particular importance since the architecture plays a very important role in the way the application will be developed (Northrop, 2003).

This situation is only getting worse, as developments are becoming more complex every day. Especially since the boom in cloud computing and mobile devices that makes application architectures increasingly complex and with more layers to take into account.

Thus, a development success will largely depend on the architect's experience, expertise, and skill in avoiding the introduction of potential errors (Dalgarno, 2009). "The right architecture paves the way for system success. The wrong architecture usually spells some form of disaster" (Northrop, 2003). "Examples of the most serious computer-related accidents in the past 20 years such as Therac-25 and Ariane 5 can be attributed to flawed system and software architectures" (Wu & Kelly, 2006).

Defining the architecture requires the architect to follow an arduous and complex process for getting information on the system requirements and for making decisions about how to structure the application to comply with them (Bengtsson, 1998; Bass, Clements, & Kazman, 2003; Clements et al., 2007). Firstly, the architect has to acquire a great knowledge on the requirements and the relationships between them (Capilla, Babar, & Pastor, 2012). Subsequently, the knowledge extracted from the analysis of the requirements is used as the basis for making decisions about how to structure the system (Clements, 2001; Lung & Kalaichelvan, 2000). This implies that the architect cannot make these decisions based on a single requirement, she must have a complete view of all the requirements and how they interact. This conjuncture complicates the architect's work and exposes her to situations in which a misinterpretation can lead to the selection of an incorrect architectural pattern.

Moreover, architectural patterns and development technologies are closely interrelated. The application of a given pattern favors the selection of other patterns (Harrison & Avgeriou, 2010). Therefore, the incorrect selection of a pattern can lead the architect to make incorrect decisions during the refinement of the architecture. This may cause the final design to fail the requirements of the system, jeopardizing the success of the project.

1.3.2 Development framework problems

Development frameworks have become one of the most used tools in complex software development. Although their presence in scientific literature is not too prolific nowadays (it was during the late nineties), their proliferation (Vosloo & Kourie, 2008; Shan & Hua, 2006), the number of versions released annually, the traffic generated in forums and mailing lists, and the job offers that require their skills (Raible, 2012) demonstrate it.

However, their own success is the cause of one of its major drawbacks. The increasing amount of frameworks and their rapid evolution rate (Raible, 2007) make it really difficult to keep up-to-date knowledge about them. This problem affects not only software developers, who have to devote more effort to stay updated, but it's really detrimental to software architects. They should know not only the latest technologies and their impact on the developed projects, but should also understand the effects of the combination of several technologies on the same project (Harrison & Avgeriou, 2010).

Other problems emerging from the use of development frameworks, like the integration of multiple framework or the difficulty of understanding one of these technologies, are well know (Bosch et al., 1997; Mattsson, Bosch, & Fayad, 1999). However, problems like the difficulty of integrating frameworks from different vendors remain present. Software architects with years of experience in the use of these technologies still have to devote significant efforts to incorporate new frameworks or new versions of existing frameworks into their projects, even in cases where the frameworks belong to the same vendor (Paraschiv, 2013). Evidence of this can be found in the number of queries in this regard in professional development forums or in the importance given to this aspect by some of the most representative frameworks developers like Spring (Johnson et al., 2013a, 2013b).

1.3.3 Distributed development problems and competitiveness in the software development industry

As mentioned above, the use of multi-layer architectures and development frameworks fits perfectly with the distributed software development model. However, this development model poses its own drawbacks (Prikladnicki, Audy, Damian, & de Oliveira, 2007).

One of the main goals pursued by companies with distributed development centers is cost reduction. In an industry like software development, where developers work hours are the main raw material, this implies that the cost reduction achieved by these development models comes largely from the lower labor costs in the places where they are located. Competitiveness to offer cheaper development rates causes high staff rotation since a significant number of developers will move to places where they can obtain a better salary (Ågerfalk & Fitzgerald, 2008). The relationship of the author of this thesis with several development centers in the region serves to confirm this problem locally.

Internal rotation should be added to this external rotation. In this kind of development centers several projects are conducted simultaneously and it is common that the staff change from one project to another every few months. This high staff rotation makes the training costs rise, thus decreasing the effectiveness of the development model.

Large software development organizations have tried to minimize the problems caused by staff rotation by using reference architectures. These architectures enable organizations to specify a core software architecture that can be used in multiple developments. Therefore, all projects developed using the reference architecture have a similar structure and use the same set of technologies. This allow organizations to decrease their training costs since they only have to train their staff in the technologies included in their architecture. Also, managing staff rotation and reusing previously developed components becomes easier.

Proof of the success of these reference architectures can be found in Spain, where several regional governments have defined their own reference architectures. These architectures are imposed in outsourced projects so local development centers should use them. Some examples of the most relevant are openFWPA from Asturias (de Asturias, n.d.), MADEJA from Andalucia (de Andalucía, n.d.), AMAP from Cantabria (de Cantabria, n.d.) or JAVATO from Murcia (de Murcia, n.d.). Private companies have used the same solution to similar problems in a less public way. Thanks to the author's collaboration with some of the local development centers it can be assured that at least two of them use their own reference architecture.

However, reference architectures have some major drawbacks of their own. Technological evolution makes these architectures become outdated very quickly and the costs to stay updated grow with the complexity and richness of the architecture. This implies that most of these architectures become obsolete within a few years after its creation. Additionally, these architectures restrict the technological spectrum of the organization know-how, focusing it on the technologies included in the architecture. This becomes a problem when a better technology appears or when a development has specific technological requirements which differ from the architecture.

1.3.4 Final outcome

Each of the problems stated in the previous sections can usually be resolved.

The complexity of multi-layer architectures is counteracted by the software architects experience. The job of a software architect requires decision making that play an important role in the success or failure of projects (Kruchten, 2008). That is why such position is usually occupied by people with proven experience and able to face the challenge of designing a complex architecture.

The quick evolution rate of development frameworks is barely faster than the general evolution of software engineering. A significant part of software engineering has been changing dramatically each decade (Boehm, 2006). This implies that software development professionals are, normally, used to the quick evolution of the technologies used and training in new technologies is considered part of their regular work.

The continuous staff rotation in distributed development centers is balanced by the industrialization of the developments. This industrialization involves a standardization that allows to replace staff with little cost, as long as the staff know the standards employed. As mentioned above, reference architectures also help to mitigate the staff rotation problem, as long as enough resources can be allocated to keep them updated.

The real difficulties occur when these conditions come together. Projects use increasingly complex multi-layer architectures, development frameworks with high evolution rate and are developed by staff not necessarily experienced, including the software architect. This thesis is focused in such situation

1.4 Thesis goals

This thesis focuses on mitigating the problems described in the previous section. The development of multi-layer application in development centers with high staff rotation, the complexity of the multi-layer architectures and the quick evolution of development technologies can be overcome with mechanisms to help the software architect and developers of such applications.

By studying the state of the art regarding the problems mentioned in the previous section, a set of concerns has been identified that constitutes the goals to be achieved in this thesis. In particular, the objectives of this work are the following:

- Identify the most common architectural decisions in the development of framework based multi-layer applications.
- Organize such architectural decisions in a common taxonomy that can be used to help software architects.
- Facilitate the reuse of architectural knowledge obtained during the development of a project in future developments.
- Analyze the integration and usage patterns of a significant number of development framework.
- Specify a common set of rules for the usage and integration of the largest possible number of frameworks.
- Define a process that simplifies obtaining a specific design for a multi-layer architecture, starting from an architectural and technological independent design.
- Provide a mechanism to help the architect to define such specific design.
- Provide a mechanism to automatically generate part of the applications being developed from the specific design.
- Develop tools to support the fulfillment of all the above objectives.

1.5 Proposed solution

To solve the problems mentioned in Section 1.3 and fulfill the goals detailed in Section 1.4 the **ArchLayer** development process is proposed. This process can be used by distributed development centers for building framework based multi-layer applications. Figure 1.1 shows a diagram of the ArchLayer process highlighting the specific contributions of this thesis.

The contributions of this thesis are divided into two categories: contributions and

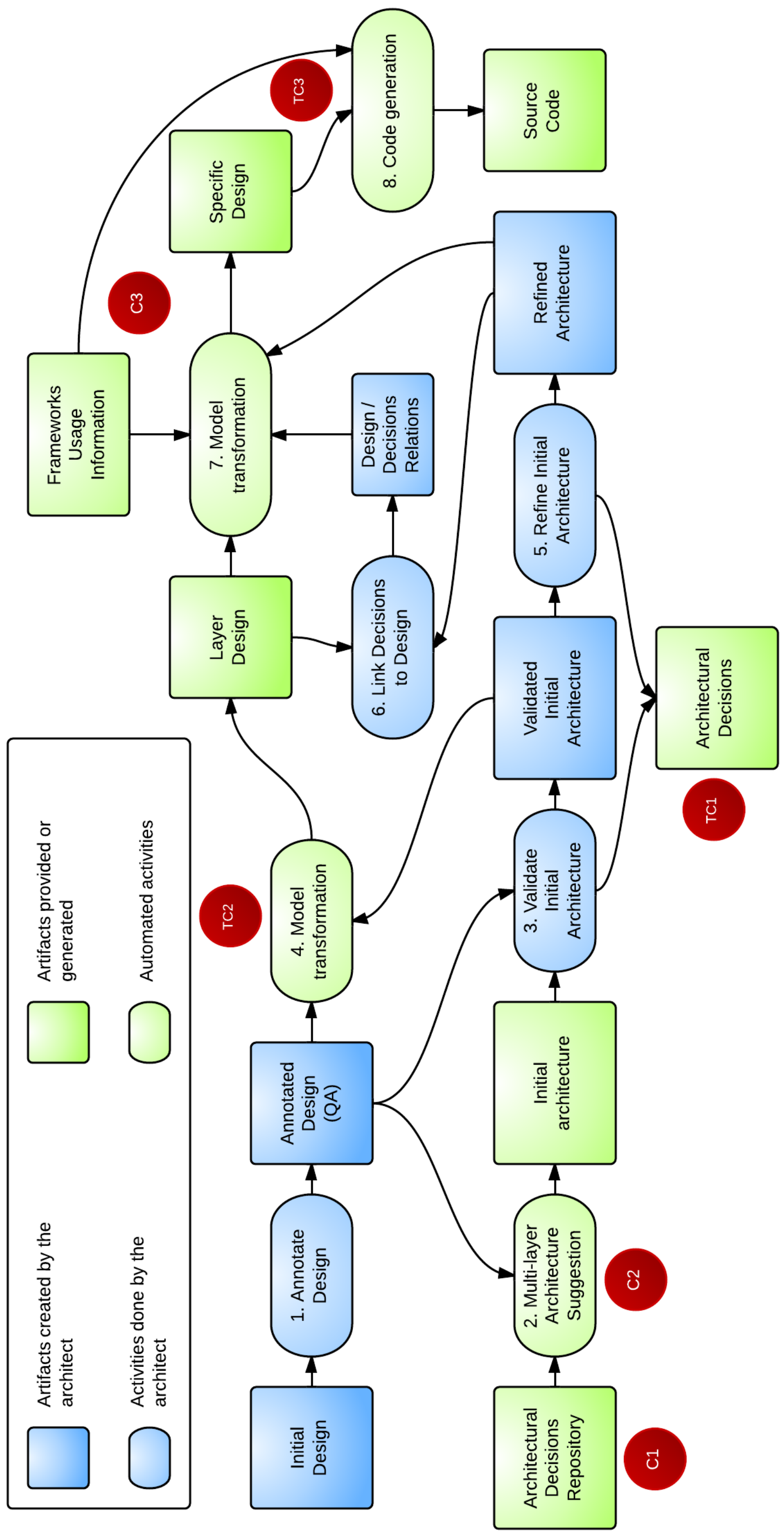

Artifacts created by the architect
Artifacts provided or generated
Activities done by the architect
Automated activities
Initial Design
1. Annotate Design
Annotated Design (QA)
4. Model transformation
TC2
Layer Design
6. Link Decisions to Design
Design / Decisions Relations
7. Model transformation
Frameworks Usage Information
C3
Specific Design
8. Code generation
TC3
Source Code
Architectural Decisions Repository
C1
2. Multi-layer Architecture Suggestion
C2
Initial architecture
3. Validate Initial Architecture
Validated Initial Architecture
5. Refine Initial Architecture
Refined Architecture
Architectural Decisions
TC1

Figure 1.1: Overview of the thesis contributions.

technical contributions. Contributions present a scientific advance over the current state of the art. Technical contributions represent the application of state of the art techniques into the context of framework based multi-layer applications developed in distributed centers.

The ArchLayer process begins with a preliminary design of the application to be developed. This design has to be refined by the architect or requirements experts, in the activity 1 in the diagram, to include information about the quality attributes of the system.

Once the architect has the refined design, the next task is to define the software architecture best suited to the applications requirements. In order to simplify this task a repository containing common architectural decisions is offered to the architect. Such repository is the first contribution of this thesis, it contains an organized set of common architectural and technological decisions in the development of multi-layer applications. This contribution has been published in (García-Alonso, Olmeda, & Murillo, 2012).

The second contribution of this thesis consists in a mechanism to automatically suggest the architect, based on the refined design and the architectural decisions repository as shown in activity 2 in the diagram, an initial multi-layer architecture. After the architect has validated, in activity 3, and refined, in activity 5, the suggested architecture, the architectural decisions taken by the architect are stored so traceability can be maintained to the source code of the project developed and so that they can be used as a basis for decision-making in future developments. A mechanism to store and reuse such architectural decisions is the first technical contribution of this thesis. These contributions have been published in (García-Alonso, Olmeda, & Murillo, 2013).

The second technical contribution of this thesis is a set of model transformations for converting the refined design into a design tailored to the architectural decisions made by the software architect. This transformation is performed in two steps. The first, represented by activity 4 in the diagram, provides a design tailored to the architecture layers. The second, represented by activity 7, provides a design tailored to the specific technologies that will be used in the system development. This contribution is currently under review to be published in (García-Alonso, Olmeda, & Murillo, 2014a).

To perform this second transformation two additional elements are needed. First, the architect must link the technological decisions with the design elements to which they affect, as shown in activity 6. Second, specific information is needed about how the technologies chosen by the architect should be integrated and used in the project. This is precisely the third contribution of this thesis, a common language for specifying the usage information of different development frameworks. This contribution has been published in (García-Alonso, Olmeda, & Murillo, 2014b).

Finally, the third technical contribution of this thesis consists of a set of code generation transformations. These transformation take as input the tailored design and the specific information about the technologies and provide a significant part of the source code of the application, corresponding to activity 8 in the diagram. This contribution has been published in (García-Alonso, Olmeda, & Murillo, 2010).

1.6 Thesis context

This thesis has been developed in the context of the Quercus Software Engineering Group of the University of Extremadura, where the author holds a position of assistant professor. As a member of this group the author has participated, during the development of this thesis, in several research projects that have resulted in several publications and contributions, always related to the objectives of this work.

The following sections detail the research projects in which the author has participated, the publications obtained related to the major contributions of this thesis and the collaborations conducted.

1.6.1 Research projects

The work of this thesis has been developed as part of the following research projects, both regional and national:

- **Model-driven development of business process in software factories: applications to the Web 2.0 and J2EE multi-tier architectures (TIN2008-02985).** In this project development processes used in distributed development centers were analyzed. From this work, we began to define how model-driven development (MDD) techniques could simplify such processes.
- **JACA. Java for the regional government applications (PDT08A034).** This project was conducted in collaboration with the regional government and Indra local development center. The objective of this project was to develop the software architecture to be used in all projects commissioned by the government. Specifically, the project consisted of developing a reference architecture as the ones described in Section 1.3.3. The development was based on Indra reference architecture and a set of tools was built implementing a large part of the contributions of this work.
- **Updating the development of framework based multi-layer applications: Integrating MDD and PL techniques (TIN2011-24278).** In this project, product lines (PL) techniques were used to manage the variability of architectural decisions in the development of multi-layer applications. The first step to use this technique in a MDD process were done during this project development.
- **PATTERN. Product lines and transformation techniques in the multi-layer architectures design (TIN2012-34945).** In the context of this project, we develop the set of transformation to bind PL artifacts containing architectural decisions with the design of a multi-layer application.
- **Product lines in the development of multi-layer applications (ACVII-08).** In this project, we define how to use product lines techniques to store the architectural decisions taken during the development of multi-layer applications for its future use.
- **Methodology for use case extraction from business processes (ACVII-09).** In this project we contribute to define an annotated use case model that would include information about the quality attributes (QAs) each use case

should meet.

1.6.2 Publications

Table 1.1 shows a summary of the papers published and the forums in which they have been published. The complete list of papers published in the context of the thesis is detailed in Section 6.2. The importance of the conferences is obtained from the Computing Research and Education Association of Australasia (CORE)[4]. The importance of the journals is obtained from the Journal Citation Report (JCR)[5].

As can be seen in Table 1.1, in the development of this thesis a total of fourteen papers has been published, of which eight have national scope and six have international scope. Eleven of these papers were published in conferences/workshops, of which two were accepted in conferences indexed in the CORE ranking. Finally, the other three papers were published in journals, of which one is indexed in JCR, with an impact factor of 1,616.

Table 1.1: Summary of the published papers and their relevance.

Forum	**Type**	**Scope**	**Num**	**CORE**	**JCR**
DSDM	Workshop	Nat	1	-	-
JENUI	Conference	Nat	1	-	-
JCIS	Conference	Nat	1	-	-
JISBD	Conference	Nat	4	-	-
Wiki4SE	Workshop	Int	1	-	-
FOSD	Conference	Int	1	-	-
ICSEA	Conference	Int	1	C	-
SERA	Conference	Int	1	C	-
REICIS	Journal	Nat	1	-	-
Agile PME	Journal	Int	1	-	-
IEEE Software	Journal	Int	1	-	1,616
Total	**2 Works.** **9 Conf.** **3 Journal**	**8 Nat.** **6 Int.**	**14**	**2**	**1**

[4] http://core.edu.au/index.php/categories/conference%20rankings
[5] http://thomsonreuters.com/journal-citation-reports/

1.6.3 Collaborations

During the course of this thesis, there have been numerous collaborations with different organizations. Following are those most relevant to the development of this work:

- **GEPRODIST. Distributed project management.** This project was developed in collaboration with Indra development center. The project goal was to build a system that would facilitate the management of distributed projects. The system take as input the software process to be used and is responsible for adapting it to the specific needs of each development. The development process proposed in this work was one of those used in the creation of this system.
- **GLOCO. Global connector.** Some of the ideas detailed in this thesis were applied in this project for the development of a system that connectedthe various sources of information used by LatinAutor, the organization grouping the Latin American musical copyright societies.
- **DCI.Data clean-up and integrity.** This project involved the development of an application that would support a set of processes to manage and verify the data quality of musical copyright societies. To develop this application some of the ideas proposed in this thesis were applied.
- **Gloin.** During the development of this thesis, the company Gloin was cofounded by the author of this thesis, its supervisor and another research partner. One of the company's goals is to bring companies the research advances made in the research group. The process proposed in this thesis has been successfully used by the company in several commercial projects.

Moreover, during the development of this thesis the author made a five months research stay at the IT University of Copenhagen. This university is considered a top institution on system and software engineering.

This stay was supervised by Prof. Muhammad Ali Babar. The main research areas of Prof. Babar include the development and/or rigorously evaluation of approaches and tools for supporting the design, analysis, and evolution of complex and dependable software intensive system and services that meet both the functional and non-functional requirements as derived from the quality goals.

As a result of this stay, and thanks to the experience and contributions of Prof. Babar, we have improved the management of quality attributes in this work as well as the way architectural decisions affect these attributes in the final system.

1.7 Structure of this dissertation

This dissertation is organized as follows:

Chapter 1 Introduction. It comprises this introduction, which contains the research context of the work developed needed to understand the content of this dissertation, a detailed statement of the most common problems in such context and the goals to be achieved with this work. Additionally, it contains a summary of the proposed solution to achieve the aforementioned goals and a description of the context in which this thesis has been developed.

Chapter 2 Related work. It provides specific information about relevant scientific works in the scope of the research context of this work. Specifically, this chapter provides a detailed description of works in the areas of web engineering and development frameworks summarizing all the issues identified. Additionally, the concepts related to variability management and architectural decisions that are the works on which this thesis is based are introduced.

Chapter 3 Bridging the gap between design and implementation. It details the process proposed in this dissertation to solve the problems stated in the previous chapter. Specifically, a running example is used to illustrate the proposed process and the different activities and artifacts that compose it are thoroughly detailed.

Chapter 4 JACA Code Generation Tool. It describes the tool developed to automatically generate part of the source code of multi-layer framework based web application based on the process described in the previous chapter. Specifically, the context and motivation behind the development of this tool is detailed and its use, both as part of the proposed process and as an independent tool, is explained.

Chapter 5 Validation. It describes the validation performed to evaluate the process usefulness. Specifically, the two industrial projects in which the proposed process was used are detailed and the results of the application of the process are presented and discussed.

Chapter 6 Conclusion. It includes the main conclusions and reflections drawn after development of this thesis.

Appendix A Architectural Decisions Repository. It shows the complete architectural decisions repository that was built as part of the research work done during this thesis.

Appendix B Framework information model. It shows an example of a complete framework information model.

Appendix C Model transformations. It shows the ATL model transformations that support the process presented in this thesis.

Appendix D Additional material. It includes a detailed list of the additional material presented alongside this thesis.

Chapter 2

Related work

In the moment when I truly understand my enemy, understand him well enough to defeat him, then in that very moment I also love him. I think it's impossible to really understand somebody, what they want, what they believe, and not love them the way they love themselves.

Ender's Game, Orson Scott Card.

As far as the author knows, there are no other works focused on the development of framework based multi-layer applications in distributed development centers. However, this thesis, like any other scientific work, is based on previous works. These works have enabled the author to develop a solution to the problems detailed in the previous chapter.

This chapter reviews the state of the art related to this thesis. This review includes works from different fields and disciplines related to model driven engineering, product lines, software architecture and development frameworks. The review of this works is organized as follows. Section 2.1 summarizes the problems that drive the development of this thesis and, therefore, the analysis of the related works. Section 2.2 provides a detailed description of works in the area of model-driven engineering given their proximity to the work presented here. Section 2.3 discusses the most relevant works dealing with development frameworks. Section 2.4 introduces the concepts related to product lines and architectural decisions that are the basis of this work. Finally, Section 2.5 presents the conclusions of this chapter.

2.1 Introduction

As stated in the previous chapter, companies developing framework based multi-layer applications in distributed development centers face three main problems.

- Multi-layer architectures are very complex and have a high degree of variability. This requires companies to have expert software architects and, therefore, they become very dependent on them.
- Development framework evolves very quickly, with new frameworks or new versions of the existing frameworks appearing constantly. This requires companies to devote significant resources to the continuous training of its staff. Again, this makes companies become very dependent on their trained staff.
- The business model used in distributed development centers causes a continuous staff rotation. Companies face this problem by industrializing their development process and by using reference architecture. This requires companies to spend resources on training the rotating staff and in keeping the reference architecture updated.

As can be seen, these problems are tightly coupled. Each one of them magnifies the others, worsening the situation.

Multiple research works try to solve any of these problems or a combination of them. The rest of this chapter will review the most relevant of them in relation to the work presented in this thesis.

2.2 Model-Driven engineering

Model-driven engineering techniques are designed to rise the abstraction level at which architects and developers work by focusing in the use of models at a higher abstraction level than code-oriented models. These techniques have been used to help mitigate the problems of technological evolution and complexity of the developments stated above. In this section, the model-driven proposal related to the work presented in this thesis are reviewed.

2.2.1 Foundation

Model-Driven Engineering (MDE) is a paradigm for increasing the abstraction level in the development of different projects by the systematic use of models as primary artifacts. With a model being a simplification of a system built with an intended goal in mind and able to answer questions in place of the actual system (Bézivin & Gerbé, 2001). In the specific case of software development, MDE proposes the use of models to obtain a higher abstraction level than the one provided by programming languages (Schmidt, 2006).

The use of MDE techniques to solve the problems faced during the development of multi-layer applications is a natural step. Models are already used by distributed development centers to represent projects requirements, generally use case diagrams (Jacobson, Booch, & Rumbaugh, 1999; Leffingwell & Widrig, 2003).

Within the MDE paradigm there are numerous proposals for software development. One of the most representative examples is the Model Driven Architecture (MDA) architectural framework (Miller & Mukerji, 2003) from the Object Management Group (OMG), an international, open membership, not-for-profit technology standards consortium (`http://www.omg.org`). MDA promotes the creation of highly abstract models that are developed independently of the implementation technology. This models can be automatically transformed into less abstract models, code skeletons or other artifacts (Kleppe, Warmer, & Bast, 2003).

In MDA models of a system are grouped into three categories. Computation Independent Models (CIMs) are views of a system from a computation independent viewpoint. Platform Independent Models (PIMs) are views of a system from a platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type. Platform Specific Models (PSMs) are views of a system from a platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform (Miller & Mukerji, 2003).

Closely related to MDA, there are a others standards and technologies relevant to the work presented in this thesis. In MDA, models are first-class artifacts and, therefore, they should be defined in a common language that allows their manipulation by all involved in the development. Meta-Object Facility (MOF), another OMG standar, provides the basis for model management (OMG, 2011b).

MOF proposes a four layer infrastructure that support the generation and representation of arbitrary meta-models (Atkinson & Kühne, 2003). The Meta-Meta-Model Layer or M3 contains the MOF language, which is used to describe the structure of metadata (and, also, of MOF itself). It provides a meta-meta-model at the top layer. The Meta-Model Layer or M2 contains definitions for the structure of meta-data. The M3-model is used to build meta-models on level M2. The Model Layer or M1 contains definitions of data in the information layer. The meta-models of level M2 describe the structure of elements of the M1- layer. The Model Layer or M0 contains objects or data in the information layer (Andova, van den Brand, Engelen, & Verhoeff, 2012).

MOF is the international standard to define and manipulate a set of interoperable meta-models and their corresponding models (ISO, 2005). In particular, the more widespread implementation of the standard is the Eclipse Modeling Framework (EMF), which has become very popular and has an extensive tool support (Steinberg, Budinsky, Paternostro, & Merks, 2009).

Aditionally, within MDE, model transformations are one of the key elements. Model transformations can be defined as programs that takes as input a model conforming to a given source meta-model and produces as output another model conforming to a target meta-model (Ruscio, Eramo, & Pierantonio, 2012). There are numerous proposals for model transformation languages and techniques (Czarnecki & Helsen, 2006; Taentzer et al., 2005). However, there is no model transformation standard accepted by the vast majority yet (Ruscio et al., 2012).

The Atlas Transformation Language (ATL) (Jouault, Allilaire, Bézivin, Kurtev, & Valduriez, 2006) is usually used over other proposals such as the OMG standard Query/View/Transformation (OMG, 2011a) or others because its extensive tool support (Jouault, Allilaire, Bézivin, & Kurtev, 2008) and its wide acceptance by the

MDE community (`http://www.eclipse.org/atl/usecases/`).

Finally, the remaining piece in the MDE mosaic is the Object Constraint Language (OCL). OCL is another OMG standar (OMG, 2012) that has become a key component of any MDE process. OCL is frequently used to express model transformations, well-formedness rules or code generation templates (Cabot & Gogolla, 2012).

2.2.2 Web engineering

Web engineering can be defined as "the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based system and applications" (Murugesan & Deshpande, 1999). In this area a significant number of works have successfully applied MDE techniques to the development of web applications (Moreno, Romero, & Vallecillo, 2008).

A significant portion of the framework based multi-layer applications being developed in distributed development centers are web applications. Additionally, web engineering works usually use multi-layer architectures implicit in the models of the different proposal. And also, although it is not explicit in these works, most web engineering approaches generate applications source code using development frameworks (Vosloo & Kourie, 2008).

Therefore, in this section the most important works in this area are detailed. Most of these works present a complete process for developing web applications. In this review the focus is set on the models used by each proposal, by themselves these models provide a lot of information about the architecture used by the applications developed, and in the proposed process for the development of web applications, for the similarities or differences these processes can have with ArchLayer. Special attention will be paid to the contributions of these proposals to try to solve the problems mentioned in Section 2.1

Web Modeling Language

Web Modeling Language (WebML) is one of the more successful web engineering techniques. It was conceived to support the design and implementation of data intensive web applications using MDE techniques (Ceri, Fraternali, & Bongio, 2000). Its goal is to allow web developers to express the core features of a web application at a high abstraction level, without having to focus on the architectural details. To this end, WebML proposed the use of four different models:

1. **Structural model.** This model is used to express the data content of the web applications, including the relevant entities and the relationships between them. Instead of proposing a new modeling language for modeling the data contents of the applications, WebML was made compatible with some classical notations like UML class diagrams or entity–relationship models.
2. **Hypertext model.** This model is composed of two sub-models to describe the hypertexts of the application being developed.
   (a) **Composition model.** This model is used to specify the different pages composing a hypertext and the content units forming each page. Initially, six types of content units were defined to compose the pages. They range from publishing the information of a single object to different ways of representing a set of objects. On top of these content units, composition units were defined to link the content units with the underlying entity in the structural model providing the content.

   (b) **Navigation model.** This model describes how pages and content units are linked to form a hypertext. Links can be non-contextual, if they connect independent pages, or contextual, if the destination link depends on the content of the source.
3. **Presentation model.** This model is used to express the layout and graphical design of the application pages. This model is intended to be independent of the language and device used to actually render the page. This model includes two presentation specifications. Page specific presentations are used on a single

page and include references to its content and generic presentations are based on predefined models independent of the page content and include reference to generic content elements.

4. **Personalization model.** This model is used to represent the application user and user groups so group-specific or individual content can be used to customize the application. The personalized content can be used to customize the composition model as well as the presentation model.

These models are used in an iterative and incremental process to develop web applications. The first phase of the process involves the requirements analysis. Using the business requirements as input, the main results of this phase are the identification of the group of users addressed by the applications, the functional requirements, the core information objects and the decomposition of the application into views. Any format can be used to specify the requirements of the applications, including use case diagrams and activity diagrams, which are used in the work presented in this thesis with the same purpose.

The next phase on the development process is the conceptual modeling phase. It consists of defining conceptual schemes that express the organization of the application at a high level of abstraction, specifically the data design and the hypertext design are modeled in this phase using the above mentioned models. For the creation of these models the requirements obtained in the previous phase are used. These requirements or initial design of the system must be adapted to the level of detail required by WebML models. This is a significant gap, since these models have to be sufficiently detailed to allow the generation of the source code of the application, that must be bridged by the developers without any specific support of this approach. From these models the application is then implemented, tested and evaluated in the remaining phases of the iterative process.

WebML is a clear example of the success of model-driven web engineering. Since its inception, this model-driven technique has been constantly improved and expanded with new features. Throughout the years, WebML has been widely expanded for example with support for representing business actions triggered by users navigation or the support of web service interaction and workflow modeling primitives (Brambilla, Comai, Fraternali, & Matera, 2008; Ceri, Brambilla, & Fraternali, 2009).

In addition to its presence in the research world, WebML also has a strong industrial presence. A company was created in 2001 to bring WebML to the industrial world. The result of this industrialization was the creation of a tool called WebRatio[1]. A commercial tool implementing the different WebML models and able to automatically generate the source code of the web application from such models.

In recent years, the WebML creators have achieved another important milestone. WebML has been used as the basis to define a new OMG standar. The Interaction Flow Modeling Language (IFML) is designed for expressing the content, user interaction and control behavior of the front-end of software applications (OMG, 2013).

Heavily based on WebML, an IFML model is formed by one or more view containers representing web pages. A view container contains view components, which represent the publication of content or interface elements for data entry. View components can have input and output parameters. Containers and views can be associated with events denoting the support of user interaction.

The adoption of this new standard represents the creation of a common framework that can be used by the different model driven web engineering proposals to contribute to the area progress (Rossi, 2013).

The use of this proposal helps to greatly reduce the complexity of developing framework based multi-layer applications. However, to do that WebML significantly reduces the architectural variability. The software architecture used by all the applications developed with WebML is implicit in the set of model used during the development and cannot be changed or adapted to the application requirements. Technological variability is also constrained since it completely depends on the available transformations and no simple extension mechanism is provided to support new frameworks. Finally, the use of the WebML models, while practical and efficient for it purposes, causes an overload on staff training.

[1] `http://www.webratio.com`

UML-Based Web Engineering

UML-Based Web Engineering (UWE) was designed with the idea of finding a standard way for building analysis and design models of web applications. (Hennicker & Koch, 2000). Its original goal was to propose an UML-based design methodology for hypermedia applications. To this end, UWE proposes a UML extension for hypermedia and a three steps iterative design process. This process produces four different models:

1. **Conceptual model.** This model is built taking into account the functional requirements captured with use cases. Traditional techniques are used to construct this model, such as finding classes, defining inheritance hierarchies, etc.

2. **Navigation space model.** This model is based on the conceptual model. It specifies the objects in the conceptual model that can be visited through navigation in the web application. Two modeling elements are used in the construction of this model. Navigational classes model the instances of a class that are visited by the users during navigation. Direct navigability associations model the existence of a navigational path from the source class to the target class in the web application.

3. **Navigational structure model.** This model details how the objects present in the navigation space model can be reached by the users. For this purpose various elements are used. Specifically, indexes, guided tours and queries. Navigational choices are represented by menus.

4. **Presentation model.** This model defines how the information within the navigation space is presented to the users. The presentation model focuses on the structural organization of the presentation and not on its physical appearance and it is based on the use of framesets.

UWE proposes the use of at least one kind of UML diagram or an extension of them for the visualization of each of these models. Additionally, other diagrams are used to represent behavioral aspect of the application.

The process for the developing of web applications proposed by UWE follows the MDA approach and is based on OMG standards. The process start with a CIM used to specify the requirements of the application. These requirements are modeled, as in this thesis, using use cases and activity diagrams.

From this initial model the PIMs are derived using model transformations. Specifically, the design models represent the different concerns of the application comprising the content, the navigation, the business process or the presentation. These models are combined in a single model representing the big picture of the application.

Finally, from this combined model different PSMs can be generated using model transformation. These PSMs are specific of the technology used to the implementation and are used as the starting point for code generation.

Over time, UWE has been continuously adapted to new features of web applications such as transaction-based application, personalization, context-awareness or asynchrony (Koch, Knapp, Zhang, & Baumeister, 2008). Aditionally, a tool has been developed supporting the described models and the proposed development process, ArgoUWE (Knapp, Koch, & Zhang, 2005).

The use of UML based models, helps UWE reduce the overhead of training the development staff in the use of new models, since UML is widely known in the software industry and extensively used in distributed development centers. However, it still lacks mechanisms to manage architectural variability and technological evolution.

Object Oriented Web Solution

Object Oriented Web Solution (OOWS) is an extension of the OO-Method, an object-oriented model-based software production method, that introduces the expressiveness needed for the development of web applications. To this end OOWS is based on the three models proposed by OO-Method:

1. **Structural model.** This model defines the system structure and the rela-

tionships between the system elements by means of a class diagram.

2. **Dynamic model.** This model is used to describe the valid object-life sequences for each class of the system using state-transition diagrams. Interaction between objects are represented in this model using sequence diagrams.

3. **Functional model.** This model captures the semantics of state change to define service effects. To this end a textual formal specification is used.

To provide support to the web applications specifics properties, three additional models were added to OOWS:

1. **User model.** This model is used to specify the types of users that can interact with a web application. This model customize the navigation of the different users in the application.

2. **Navigational model.** This model was introduced to specify the view over the system for each kind of user defined in the previous model. This model is built in two phases, the first one to define a global view over the navigation and the second one to make a detailed description of the elements previously defined.

3. **Presentation model.** This model is used to represent the presentation requirements of web applications. It is based on the previous model, it uses the navigational information to define the presentation properties.

These models are used to generate web applications.

First, the OO-Method models are used to generate an application using a three-layer architecture. The layers are a persistence layer to implement the access to persistent data and to hide the details of data repository, an application layer to implement the business logic of the application, and a presentation layer to implement the graphical interface.

Then, the OOWS specific models provide all the information needed to generate the interface of a web application. The generated interface is used to replace the presentation layer of the application generated from the OO-Method while keeping the other two layers (Fons, Pelechano, Pastor, Valderas, & Torres, 2008).

As the above proposals, this technique has also been extended to accommodate the evolution of web applications. Enhancements such as web requirements modeling, business processes, semantic web and service-oriented architectures have been incorporated into this method (Torres, Pelechano, & Pastor, 2006).

Again, the architectural variability and technological evolution mechanisms provided by this approach is very limited. A three layer architecture is always used and it cannot be adapted by the architect to meet the application requirements. However, OOWS presents an advantage over the previously analyzed works, since its based in a general proposal it can be used to develop not only web applications but any kind of multi-layer application.

Object Oriented Hypermedia Design Method

The Object Oriented Hypermedia Design Method (OOHDM) is another model-based approach to develop web applications (Schwabe & Rossi, 1998). It allows the developer to specify a web application seen as an instance of a hypermedia model. To this end, several models, each focused on a different aspect of the application, are used:

1. **Requirement model.** This model is used to gather the stakeholder requirements. User interaction diagrams are used to represent the use cases if the system. This model provides a representation of the information flows between the user and the application.

2. **Conceptual model.** Based on the requirement model, this model captures the application domain information using object-oriented modeling principles. This model is expressed used a extended version of UML.

3. **Navigational model.** This model is used to define the navigational structure of each user. It reflects the objects and relationships in the conceptual model that each user can reach.

4. **Abstract interface model.** This model defines the objects containing information to be presented to the users. The element in this model are linked to

elements in the navigationa model from which they iobtain the information to be shown.

5. **Implementation model.** This model is used to map the interface and navigational model to run-time objects. Different technologies can be used to generate the source code of the implementation model.

The process proposed by OOHDM to build web application is composed of five steps, corresponding with the five model previously described. This models can be used to generate the source code that implements the application. This proposal is also continuously extended to accommodate new aspects of web applications (Rossi & Schwabe, 2008; Nascimento & Schwabe, 2013)

This proposal was one of the first to promote the use of a navigational model to define the navigational structure of the applications developed. As mentioned above, a similar approach was later used by others like WebML or UWE. However, it does not provide any support for architectural variability management or technological evolution.

Model-Driven Approach for the Semi-automated Generation of Web-based Applications from Requirements

The main goal of this proposal is the semi-automated generation of design models related to web based application from requirements (Fatolahi, Somé, & Lethbridge, 2008). In order to achieve this goal, the authors propose a process that begins with a requirements model which is transformed into increasingly specific models following the MDA standard. Specifically, the models used in the proposed process are:

1. **Computation independent model.** At the CIM level, author propose the use of use case description and default domain objects. This elements are used to represent the application requirements.

2. **Platform independent model.** From the previous model, a PIM is generated using model transformations. This model includes state machine diagrams, an user interface model and a refined domain model.

3. **Platform specific model.** The PSM is obtained from the PIM by applying a mapping to the specific platform used. In (Fatolahi et al., 2008), authors show this transformation using some existing tool to adapt the PIM to a platform composed of a two-layer architecture using a fixed set of well-known development framework.

Finally, the last model can be used to generate the source code of the application. Over time, this proposal has led to the definition of a meta-model for model-driven web development. This meta-model can be used to model web applications with a higher abstraction level than the one offered by other proposal, and later transformed into more specific model, like the ones proposed by WebML (Fatolahi, Somé, & Lethbridge, 2012).

In contrast to the previous approaches, this one is focused on obtaining the models to develop an application from its requirements. The same approach is used in the process proposed in this thesis. However, like the proposal above this one does not provide any support for architectural variability management or technological evolution.

Interactive Dialogue Model

The Interactive Dialogue Model (IDM) is a dialogue-based design model to shape interactive applications, including web applications (Bolchini & Paolini, 2006). This approach is derived from a previous Hypertext Design Model (Garzotto, Paolini, & Schwabe, 1993). IDM is based in the description of the dialog between the user and the application that has to be supported to meet the requirements. For this purpose, IDM uses the following models:

1. **Conceptual model.** This model represents a conceptual schema of the application. It must contain all the necessary dialogue strategies without focusing on technical details. The primitives used to compose this model include the topic of the conversation, the kind of topic, etc.

2. **Logical model.** This model is used to represent the decisions that are dependent on the channel used to establish a dialogue, for example a web application.

More than one logical model can be defined for the same conceptual model, one for every channel that is going to be used.

3. **Page model.** This model is used to define the elements to be communicated to the user in a single dialogue act. This model is based on a logical model and include all aspects that contribute to define the visual communication strategy of the application.

Unlike the above, the main goal of this proposal is not to facilitate the development of applications, but serve as a communication tool with stakeholders. That is why the nomenclature used differs greatly from other approaches and the concept of dialogue is used to model the applications (Bolchini & Garzotto, 2008). By being more focused in the users of the application, this approaches helps the application developed meet its user requirements but it does not provide support to help the software architect or development team during the development process.

Hera

Hera is another model-driven methodology for the design of web applications (Vdovjak, Frasincar, Houben, & Barna, 2003). It is organized around three design phases: integration, data retrieval and presentation generation. The integration phase manages the gathering of data from different source. The data retrieval phase handles the user queries and produces the data that represent the result. The presentation generation phase transform the result obtained in the previous phase into a web presentation.

Specifically, the model used in Hera are:

1. **Domain model.** This model is used as the starting point of the Hera methodology and it describes the structure of the content data. The main purpose of this model is to define the semantic structure of the content data.

2. **Application model.** This model is based on the domain model and it describes the navigation structure over the content. Its goal is to deliver and present the content to the user in a semantically effective way.

3. **Context model.** This model is used to personalize or adapt the navigation

structure defined in the application model. This adaptation is done dynamically based on the information represented in this model.

4. **Presentation model.** This model contains the concrete presentation details of the applications. It is used in combination with the application model and the context model to generate a suitable representation of the information.

With this approach, Hera specializes in the personalization and adaptation of web application as well as in the inclusion of external data sources (Houben et al., 2008).

A different of the proposal presented above, Hera models are based on the RDF standard (Klyne & Carroll, 2004). However, like the previous ones the software architecture of the applications being developed is inherent to the models used and cannot be modified by the software architect.

Web Semantics Design Method

The Web Semantics Design Method originates in a design method for the development of web sites (Troyer & Leune, 1998). Over the years, WSDM has evolved to focus in the development of semantic web applications. To develop these applications WSDM proposes the following models:

1. **Audience model.** This model is used to identify and classify the users into audience classes. This classification is based on the users requirements.
2. **Task and information model.** This model is used to represent the requirements of each of the audience classes defined in the previous model. In it, the information and functionality needed to satisfy the different requirements is modeled. Additionally, the different tasks that each audience class need to be able to perform are also represented in this model.
3. **Navigational model.** This model is used to define the conceptual structure of the web application and to represent how the different audience classes can navigate through the application.

4. **Site structure model.** This model represents how the components from the navigational model will be grouped into pages.

5. **Page model.** In this model the look and feel of the web application as well as the layout of each page is defined.

6. **Logical data model.** This model is used represent the structure of the information provided by the web application. This model is incrementally constructed during the definition of the previous models and takes the form of a reference ontology.

This approach allows developers to generate semantically annotated web application by means of one or more ontologies, contributing to the semantic web (Troyer, Casteleyn, & Plessers, 2008). However, it does not provide support for architectural variability or technological evolution.

The WebSA approach

The Web Software Architecture (WebSA) approach proposes to incorporate in web engineering techniques, as the ones detailed above, a new concern who has been lacking for other proposal, an architectural concern (Meliá & Gómez, 2006). They propose an extension to these methods to make then more architecturally flexible. To achieve this goal new models with architectural information are used. Specifically, the proposed models are:

1. **Subsystem model.** This model represent the subsystems that will compose the application. The subsystem decomposition proposed on this approach is based on the layer architectural pattern and makes use of additional architectural patterns to determine the best layer distribution for the application.

2. **Configuration model.** This model defines an architectural style based on a structural view of the web application using the web components and the connections between them.

3. **Integration model.** This model is the result of merging the two previous model with the functional models of the application. This approach proposes

the use of functional model from other well established web engineering approaches like WebML or OOWS.

This approach complements the existing web engineering techniques improving the control over the software architecture of the web applications.

In a more recent work, the same authors made a similar proposal to increase the technological and architectural variability of web engineering methodologies for the development of rich internet applications (RIAs) (Meliá, Gómez, Pérez, & Díaz, 2010). In this approach they propose the use of two new models:

1. **RIA feature model.** This model is used to represent the architectural and technological variability of rich internet applications. To represent this information a feature model, as the one described in Section 2.4.1 is used.
2. **RIA component model.** This model is used to define the architectural style of the application, in a similar way as the configuration model was used in the WebSA approach.

This approach was also designed to complement existing RIAs web engineering techniques and not as a standalone methodology.

These proposal improves the architectural variability of other web engineering techniques. However, they do not pay special attention to technological evolution.

Quercus Software Engineering Group

The research group of which the author of this thesis is a member has also made notable works are in the web engineering area. Although the author has had no direct connection with these works, they are listed here for their relevance and proximity to the work done in this thesis.

Of the works done by the Quercus research group stand out those related with RUX-method (Trigueros, Preciado, & Sánchez-Figueroa, 2007; Preciado, Trigueros, & Sánchez-Figueroa, 2008). In these works a methodology was proposed for engineering the adaptation of legacy model-based web application to new user interfaces based on RIAs. The proposal reuses the business logic from the web application being adapted and provides a new user interface. Accompanying this methodology a technology with great commercial success was developed, RUX-Tool.

Another relevant work in this area is the one developed in (Conejero et al., 2013; Rodríguez-Echeverría et al., 2013). A technique is presented in these works to extract models of legacy web applications, so that applications that have been built using certain architectures and development frameworks can be reverse engineered into model-driven web engineering methodologies.

2.2.3 Discussion

This section has detailed the best established works in the area of web engineering. The number of works, their continuous enhancement over time and the commercial success of the tools derived from them prove that web application development is a domain where MDE techniques are particularly well suited.

However, the complexity of the web applications being developed keep growing. New requirements are imposed on web applications and new technologies emerge to cope with them. This causes the detailed proposals to have certain limitations.

Many of these proposals were originally conceived to develop a specific type of web applications. Therefore, they are prepared to deal with a fixed set of concerns. However, they are not prepared to deal with new concerns that may arise (Moreno et al., 2008). By itself, this does not pose a serious problem for these techniques. The research community is very active in this area and each time that a significant evolution in the development of web applications has emerged, every proposals has been updated to support it or external additions to the existing proposals has been developed by other researchers. Proof of this are the many proposals that have been adapted with the boom of RIAs (Preciado, Trigueros, Sánchez-Figueroa, & Comai, 2005; Fraternali, Rossi, & Sánchez-Figueroa, 2010).

However, adding extensions to the different proposals whenever there is an evolution in web applications can lead to overly complex or poorly cohesive methodologies. This has led to proposals such as IDM having to be reinvented after becoming too complex trying to cover too many concerns of web applications (Bolchini & Garzotto, 2008).

Another problem with this type of proposal is that, even when several of them are very similar, each one uses its own modeling languages. This problem is mitigated by the use of the OMG standards that allows researchers to transforms models from a proposal to another or to define extensions that can be applied to more than one proposal. In this regard, a proposal has been made to try to unify various methodologies with little success (Moreno & Vallecillo, 2008). The recent approval of the IFML standard by the OMG can help to mitigate this problem.

Finally, all the proposals discussed here, except WebSA, are tied to a particular architectural style and a limited set of technologies. They do not allow the development of web applications using different platform technologies and software architectures (Moreno et al., 2008). In the rapidly changing world of web applications and development frameworks this problem must be addressed.

That is, precisely, one of the problems addressed in this thesis. The management of architectural and technological variability of web applications in a model-driven process. In this respect, this work has many similarities with the WebSA proposal. The main differences are that the work presented here can be used for all multi-layer applications not only web applications or RIAs and that it pays special attention to technology evolution, while WebSA allows developer to select the technology to be used but does not provide any facility for the incorporation of new technologies.

2.3 Development frameworks

Development frameworks are a widespread tool in the industry of web application development. However, their presence in scientific literature is quite limited in recent years, even though the authors of related topics implicitly assume the availability of development frameworks (Vosloo & Kourie, 2008).

Most of the ideas and arguments regarding development frameworks take place in blogs, newsgroups or discussion forums. Therefore, it seems that most of the evolution of this technologies happens in the world of technical discussion forums and projects. Existing development frameworks are tangible results of this process.

The multi-layer application on which this thesis is focused are implemented using development frameworks. Therefore, in this section the most important works in this area are detailed focusing on how they help solve some of the problems stated in Section 2.1.

2.3.1 J2EE Development Frameworks

In (Johnson, 2005) Johnson describes the beginning of the process that led to the current success of frameworks, especially in the field of web applications developed in Java. The shortcomings of the standards to embrace the changing requirements of web applications, its slow evolution and the fast innovation provided by open source projects led to the creation of the first successful frameworks.

Traditionally, many organizations have used in-house frameworks to solve different platform shortcomings. The increased maturity in web application development led to consensus about the generic problems that required a generic solution. Many development frameworks were built to solve these problems.

Struts[2] was one of the first successful frameworks. It was created to handle the deficiencies in the development of the presentation layer of web applications. Its release made a significant change in the development of such applications. By covering significant deficiencies in the Java standards, Struts quickly became a de facto standard, accepted by developers and organizations alike. This success paved the way for the emergence of development frameworks.

Some time later, Hibernate[3] appeared to support other area not covered by specifications. Specifically, Hibernate focuses on the implementation of the persistence layer of web applications and the access to databases. As Struts, Hibernate or similar frameworks were quickly accepted in the development community.

The appearance of Spring[4] was another important step in the acceptance of

[2]`http://struts.apache.org`
[3]`http://hibernate.org/orm/`
[4]`http://projects.spring.io/spring-framework/`

frameworks. Spring was designed to support the business logic layer of web applications by providing inversion of control and aspect oriented programming. Spring along with Hibernate and Struts can give support to the more usual layers of a web application using only development frameworks. The popularity that these initial frameworks had along with the important advantages they provided led to the appearance of hundreds of frameworks supporting any aspect of a web applications.

This work provides support to the rationale for the work done in this thesis. It explains the emergence and success of development frameworks that partly motivated the work presented here.

2.3.2 Server-centric Web frameworks

In (Vosloo & Kourie, 2008) the authors present a survey of 80 server-centric web frameworks, development frameworks that support the development and execution of web-based user interfaces. As a result of this survey two taxonomies are presented reflecting two orthogonal ways of characterizing a framework.

The first taxonomy is based on the strategies used by frameworks for handling the view concerns of web applications. The basic problem addressed by these strategies is how web applications that receive a request can generate a response to display part of the user interface in a browser.

The second taxonomy is based on the strategies used by frameworks for routing events between the view and the model while keeping them synchronized but decoupled. The basic problem addressed by these strategies is how to relay the events generated in the browser to the model and how to generate a view in response.

The taxonomies presented in this paper are interesting in themselves and allow for better categorization of server-centric web frameworks. Besides, this article is relevant to the work presented in this thesis by another set of reason.

- This work demonstrates the explosion in the number of available frameworks. It analyzes 80 frameworks covering a single specific aspect of web applications. Part of the problems addressed in this thesis comes from the large number of frameworks available.

- In this work, the lack of research on development frameworks is emphasized. Numerous research works use development frameworks in different contexts, but in most cases the frameworks themselves are not discussed. When they are, research papers usually refer to the specifics frameworks used or studied and they rarely address frameworks as a category with many representatives.
- This paper highlights how model-driven web engineering proposals generate web application code that uses development frameworks. However, this works usually do not mention the frameworks used or analyze their impact in the generated application.
- The taxonomies presented in this papers do not classify frameworks directly because individual frameworks can employ more than one strategy for dealing with certain problems. This implies that the same framework can be categorized a number of different ways on each taxonomy depending on how developers decide to use it. The same situation is contemplated in the work presented here. This may lead to the same framework appearing more than once in the repository of architectural decisions, representing the different ways it can be used to solve the same problem.

2.3.3 Programmer Questions about Framework

In (Hou, Wong, & Hoover, 2005; Hou & Li, 2011) Hou et al. analyze the problems presented by development frameworks by studying the questions written by developers in forums and newsgroups. The goal of these studies was to suggest improvements in the design, documentation and programmer practice of these tools.

Both works focus on the Swing framework, a Java graphical user interface framework that was chosen for being mature, documented, and widely used. Specifically, the studies are centered on two of the more than 30 components forming Swing. Authors collected, analyzed, and categorized hundreds of questions related to the chosen Swing components. As a result, the authors present a set of problematic features of the framework studied categorized as design problems, documentation problems, and people problems.

This work helps validate one of the starting hypothesis of this thesis, development frameworks are difficult to use. Even Swing, a stable framework that is part of the standard programming language platform, pose serious problems to developers who try to use it.

If this example is transferred to the context in which this thesis is developed, web frameworks evolving rapidly and distributed development centers with high staff rotation, the complexity of the addressed problems can be better appreciated.

The solutions proposed by the authors of these papers are usually aimed at improving the quality of a framework and the understanding obtained by programmers when learning a new framework. However, they do not address the problems associated with the integration of multiple frameworks in a project or caused by changes that happen in the evolution of a framework.

2.3.4 Framework Usage Templates

As the works described above, in (Heydarnoori, Czarnecki, & Bartolomei, 2009; Heydarnoori, Czarnecki, Binder, & Bartolomei, 2012) Heydarnoori et al. propose a technique for simplify the use of a development frameworks. Their approach is used to automatically extract templates of the implementation of framework concepts from the traces of sample applications.

Concept implementations templates are pieces of pseudo-code summarizing the implementation steps for instantiating a given framework concept. These templates are automatically extracted from execution traces recorded from the interactions between the code of sample applications and the framework API when the desired concept is instantiated. The extracted templates can be later used as an entry point for developers to explore concept implementations in the sample applications.

In these works, the presented approach is used to extract templates of 14 concepts from five widely used Java frameworks. The templates were compared with framework documentation in aiding developers in performing concept implementation tasks. The results obtained suggested that the choice of templates versus documentation improved the implementation time.

Again, these works help validate the initial hypothesis of this thesis that frameworks are difficult to use. Moreover, these studies demonstrate that developers productivity can be improved with better documentation of the framework concepts to implement.

The work presented in this thesis does not attempt to improve the available framework documentation. However, similar ideas to those used in these works are applied. Adapting the initial design of an application to the frameworks chosen by the software architect for implementation simplifies the work of developers in locating the necessary documentation. Furthermore, the usage information of development frameworks used in the work presented here and mentioned in Section 1.5 is similar in many ways to the templates presented in these works. The main difference is that in this thesis this information is used by model transformation to automatically generate a more refined design of the application instead of given to the developer.

2.3.5 Framework Specific Modeling Languages

In (Antkiewicz, 2007; Antkiewicz, Czarnecki, & Stephan, 2009) Antkiewicz et al. propose a technique for the creation of frameworks specific modeling languages. These languages are explicit representations of the concepts provided by frameworks and they are used for expressing framework specific models of applications code. These models describe instances of framework provided concepts that are implemented in the application code.

For each of the frameworks discussed in this work a detailed study is performed leading to the creation of its specific framework modeling language. These modeling languages are defined as cardinality-based feature models where a feature represent a distinguishing characteristic among framework concepts instances. These languages can be subsequently used for purposes such as generating part of the source code of an application from a model, generating a model from the source code of an application to facilitate it being understood by developers or simplifying learning a new framework for developer since they can analyze the language instead of the framework itself.

These studies are particularly relevant to the work presented in this thesis for several reasons:

- Again, they emphasize the problems arising from the use of development frameworks. Frameworks are difficult to understand and use, even when sample applications are available, and tools are needed to helps developers in these tasks.
- They demonstrate that model-driven engineering techniques can be successfully used in combination with frameworks to simplify the use of frameworks. In these works a meta-models are created, models based on them are defined, and various transformations are performed involving these models.
- They demonstrate that cardinality-based feature models can be used successfully in combination with development frameworks. Both the meta-models and the models used in this work are represented using this variability management technique.

All these aspects pose similarities with the work done in this thesis. In this thesis model-driven engineering techniques are also used to simplify the use of frameworks and cardinality-based feature models are also used to represent frameworks information. However, there are three significant differences between these works and this thesis.

First, Antkiewicz et al. propose a specific language for each framework. This approach has the advantage that the modeling language is perfectly adapted to the characteristics of each framework, which makes it ideal for learning the framework or for code generation. However, this approach is not the most appropriate in the context in which this thesis is developed, where a high number of frameworks is used and they evolve quickly. Which is why, in this thesis, a single modeling language is used supporting several frameworks.

Second, the level of abstraction used in these studies is low. The proposed modeling languages are defined at an abstraction level very close to the application code. Again, this approach is ideal if one's goal is to automatically generate the source code of the application from the model or to generate the model from the source code. However, this approach is not the most appropriate in the context in which this thesis is developed, where a high abstraction level design of an application has to be adapted by the architect to the frameworks that will be used during development.

And third, in these studies no attention is paid to the integration of multiple frameworks in the same project. This is one of the most common situations in the context of this thesis and therefore one of the issues addressed by it.

2.3.6 Discussion

This section has detailed the most relevant works in the area of development frameworks. The study of these works helps to highlight the problems affecting development frameworks and the lacks in this research area.

One of the most important shortcomings in this area is the lack of a systematic methodology for the integration of multiple frameworks in the same project. This is one of the problems addressed in this thesis, along with others like the high number of frameworks available, making it difficult choosing the optimum for each case, and their high rate of evolution, that makes techniques or tools that are not originally designed with this situation in mind become obsolete very quickly.

2.4 Variability management and architectural decisions

Besides the above mentioned research areas, there are other works that face some of the problems that led to this thesis. Specifically, this section will review some of the most relevant works in the areas of variability management and architectural decisions focusing on how these works can be applied to solve the problems mentioned in Section 2.1.

2.4.1 Variability management

Besides the use of models, the other main feature of the work presented in this thesis is the management of multi-layer architectures variability. As mentioned in Section 1.3, these architectures support a high degree of variability in terms of the variety of development frameworks that can be used and their volatility and in terms of the number of layers and design patterns that can compose them.

There is an area of software engineering that has devoted significant effort to variability management for quite some time, Product Lines (PL). Product Lines are a set of software systems with a similar purpose which share a set of features (Bosch, 2000). The scope of a product line is defined as the set of systems that it is made up of (Clements, 2001). In order to represent this scope, one must identify the common elements shared by all of the product line's systems, the core assets, and how such assets may vary, i.e., their variability. In the domain engineering phase of a product line, a series of re-usable artifacts are created and maintained. These artifacts are used during the application engineering phase to build an application (Bosch, 2000; Pohl, Böckle, & van der Linden, 2005).

There are numerous works presenting different mechanisms for variability management (Chen & Babar, 2011), even using techniques non directly related with product lines such as ontologies (Mohan & Ramesh, 2003). In ArchLayer, product line techniques are used to represent the architectural variability of multi-layer framework based applications due to its widespread use and maturity. The peculiarity of this approach is that the core assets defined are not functional elements of the applications but architectural features.

Specifically, within the area of product lines, this thesis uses feature models. Feature modeling was first introduced in (Kang, Cohen, Hess, Novak, & Peterson, 1990) as a method for discovering and representing commonalities among related software systems. A feature model represents the information of all possible products of a software product line in terms of features and relationships among them (Benavides, Segura, & Cortés, 2010).

There are many techniques for feature modeling (Schobbens, Heymans, Trigaux,

& Bontemps, 2007). This work specifically uses Cardinality-Based Feature Modeling (Czarnecki, Helsen, & Eisenecker, 2005a). The choice of this technique is based on the following reasons. First, because this technique was created with staged configuration in mind, where different product configuration choices can be done in different stages of the development (Czarnecki, Helsen, & Eisenecker, 2005b). In ArchLayer a staged configuration of a feature model is done. Second, because it has support tools for the definition and configuration of feature models (Antkiewicz & Czarnecki, 2004). Particularly interesting for this work is the fact that these tools are built using EMF which facilitates their integration with MDE techniques as the one presented here. And third, because this technique has already been successfully used in combination with development frameworks as the one discussed in this work (Antkiewicz et al., 2009).

Finally, in this thesis feature models are used as a important part of a complex MDE process. In this process different configurations of a feature model (where a different set of features has been selected) are treated as model based on a meta-model. The meta-model in which these configuration are based is the feature model. The idea of using a feature model as the meta-model for its own configurations has already been used in previous works. Namely, in (Gómez & Ramos, 2010) Gomez and Ramos present a tool to transform Cardinality-Based Feature model into EMF meta-model that can be used to instantiate configuration of the feature model as EMF models. Although this work shares the same perspective on feature models as this thesis, the tool provided has not been used here since the tools presented in (Antkiewicz & Czarnecki, 2004) were more mature and seamlessly integrated into a MDE process.

2.4.2 Architectural decisions

The architectural decisions made in the development of a complex system always have important consequences (Kazman, Asundi, & Klein, 2001). For this reason, capturing and documenting these decisions is of vital importance in many cases (Harrison, Avgeriou, & Zdun, 2007), especially when a project will be maintained over time or when several similar projects will be developed.

In this area, there are numerous tools and techniques to manage architectural decisions (Tang, Avgeriou, Jansen, Capilla, & Babar, 2010). Particularly stand out for their close relationship with this thesis two works of Zimmerman(Zimmermann, 2011, 2012). They present a framework for the identification and modeling of recurring architectural decisions, and for converting those decisions into design guidelines for future development projects.

In particular, Zimmerman proposes seven identification rules (IRs) with which to identify potential recurring decisions. These rules have their direct counterpart in this work. For example, Zimmerman's IR1 corresponds to executive decisions, whose counterpart in this work is the possibility of using different feature models for different types of projects or situations. Similarly the IRs relating to the selection and use of design patterns, technologies, and manufacturers have their clear counterparts in the feature model structure.

The main difference between this work and that of Zimmerman is the use made of those architectural decisions. In the case of Zimmermann's work, the main objective is to gather information for its use in future projects. The focus in this work is on using that information to simplify the process of obtaining an specific design of the application on which architectural decisions are made.

Additionally, it is particularly relevant to this thesis the work presented in (Babar, 2004). In this work a set of templates are defined to capture architecturally important information about design patterns and document it in such a format, which makes it readily usable during software architecture design and evaluation activities. A slightly modified version of these templates are used in this thesis to document the effects that technological or architectural decisions can have over the final system. However, this information is not enough to describe how different technologies may affect the system. The proposed templates only allows to document individual elements and, in many cases, the use of two technologies together affects the system differently of how they would do it if the technologies were used independently. This happens because the communication between two technologies is generally not straightforward or simple, but requires additional mechanisms that may affect the system. Therefore, this information should also be recorded and a new template is used in this thesis for such purpose.

2.4.3 Discussion

This section has described the works forming the basis on which this thesis rests. Specifically, works on two research areas has been detailed, variability management, and architectural decisions.

The work presented in this thesis is situated within these areas and therefore it uses previous works in them. One of the principles followed when developing this thesis has been to utilize, to the extent possible, existing techniques that could be applied to the context in which this work is developed and avoid duplication of effort. The works reviewed in this section show the results of applying this principle.

2.5 Conclusions

In this chapter the most relevant work for the development of this thesis have been reviewed.

First, the most relevant work in the area of model-driven web engineering have been reviewed. This is a very close area in which techniques are proposed for the development of applications with architectures and technologies very similar to those used in this thesis. Specifically, special attention has been paid to the proposed models and processes for the development of web applications.

Next, several works have been analyzed in the area of using development frameworks. The work analyzed were those that are closest to the work done in this thesis and pose similar problems to those described in this thesis with the use of frameworks.

Finally, those works that have served as the basis for the proposals presented in this thesis have been reviewed. Close attention has been paid to the two areas where more knowledge has been borrowed. Specifically, variability management, and architectural decisions.

From this review of the state of the art it can be concluded that, as far as the author knows, there is no work that is placed in the same context as this thesis and the following weakness exist in the area that could be resolved with the work proposed in this thesis:

- Related to the complexity of multi-layer architectures:
  - There are several solutions for the development of web applications using multi-layer architectures. However, in general, they do not permit to manage the architecture and technologies with which applications are going to be implemented.
  - The complexity of web applications keep growing. Similarly, the complexity of the software architectures used to develop them is also increasing. However, there are no solutions to simplify architectural decisions making in such developments.
- Related to the quick evolution of development framework:
  - Parallel to the increasing complexity of web applications, the techniques and technologies used for its development evolve. However, there are no solutions that are oriented from conception to support the fast pace of technological evolution.
  - The use of model-driven techniques reduces the complexity of using frameworks. However, there are no common solutions that can be used for a large number of frameworks.
  - In the large majority of web applications developed today, multiple development frameworks are used together. However, there are no solutions that address the problem of integrating multiple frameworks.
- Related to the high staff rotation in distributed development centers:
  - A significant portion of multi-layer applications are developed in distributed development centers with staff rotation. However, there are no solutions that address these issues in a systematic way.

ArchLayer aims to address these problems found in the related works. The next chapter details how the proposed process solve these issues.

Chapter 3

ArchLayer: Bridging the gap between design and implementation

He's not a killer. He just wins... Thoroughly.

Ender's Game, Orson Scott Card.

As seen in the previous chapter, a number of problems working with multi-layer architectures and development frameworks have not been thoroughly addressed by the research community. Yet these problems are very relevant in this kind of development, especially if they are done in a context as the one this thesis is focused on, distributed development centers with high staff rotation. For these reasons ArchLayer, a development process for multi-layer framework based application, is presented in this dissertation. The main goal of this process is to obtain a multi-layer framework based application from an initial design completely independent of the architecture or technologies that will be used in the implementation, pursuing at any time to reduce the need for developers and architects to rely on deep technical knowledge of the various technologies.

This chapter describes ArchLayer in detail. To make the process easier to understand a simple sample application is used throughout the chapter to demonstrate the various steps of the process. The description of ArchLayer is organized as follows. In Section 3.1 a detailed description of the proposed process is presented. Section 3.2 is focused on the characteristic of the initial design of the applications that is used as starting point of the development process. Section 3.3 describes in detail the architectural decisions repository and its use throughout the process. Section 3.4 explains the proposed meta-model to contain specific information about each framework. In Section 3.5 the model transformations proposed to tailor the initial design of the application to a design specific to the architectural choices are detailed. Finally, Section 3.6 summarize this chapter focusing on the solutions that ArchLayer presents to address the previously identified problems.

3.1 ArchLayer overview

As stated above, this section contains a detailed overview of ArchLayer. Figure 3.1 shows a diagram with the different activities and artifacts involved in the process. Rectangular shapes represent the different artifacts involved in the process, while rounded shapes represent the activities or steps to be performed. Blue colored shapes indicate artifacts or activities to be created or performed by the architect or the developers of the application. Green colored shapes represent automatically generated artifacts or automated activities.

The main goal of ArchLayer is to obtain a multi-layer framework based application from an initial design obtained from the application's requirements. This initial design is completely independent of the layers, design patterns and frameworks to be used in the implementation. During the process a design tailored to the architectural decisions made by the architect is obtained semi-automatically and used as the basis for development. As can be seen in the figure, this is a complex process comprising several steps in which multiple artifacts are involved. The different activities showed in the figure are numbered in the order in which they will be described here.

The figure shows how ArchLayer begins with two artifacts: an initial design of the application to be developed and the architectural decisions repository. The whole process proposed here revolves around these two elements.

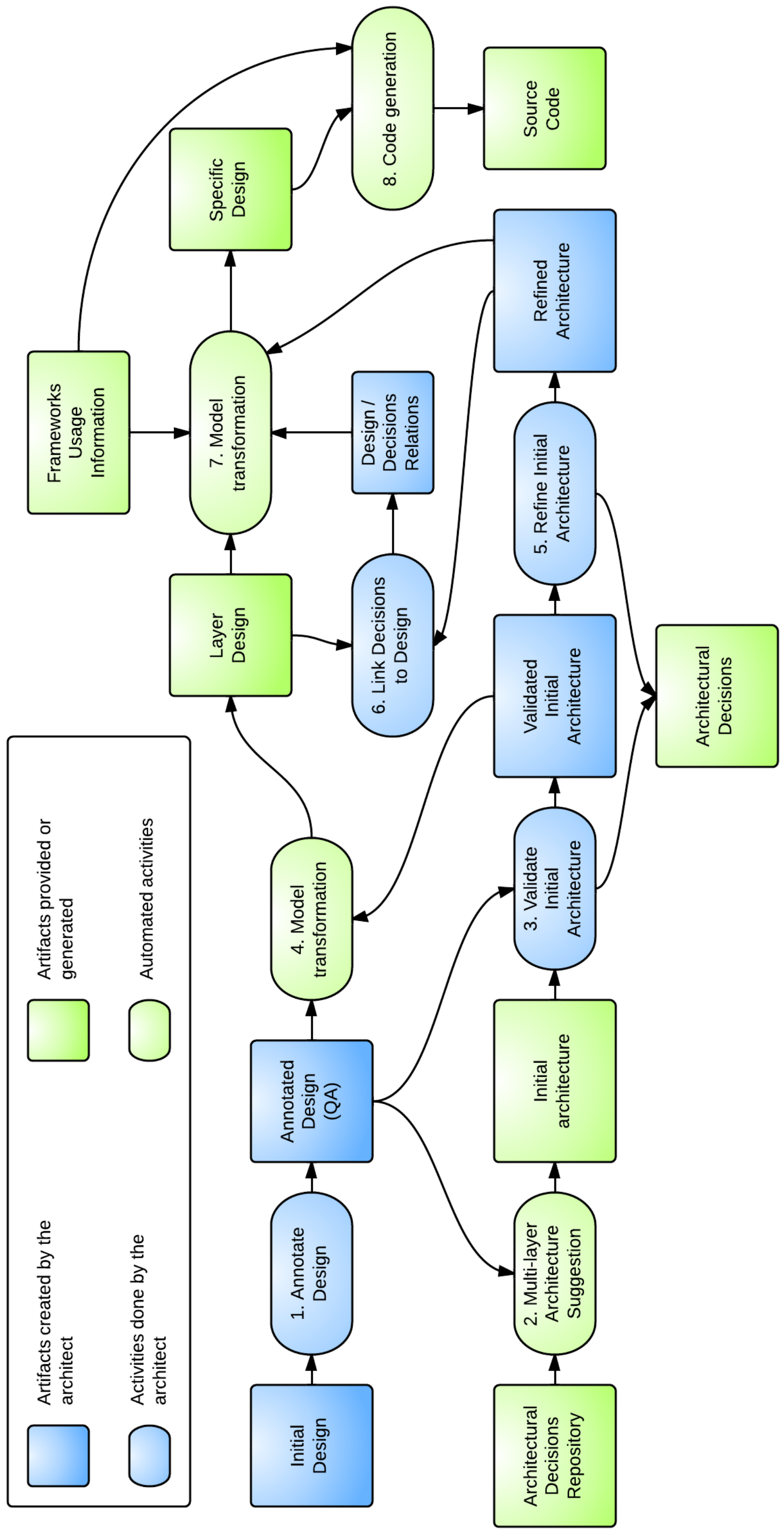

Artifacts created by the architect
Artifacts provided or generated
Activities done by the architect
Automated activities
Initial Design
1. Annotate Design
Annotated Design (QA)
4. Model transformation
Layer Design
7. Model transformation
Frameworks Usage Information
Specific Design
8. Code generation
Source Code
6. Link Decisions to Design
Design / Decisions Relations
Architectural Decisions Repository
2. Multi-layer Architecture Suggestion
Initial architecture
3. Validate Initial Architecture
Validated Initial Architecture
5. Refine Initial Architecture
Refined Architecture
Architectural Decisions

Figure 3.1: Archlayer, the process proposed for the development of multi-layer applications.

The initial design represent the requirements of the application to be developed. As in most of the web engineering techniques analyzed in Section 2.2.2, the initial design consist of a use case diagram followed by several activity diagrams that provide insight into the behavior of the use cases. This initial design can be refined in order to include information about the quality attributes that the system developed must meet. These quality attributes add to the design information about the system's non-functional requirements. This corresponds to activity diagram *1: Mark design.*

Starting with the initial design including the quality attributes, the architect should take the first architectural decisions and design the initial software architecture. To this end, the architectural decisions repository begins to be used. In order to facilitate this step, an initial architecture is automatically suggested to the architect, represented by activity diagram *2: Multi-layer Architecture Suggestion.* This initial architecture comprises the layers into which the application will be organized. It is the architect's responsibility to validate these suggested layers or to change them if deemed appropriate. Process step *3: Validate Initial Architecture.*

Then, a first model transformation is performed using the information contained in the initial architecture. This transformation converts the initial design into a design tailored to the layers in which the application is to be decomposed. Process step *4: Model transformation.*

Once this design has been completed, the architect continues with the design of the architecture by choosing the design patterns to use in the development of each layer and the frameworks that support them. Process step *5: Refine Initial Architecture.*

Possibly the architect may choose to use multiple design patterns and frameworks for a layer's implementation. If so, the architect should specify, for each task of the layer in question, the design pattern or framework to be used. Process step *6: Link decisions to design.*

Once the architect has this information, the next stage consists of a model transformation that converts the layered design of the application into a specific design for the design patterns and technologies selected by the architect. In order to perform this transformation, additional information about the design patterns and frameworks must be used. This information is contained in a model that describe the characteristics of frameworks and design patterns. Process step *7. Model transformation.*

Once the architect has the final design, he or she should validate it and modify it if necessary. Once validated, the final design can be used either to start a traditional development process or as a basis for a model-to-text transformation that will generate a significant part of the project's source code. Process step *8. Code generation.*

The following subsection detail the requirements of an application that will be used to exemplify ArchLayer. Later, a detailed explanation of each activity of the process applied to the example will be discussed.

3.1.1 Running example

This section describes the requirements of the application to be used to illustrate ArchLayer. Specifically, the development of an e-commerce application will be used.

The application to be developed should have the following features:

- The developed system should be able to get orders through a web portal. The orders should be validated, the availability of the ordered products should be checked, the payment of the order should be managed and products related to the ones ordered should be offered to the customer.
- The system should support the preparation and shipment of the different orders. The preparation of an order should be managed from the moment it is formalized in the system until it is ready for shipment.
- The system should be integrated with a legacy applications used to manage the stock in the warehouse and to interact with the shipment company. The integration should cover the complete period since an order is checked until it is received by the customer.

Additionally, the developed application should consider the following non-functional requirements.

- The system should be easy to adapt. Ecommerce evolves rapidly so the application should be easy to modify in order to stay current.
- The access to customers data should be secure. The system uses personal and banking information from customer so the access to it should be protected.
- The system should be easy to use.
- Orders placed in the system should be validated quickly. Customers are usually impatient so the validation process of an order should be made as quickly as possible.

These requirements are used throughout this chapter to illustrate ArchLayer and its contributions.

3.1.2 Marking the initial design with quality attributes

As mentioned above, ArchLayer begins with the initial design of the application to be developed. This initial design contains a high level description of the system to be built based on its functional requirements. A common way to represent this initial design is by using use case diagrams and activity diagrams. This technique is used by many of the proposals discussed in Section 2.2.2 and also in this work. Figure 3.2 shows a use case diagram representing the example application discussed above. The application has been decomposed in five use cases. Each of the use cases can be defined in greater depth by using an activity diagram. Figure 3.3 show the acivity diagram of the *Check Order* use case.

The *Check Order* use case is responsibility of the seller. Each time an order is checked its information is retrieved from the datastore where it was stored when the customer validated it. Then all the information of the order is checked including the customer data, the ordered products data and the payment data. Once all the information is checked the order is saved again in the same datastore reflecting its new state. At the same time, additional products similar to the ones ordered are

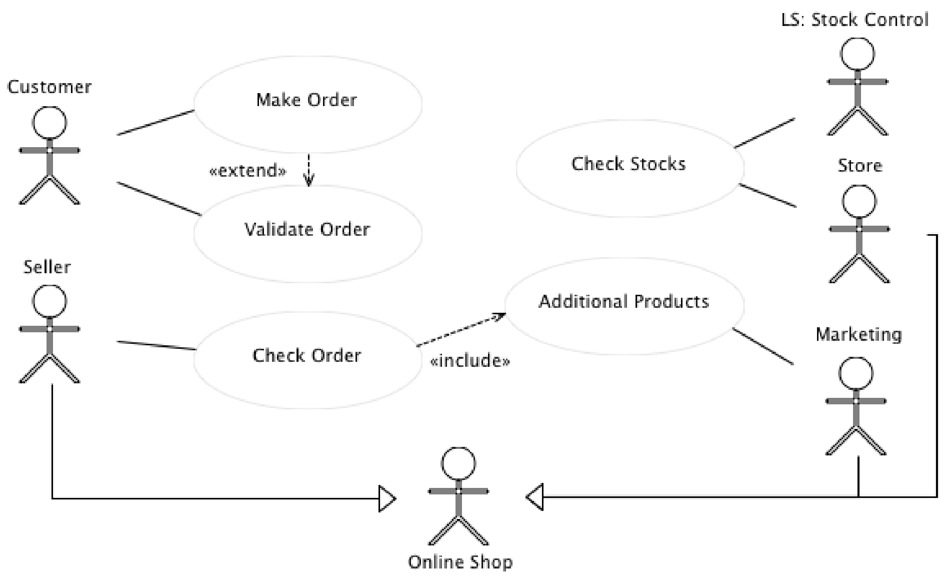


Figure 3.2: Use case diagram of the example application

searched to be offered to the customer later. This search is performed in a different use case so in the diagram is represented by a different type of action. The products retrieved by the search are saved in a datastore. Finally, the checked order and the associated additional products are retrieved from the datastores and sent to the customer.

As mentioned above, this way of representing the initial design of an application is widely accepted. However, it has some drawbacks. The information contained in the diagrams is not enough for the architect to select the system architecture. The architecture should not only support the functional requirements but also take into account the quality attributes to be met by the system as well as the relations between them and the functional requirements.

For this reason the first step of ArchLayer involves adding information about quality attributes to the initial design of the application. To do so we propose the use of a technical approach. This technique employs a requirements model marked with the relationships and constraints imposed by the quality attributes. In this way, the architect is offered a clear and concise view of the information about the requirements and their relationships. Additionally, the marks provide information that will be used later in the process to detect the layers and architectural patterns that best

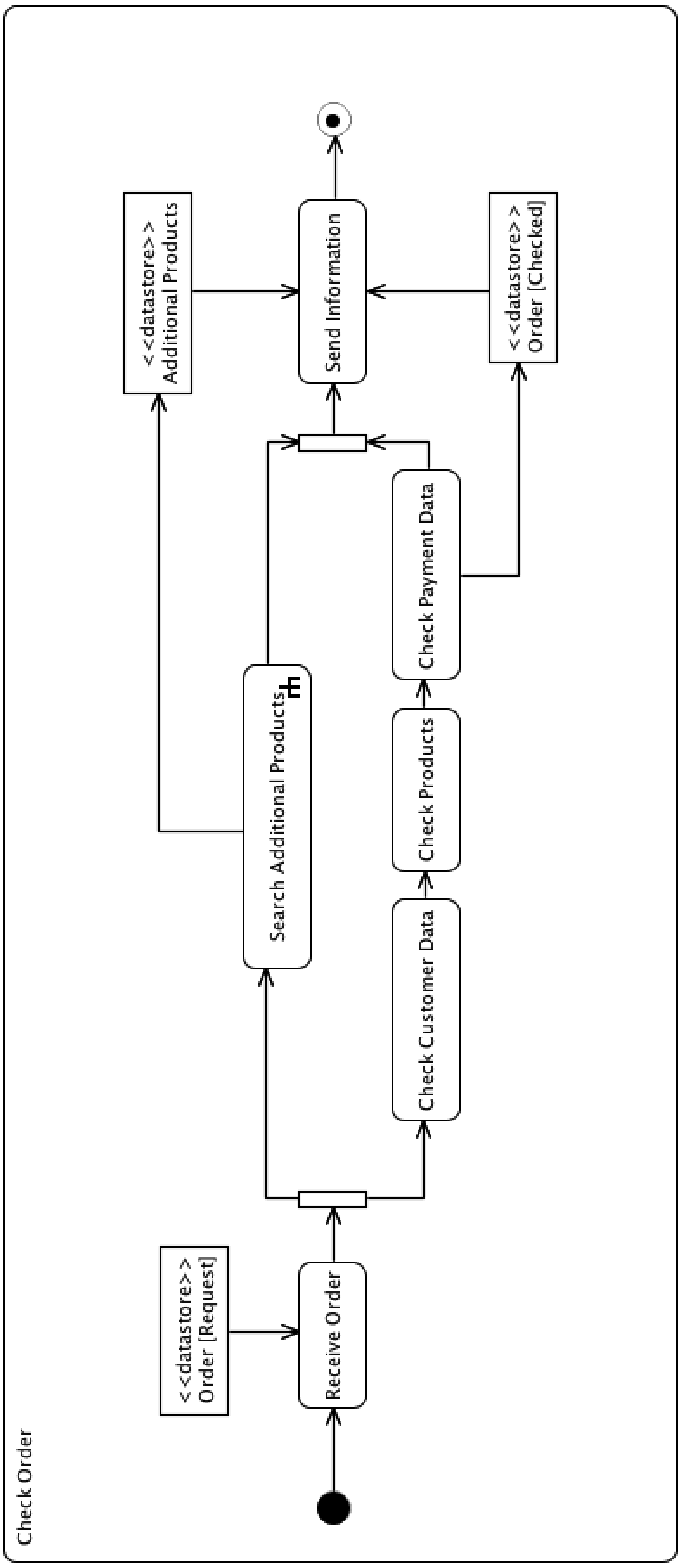

Check Order
<<datastore>> Order [Request]
Receive Order
Check Customer Data
Check Products
Check Payment Data
Search Additional Products
<<datastore>> Additional Products
Send Information
<<datastore>> Order [Checked]

Figure 3.3: Activity diagram of the *Check Order* use case

suit the application. A detailed description of this technique will be presented in Section 3.2.

Hence, applying the marking technique to the diagrams presented in figure 3.2, a marked diagram is obtained with the additional information about the quality attributes of the system that is necessary for a good architectural design. Figure 3.4 shows how the use case diagram is left once the marks have been added and the quality attributes affecting each of the use cases.

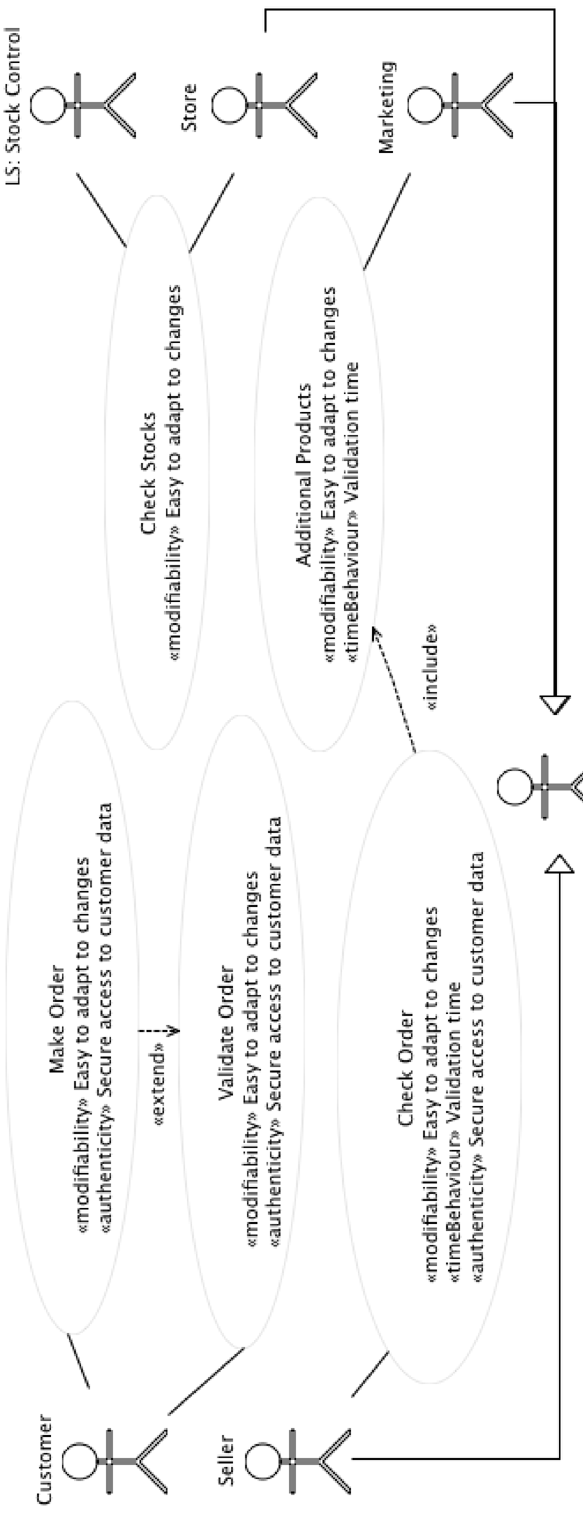
Figure 3.5 shows the activity diagram describing the *Check Order* use case annotated with the marks representing the quality attributes affecting the different actions in the diagram. The figure includes marks that affect the entire use case, like the modifiability mark, and therefore encompass all the actions in the diagram.

Other, more specific marks, affect only a single action or a reduced set of actions. For example, the authenticity mark only affects the *Check Payment Data* action, since this is the only action in which the customer payment data are processed, or the timeBehaviour mark encompass the three actions related with the processing of the order data, meeting the non functional requirements of the application.

By marking the initial design of the application with the quality attributes that must be met by the application a model is obtained that contains all the information needed to design the most appropriate software architecture for implementation. This model is used as the basis for the rest of the process described in this thesis.

3.1.3 Modeling architectural variability

The other key element in the ArchLayer's execution is the architectural decisions repository. This repository captures the architectural and technological variability of framework based multi-layer applications. Thanks to its use throughout various steps of the process, the architect is assisted in the work of transforming the initial design of the application into a design tailored to a specific architecture and technology. In particular, this work is simplified without limiting the architect's choice of possibilities in the selection of the design patterns or technologies that will be used in the development process.



LS: Stock Control
Store
Marketing
Check Stocks
«modifiability» Easy to adapt to changes
Additional Products
«modifiability» Easy to adapt to changes
«timeBehaviour» Validation time
«include»
Online Shop
Make Order
«modifiability» Easy to adapt to changes
«authenticity» Secure access to customer data
«extend»
Validate Order
«modifiability» Easy to adapt to changes
«authenticity» Secure access to customer data
Check Order
«modifiability» Easy to adapt to changes
«timeBehaviour» Validation time
«authenticity» Secure access to customer data
Customer
Seller

Figure 3.4: Annotated use case diagram of the example application

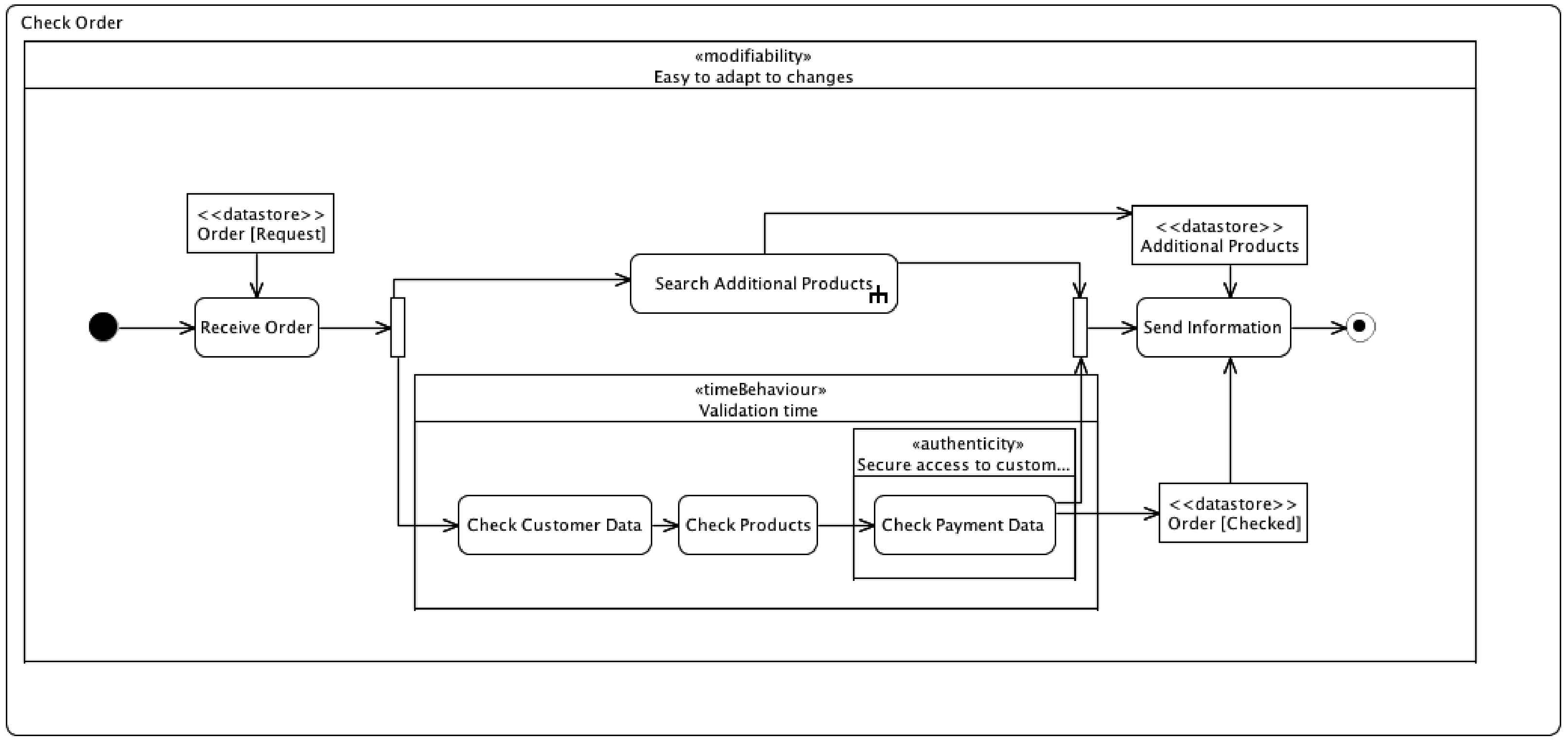


Figure 3.5: Annotated activity diagram of the *Check Order* use case

The repository consists of a feature model. A detailed description of the process followed in creating this repository and the reasons behind the selection of feature modeling techniques will be presented in Section 3.3.

The repository is structured into hierarchical levels. Each of these levels represents a level of abstraction in a multi-layer architecture. The first level encompasses the layers that may be included in the application.

Each supported layer contains the design patterns or techniques commonly used for its implementation in the second hierarchical level of the feature model. The model's third level details the technological options available for implementing a particular design pattern or technique. Sometimes the model contains a fourth hierarchical level that contains the different ways to use a technology.

This repository is used throughout the entire ArchLayer process. A configuration of the feature model will be created in different steps and used to help the architect in tailoring the initial design of the application to the architecture chosen.

3.1.4 Choosing the application layers

The next step in the process of converting the initial design into a specific design for the chosen architecture and technologies involves selecting the layers that will conform the application. In order to assist the architect in this task, a set of model transformations automatically suggests an appropriate set of layers. These transformation take as input the application's requirements that have been marked with the quality attributes as described above and the architectural decisions repository. The output of the transformation is a partial configuration of the feature model in which the suggested layers have been selected. A detailed description of these transformation will be presented in Section 3.5.

Starting from the marked use case diagram presented in figure 3.4, an initial choice of layers is offered to the architect. Figure 3.6 shows the suggested selection of layers based on the marked use case diagram.

The suggestion given to the architect is the use of five layers for the application's

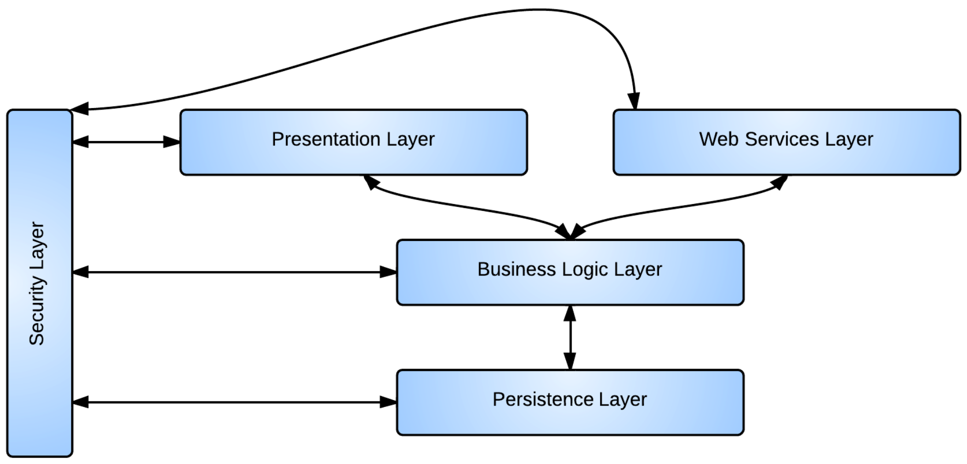


Figure 3.6: Suggested layers

development. These layers are the following:

1. Persistence. The execution of the use case requires the retrieval of data that has been recorded by other system use cases (such as listing an order information in the *Check Order* use case) and the storage of data for later use (such as the additional products information). The transformation recommends that the architect include a persistence layer containing all the elements related to information persistence.

2. Business logic. This is a standard layer in multi-layer applications which includes elements related to the application's behaviour.

3. Presentation. Some of the use cases are executed by human users, therefore the process recommends that the architect include a presentation layer. This layer includes all the elements related to user interaction.

4. Web services. The *Check Stocks* use case is carried out by an external system. To support the integration with such systems, the transformations recommends that the architect include a Web services layer that encapsulates communication facilities with external systems.

5. Security. Checking the payment data of the customer has to be a secure task. Therefore the transformations recommends that the architect include a security layer which combines all the elements related to this aspect of the

system.

It can be noted from the above descriptions of the suggested layers that the layer suggestion process is based on a relatively simple rule set. In particular, a layer is suggested on the basis of two criteria.

The first criterion considers specific features found in the initial design of the application. Their presence will determine whether a layer should be suggested to the architect as part of the application and it can be detected through simply querying the design model. For example, the process suggests a presentation layer when a human actor is interacting with a use case.

The second criterion is based on the marks that contain information about quality attributes. The process proposes the use of a layer when the design has a quality attribute mark suggesting it. The process detects the presence of these marks through querying the design model. For example, if the process finds a mark stating that an activity must support authenticity it will suggest a secure layer.

Once the transformations have provided the suggested set of layers, the architect must validate it or modify it if appropriate. For the example shown in figure 3.6, the architect may suggest including a test layer which had not been suggested initially because no quality attribute mark had indicated that it was necessary. The architect may also choose to eliminate the security layer. If the activity represented by this use case happened to be the only operation of the entire system needing authenticity, it would not be necessary to add a layer to support a single operation, and the elements related with this activity would be included in the existing layers. For the example in the following sections it is considered that the architect accepts the layers suggested by the transformations without change.

3.1.5 Tailoring the design to a multi-layer architecture

At this point, a model transformation is performed, taking the initial design marked with the quality attributes as the source model. The feature model's first level configuration is taken as an additional source of information for the transformation. Again, a detailed description of these model transformations will be provided in

Section 3.5.

After applying the transformations, the architect obtains a design adapted to the layers specified in the feature model's configuration. Figure 3.7 shows the result of applying this transformation to the marked design presented in figure 3.5.

This new model contains the layers chosen by the architect for the application's development process. Each action is represented in all the layers in which it is involved. For example, the action responsible for checking the customer payment data is represented in the business logic and persistence layers. These are the layers in which that action should be developed based on the information found in the marked design and the feature model.

The criteria used for the transformation are similar to those presented above for the layer suggestion process. The transformation creates the number of layers that the architect has selected in the feature model. By default, the main flow of actions is placed in the business logic layer which controls the process execution. If a presentation layer exist, the start and end action are placed in it in order to grant the user control over the activity's execution. Each action is included in all those layers in which it is present. To determine which action are present in a layer, similar criteria to the ones used to select the layers are applied. Actions related to a layer that has been deleted by the architect are re-positioned in the business logic layer. The layers added manually by the architect are created without any action.

The new model, adapted to the chosen layers, is very interesting for architects because it allows them to observe the distribution of actions in each of the layers of the application. However, it provides little information about the actual design of the application for a specific architecture and set of technologies. Its main utility is to establish a basis for the subsequent transformations that will be detailed in the following sections.

3.1.6 Choosing design patterns and technologies

At this point in ArchLayer, the architect has a model of the application to develop which is adapted to the layers into which he or she decided to split the application.

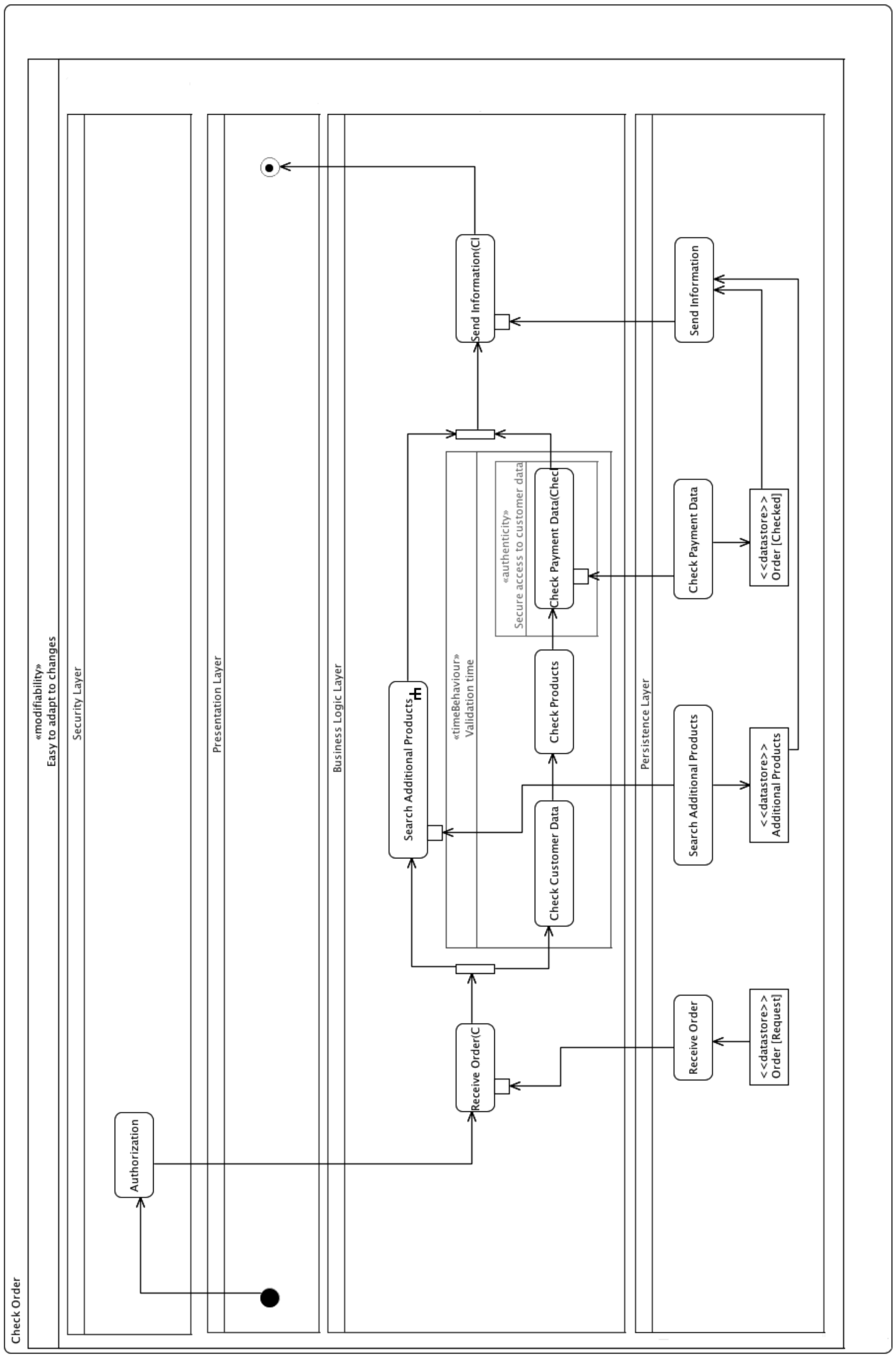


Figure 3.7: Layered activity diagram

The next step of the process involves choosing the design patterns and technologies that will be used in the system's development. For this task, the architect has available the architectural decisions repository.

Similarly to the case when the layers comprising the application were selected, a model transformation is used to provide a series of recommendations to the architect regarding the design patterns and technologies that could be used. To make appropriate suggestions the transformations need to have in the repository information about the quality attributes affected positively or negatively by each design pattern and development framework. A detailed description of how the architectural decisions repository is enriched with this information will be presented in Section 3.3.

In this step, however, the decisions made by the architect are much more important. In many cases the choice of a particular technology or pattern does not depend on the requirements of the application or the quality attributes of the system, but on the company's internal criteria. For example, one of the most important factors when it comes to choosing a technology to implement the MVC pattern in the presentation layer is the developers' experience. There are many frameworks that fully cover this role. Given that whichever one is chosen will continue to satisfy the application's requirements, the one in which the development team has most experience is usually picked.

Whatever the case, the transformations provided supports a rule set that allows it to offer an initial selection to the architect. These are simple rules which may express the company's preferences or other criteria for technology selection.

As mentioned above, the burden of this task has to be borne by the architect. The task is done in two steps. First, the architect selects the design patterns that will be used in the development of each layer. Figure 3.8 shows a possible selection of design patterns to be used in the process of building the presentation and Web services layer of the example application. In this case, the architect decided to use the classic MVC pattern together with the WebRemoting pattern for the presentation layer. For the Web services layer, the architect decided to use the REST protocol (Webber,

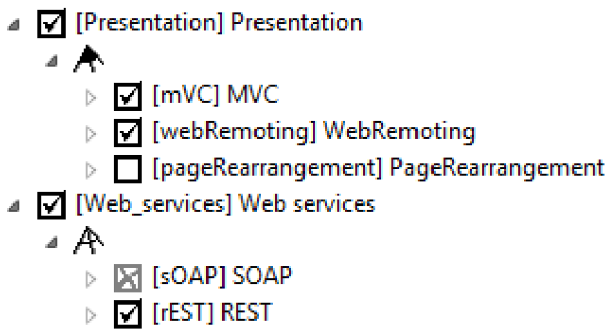
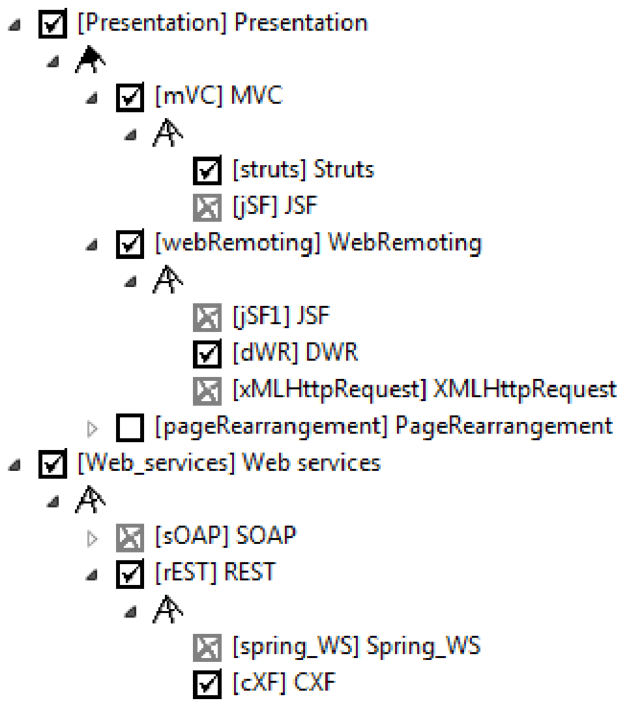

[Presentation] Presentation
[mVC] MVC
[webRemoting] WebRemoting
[pageRearrangement] PageRearrangement
[Web_services] Web services
[sOAP] SOAP
[rEST] REST

Figure 3.8: Excerpt of the configuration of the feature model with the selected design patterns.

Parastatidis, & Robinson, 2010) instead of SOAP (Snell, Tidwell, & Kulchenko, 2001).

In the second step, the architect should select the technology that will support the implementation of the chosen patterns. This will simplify the architect's work when it comes to deciding which technologies to use. The architect will not have to choose a framework for the application's web services from among all those that are available. Instead the selection will be made only from among those which support the REST protocol. Figure 3.9 shows a possible technology selection for the design patterns chosen previously. Here the architect has chosen the Struts (*Struts development framework*, n.d.) framework to support the MVC design pattern, DWR (*DWR development framework*, n.d.) to support the WebRemoting pattern, and CXF (*CXF development framework*, n.d.) as the technology used for the REST web services.

Sometimes the architect must take an additional step in this task. This is because a given technology may provide different ways to implement a particular design pattern. In such cases, ArchLayer does not limit the architect's possibilities, but allows him or her to choose whichever method is considered most appropriate. Figure 3.10 shows an example of such a situation. The Hibernate development framework allows the DAO pattern to be implemented using three alternative techniques: with proprietary Java annotations, with XML configuration files, or in a way that complies with the JPA standard. In the feature model shown in figure 3.10, the architect decided to use Java annotations in the Hibernate framework to implement the DAO



```
◢ ☑ [Presentation] Presentation
    ◢ 
        ◢ ☑ [mVC] MVC
            ◢ 
                ☑ [struts] Struts
                ☒ [jSF] JSF
        ◢ ☑ [webRemoting] WebRemoting
            ◢ 
                ☒ [jSF1] JSF
                ☑ [dWR] DWR
                ☒ [xMLHttpRequest] XMLHttpRequest
        ▷ ☐ [pageRearrangement] PageRearrangement
◢ ☑ [Web_services] Web services
    ◢ 
        ▷ ☒ [sOAP] SOAP
        ◢ ☑ [rEST] REST
            ◢ 
                ☒ [spring_WS] Spring_WS
                ☑ [cXF] CXF
```

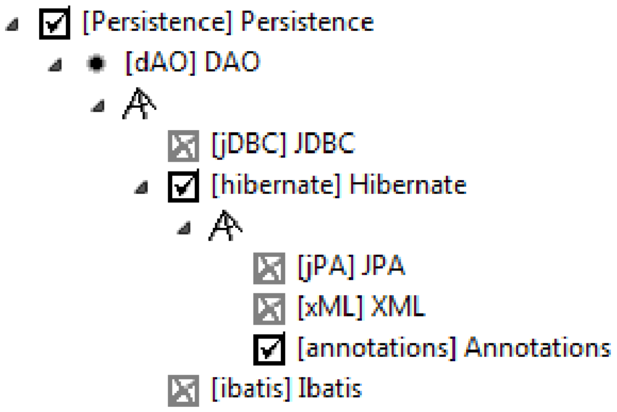
Figure 3.9: Excerpt of the configuration of the feature model with the selected technologies.

design pattern for the application's persistence layer.

For clarity reason, in this description of ArchLayer the architectural decisions are taken in a specific hierarchical order. However, the architect could choose a different order, for example by choosing the frameworks that will be used during the development before choosing the design patterns. The use of feature models to specify architectural knowledge and the architectural decisions make it possible whilst the consistence is kept during the transformation process.

3.1.7 Relating the chosen architecture and the system model

As stated in the previous section, it is possible to develop an application using any of several design patterns and development frameworks for the implementation of a single layer. Figure 3.9 shows an example in which the architect has decided to use two different design patterns (each supported by a development framework) to implement the presentation layer of an application. When this is the case, an additional step has to be inserted into ArchLayer.



- ✓ [Persistence] Persistence
  - ● [dAO] DAO
    - ☒ [jDBC] JDBC
    - ✓ [hibernate] Hibernate
      - ☒ [jPA] JPA
      - ☒ [xML] XML
      - ✓ [annotations] Annotations
    - ☒ [ibatis] Ibatis

Figure 3.10: Excerpt of the configuration of the feature model with the selected technology usage technique.

This additional step will be responsible for relating the actions within the layer to the design pattern or framework to be used for its development. Whenever a single pattern or technology is used to implement a layer, ArchLayer will by default relate all the actions in that layer to the chosen pattern or technology.

Figure 3.7 shows four tasks involved in the persistence layer. If the software architect selects more than one framework for the implementation of this layer, he or she would also have to indicate which of these actions would have to be implemented using each of the selected frameworks.

To perform this task bindings, i.e., relationships, should be established between an element of the layered design and a framework in the architectural decisions repository. These bindings are expressed in the form of annotations in the layered design indicating the design pattern and framework to be used in the implementation of a given action.

3.1.8 Additional technological information

At this point in the process the architect knows the design patterns and technologies that will be used in the development process. Also, thanks to the fine-grained configuration detailed in the previous section, the architect knows which specific patterns and technologies are to be used for each action in the layered model of the application. This information is not enough, however, to perform the last step of

ArchLayer and obtain a specific model for the chosen technologies.

To perform this step, detailed information on the design patterns and in-depth knowledge of the frameworks that are to be used are required. As noted above, gaining this knowledge and applying it correctly to each development process is an arduous task for architects.

In order to make the framework and pattern knowledge explicitly available for re-use in multiple developments, a meta-model is presented in which the configuration and usage of these technologies is specified. This meta-model allows architects to define the configuration and usage of a large number of frameworks.

To model a framework's behaviour using this meta-model the architect should identify the version of the framework that will be used. In addition, he or she must also specify the dependencies involved in the use of that framework. The configuration files affected by this framework and the code artifacts that will contain the framework's code should also be included. Detailed information about this meta-model will be presented in Section 3.4

In order to use a framework in ArchLayer, the architect will need a model of its behaviour containing the implementation details of each design pattern or technique supported by that framework. These models will not be created by the architect, they will be created by experts in a particular framework and later re-used by other architects or teams less experienced. The models of all the frameworks used in the running example are provided by the author. More information about these models can be found in Appendix D.

3.1.9 From multi-layer to framework-based

At this point, all the information needed to make the final model transformation is available. The initial model for this transformation will be that of the application split into the layers previously chosen by the architect.

Additional information is applied to the transformation in the form of the configuration of the feature model containing all the technological and architectural decisions taken by the architect. In this configuration, the architect has selected the design patterns that will be used in each layer, the development frameworks that will support each design pattern, and how each framework will be used.

Accompanying this configuration of the feature model, the fine-grained binding that relates model elements from the application's layered design with the architectural and technological features that they should comply with is available. Finally, the process can count on models that describe the behaviour of the selected frameworks.

With all this information, a model transformation is applied to the layered model of the application to obtain a specific model for the architecture and the technologies chosen by the architect.

The criteria followed in applying this transformation are more complex than those of the previous transformation. A transformation is applied to each element of the layered model, and the criteria used to determine which transformation to apply are based on three elements of information. Firstly, the transformation applied depends on the layer in which the activity is located; this information is obtained by querying the model. Secondly, the process will apply the transformation corresponding to the chosen design pattern; this information is located in the fine-grained binding. And thirdly, the process needs to know the technology that will support the design pattern since this will affect the transformation that will be applied. On the basis of these criteria, the appropriate transformation is applied to each activity.

Figure 3.11 shows the result of completing the transformation of the Check user permission activity. The transformation applied by the process corresponds to using the Hibernate framework to implement the DAO pattern with Java annotations.

Applying this transformation to all the activities contained in the intermediate model results in the creation of a specific model for the chosen technologies and patterns. By following ArchLayer to this point, the architect will be provided with assistance in the difficult task of translating the initial design of an enterprise application into a specific design for a multi-layer architecture based on development frameworks. This facilitates the architect's work and reduces the risk of introducing

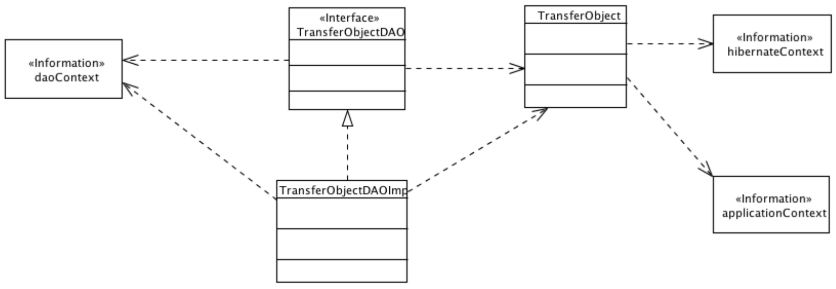


Figure 3.11: Excerpt of the model adapted to the technologies that will be used in the application's development process.

errors in the design of the application. Additionally, ArchLayer handles the re-use of knowledge that is crucial to the development of applications, such as the best way to use each pattern and framework.

3.1.10 Code generation

Finally, the detailed model obtained as a result of the previous transformations can be used to automatically generate part of the source code of the application. This code generation is performed by a model to text transformation that take as input the model tailored to the frameworks chosen by the architect and produces the code of the application integrating all the selected frameworks and the instances of the frameworks concepts needed to implement all the actions in the initial design. More detailed information about JACA, the tool for automatic code generation, will be provided in Chapter 4.

3.2 Marked design

This section describes in detail the approach used to annotate the initial design of an application in order to better support architecture design decision making. The presented approach provides mechanisms to annotate and model requirements relations for helping architects gain a relatively complete picture of functional and non-functional requirements relationships. To annotate the requirements interdependencies, a set of UML profiles for Use Case and Activity Diagrams have been defined.

The information added by the marks is used later in the process for selecting and reasoning about appropriate design patterns and frameworks that can help satisfy both the functional requirements and the quality attributes. In addition, these models can be reused by tools that can assist architects in making architectural design decisions. The following subsection detail the proposed technique for annotating the initial design of the applications.

3.2.1 Annotating the initial dessign

For annotating the requirements relationships, some extensions of the UML diagrams that are commonly used to document requirements were defined. Specifically, profiles have been defined for Use Case and Activity Diagrams. These profiles allow the requirements engineer to annotate which functional requirements, or part of the functional requirements, are affected by quality attributes. Thus, the effort required of the architect to identify the architectural significant requirements and their interdependencies and the chance of making errors during this analysis are both significantly reduced.

These profiles are based on the quality model defined in the ISO/IEC 25010 (SQuaRE) (Organization, 2011). This standard classifies software quality on the basis of a structured set of characteristics and subcharacteristics. Thus, each profile defines a set of stereotypes allowing the software architect or the requirement engineer to annotate the requirement models regarding to this quality model.

Annotating use case diagrams

The profile extending the use case diagram, showed in Figure 3.12, defines the set of stereotypes for modeling the characteristics detailed in the SQuaRE quality model. These stereotypes extends the *ExtensionPoint* metaclass. This class is originally used to document points where the behaviour of a use case is augmented by another use case. The stereotypes extending this metaclass document points where the

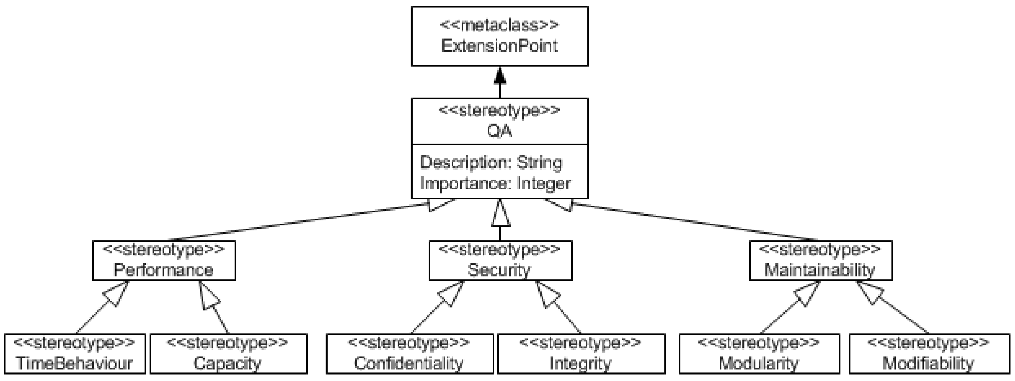


Figure 3.12: Excerpt of the profile for modeling the relationships with quality attributes in use case diagrams.

behaviour of a use case is restricted by a quality attribute. The *ExtensionPoint* class was chosen to represent the relationships because, firstly, it maintains the readability of the diagrams when the use cases are constrained by a large number of quality attributes, and, secondly, one can also document the exact use case points at which the restrictions should be applied.

As Figure 3.12 shows, the *ExtensionPoint* metaclass is extended by the *QA (QualityAttribute)* stereotype. This element defines the general information for the quality attributes, such as the description of the relationships and its importance. This stereotype is subsequently specialized using a hierarchy of stereotypes similar to the hierarchy of quality attributes defined in the SQuaRE standard. The intermediate nodes of the hierarchy represent the characteristics of the quality model, and the leaf nodes represent the subcharacteristics. Thus, depending on how deep each quality attribute is detailed in the system, the requirements engineer can detail the relationships using characteristics or subcharacteristics.

These annotations must be modeled by the requirements engineer, but are very useful to the architect for acquiring all the necessary information on the requirements without having to analyze in depth all the requirement artifacts. With these annotations, the risk of misinterpret the requirements, and therefore of reasoning and selecting wrong patterns or making wrong architectural decisions, is reduced. In addition, these annotations are used by ArchLayer to assist the architect in the selection of architectural patterns.

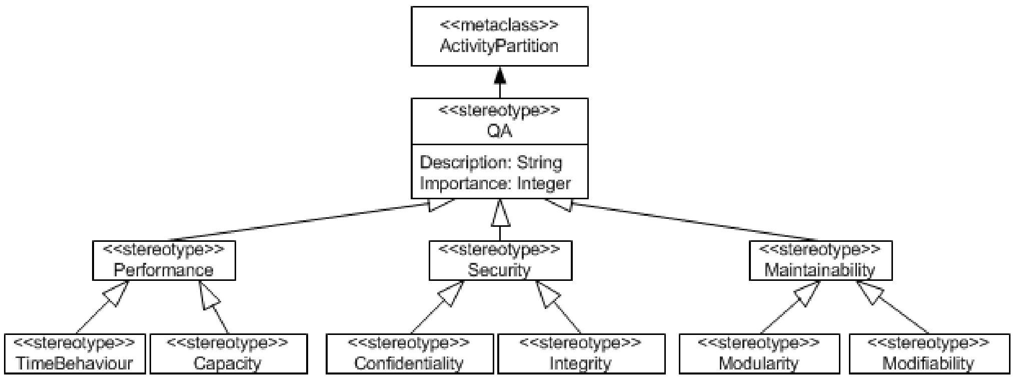


Figure 3.13: Excerpt of the profile for modeling the relationships with quality attributes in activity diagrams.

Annotating activity diagrams

The profile extending activity diagrams, showed in Figure 3.13, first, defines the QA stereotype. This stereotype extends the *ActivityGroup* metaclass. This metaclass is used to group nodes and edges. The QA stereotype extends this behaviour to groups the nodes and edges that are restricted by a specific quality attribute. Thus, each interdependence is documented by means of a rectangle drawn around the nodes and edges constrained by a specific quality attribute.

The QA stereotype defines the general information for all quality attributes and is in turn extended by a hierarchy of stereotypes similar to the one detailed for the use case diagram. These stereotypes specializes the QA stereotype following the hierarchy of characteristics and subcharacteristics defined the SQuaRE quality model. In this way, one can annotate which specific actions of each diagram are restricted by each quality attribute.

Annotating the activity diagrams provides fine-grained information on the requirements' relationships, as even the actions that have to comply with each quality attribute are documented. This information is very useful for the architect since it allows him/her to identify the specific actions that must satisfy each restriction without having to thoroughly analyze the requirements documents. This allows him/her to make much more accurate architectural decisions, without having to perform an in-depth analysis of the requirements, and facilitates their application, since he/she knows the exact actions where apply them. Also, these annotations can also be used by ArchLayer to automatically suggest the most appropriate architectural decisions.

3.3 Architectural decisions repository

This section describes in detail the architectural decisions repository used in ArchLayer. This repository is one of the main tools in assisting the architect in the design of multi-layer framework based architectures. It captures the architectural and technological variability of multi-layer applications. The following subsections detail its internal structure, an example of a possible repository, how it can be enriched with information about the quality attributes affected by each technology included and how the results of using it can be used as a knowledge source for future projects. The architectural decision repository used in this thesis can be found in Appendix A.

3.3.1 Architectural decisions repository structure

The main goal pursued in the creation of this repository is to incorporate architectural and technological variability into the development of multi-layer applications. As mentioned in Section 2.4.1, Cardinality-Based Feature Models are used in this thesis to create the architectural decision repository.

Since the purpose of the repository is to be used as a core part of a model-driven development process, it need a well-defined structure or conform to some kind of meta-model so as to be treated automatically later in the process. Such structure, however, has to be flexible enough to incorporate the large number of existing technologies. It also has to be capable of incorporating both any new technology that might arise and the evolution of existing technologies. Indeed, this was the main criterion imposed on the creation of the architectural decisions repository.

To obtain a repository that meets these requirements, a bottom-up strategy was followed. In particular, a significant number of development frameworks were studied in order to extract concepts that would form the structure or meta-model of the feature model. For the structure to be as flexible as possible, more than 10 Java development frameworks were chosen from different developers and with different roles and goals. These frameworks were selected for being among the most commonly used within their scope. Following is the lists of frameworks analysed: Axis, CXF, DWR, Hibernate, Ibatis, JDBC, JSF, jUnit, Log4j, PicoContainer, Spring, SpringSecurity, SpringWS and Struts.

The first architectural decision to be taken when building a multi-layer application is to determine the layers of which it will be composed. Therefore, the first criterion used in analyzing the development frameworks was to determine the layer or layers in which they are used.

After determining the layers that make up the application, the architect usually define the design patterns to be used in implementing each of them, although, as mentioned above, the architectural decisions can be taken in the inverse order. In particular, knowledge of which design patterns a development framework supports is of particular importance at this stage.

Finally, a framework may allow different kinds of implementation for the same design pattern. If one does not want to lose the advantages offered by the development frameworks, these different implementation techniques need to be taken into account in the architectural decisions repository.

Given these considerations, the frameworks listed above were studied. The information extracted from that analysis is summarized in Table 3.1.

Starting from this information, the structure that the feature model would need to have was extracted. Figure 3.14 shows a feature model with that structure.

In general, the scope of all the frameworks studied was in a single layer. Even when the same developer supports multiple layers, this is usually implemented by being distributed among different frameworks that can be used independently. For example, the frameworks Spring, Spring Security, and SpringWS belong to the same developer, but each targets a single layer and is treated as a separate product. This, together with the fact that the layer is the main architectural pattern applied in the development of multi-layer applications, makes it reasonable that the most

Table 3.1: Essential framework information used in the creation of the architectural decisions repository.

Framework	**Layer**	**Design Patterns**	**Implementation techniques**
Axis	Web services	SOAP	NA
CXF	Web services	REST	NA
DWR	Presentation	Web remoting Page rearrangement	NA NA
Hibernate	Persistence	DAO	JPA XML Annotations
Ibatis	Persistence	DAO	NA
JDBC	Persistence	DAO	NA
JSF	Presentation	MVC Web remoting Page rearrangement	NA NA NA
jUnit	Test	xUnit	NA
Log4j	Log	Logger	NA
PicoContainer	Business logic	IoC	NA
Spring	Business logic	IoC	XML Annotations
SpringSecurity	Security	Authentication Authorization	LDAP OpenID JaaS HTTP Web request Method invocation Access to instances
SpringWS	Web services	REST	NA
Struts	Presentation	MVC	NA

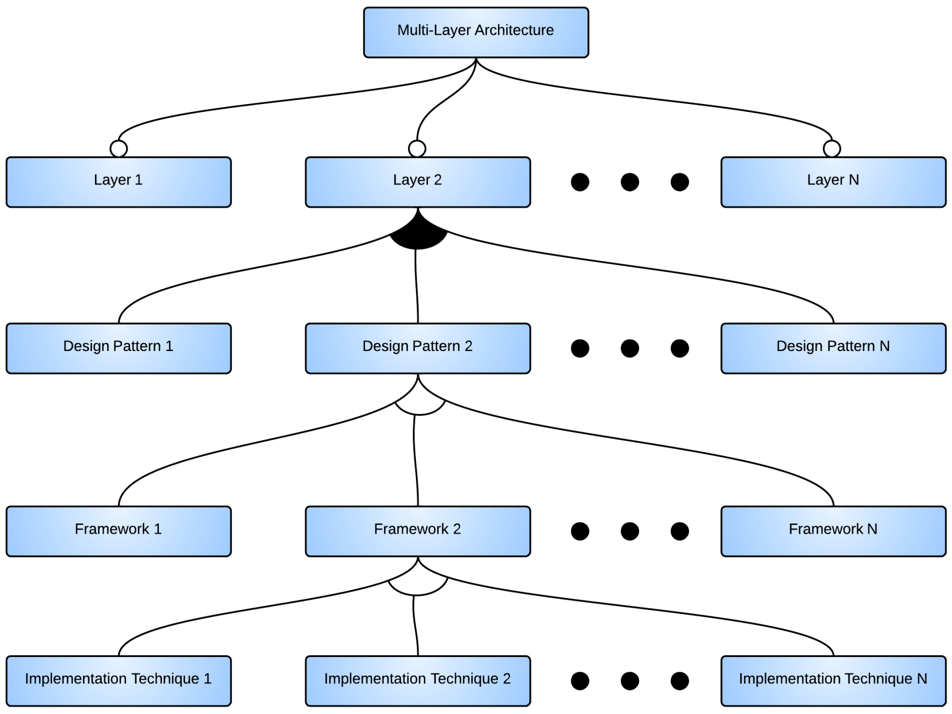


Figure 3.14: Architectural Decisions Repository meta-model.

appropriate would be for the first level of the feature model to consist of the possible layers of which an application may be composed.

As mentioned above, one or more design patterns are used to simplify the implementation of each layer. Usually these design patterns are specific to each layer. Therefore, it would be appropriate for each layer present in the feature model to include the group of features representing the design patterns that can be used.

Usually development frameworks provide support for one or more design patterns. Therefore, each design pattern included in the feature model must specify the frameworks that can be used to implement it. This may result in the same framework appearing more than once in the feature model, provided that this framework supports several design patterns. This poses no problem, since the occurrence of a framework in the model implies that the framework can be used in the implementation of a particular pattern, but it does not imply that its use for a specific pattern requires the use of the same framework in all the patterns of a given layer. For example, the JSF framework supports the implementation of three design patterns

– MVC, web remoting, and page rearrangement. This implies that the framework will appear thrice in the feature model. Nonetheless, the use of JSF to implement the MVC pattern in a particular application does not imply that other frameworks can not be used for the other patterns.

Finally, it is common for a framework to provide different techniques for implementing a design pattern. These techniques generally vary in the syntax used, but end up providing the same results. An example of this is dependency injection in the Spring framework. This can be done using Java annotations or using an XML configuration file. This variability aspect was also taken into account in the feature model. Where applicable, the feature model offers the techniques supported by a framework in the implementation of a given design pattern.

3.3.2 Example of a possible architectural decisions repository

Based on the information obtained during the framework analysis (Table 3.1) and the structure shown in Figure 3.14, a feature model that captures the variability of the frameworks considered was constructed. A fragment of that model is shown in Figure 3.15.

This model can not be created solely with the information presented in the previous section, however. For the model to be representative enough for use in a development process, it must contain information not only about the frameworks but also about their interrelationships. Next section details how this information can be added to the repository.

3.3.3 Enriched architectural decisions repository

The repository of architectural decisions plays a key role in ArchLayer. However, from the architect viewpoint, the repository described to this point only offers a set of possibilities for the decisions that must be taken but not help at all when it comes to taking them. An enriched repository is needed in order for it to be of assistance to the architect. To accomplish this, additional information has been added to the

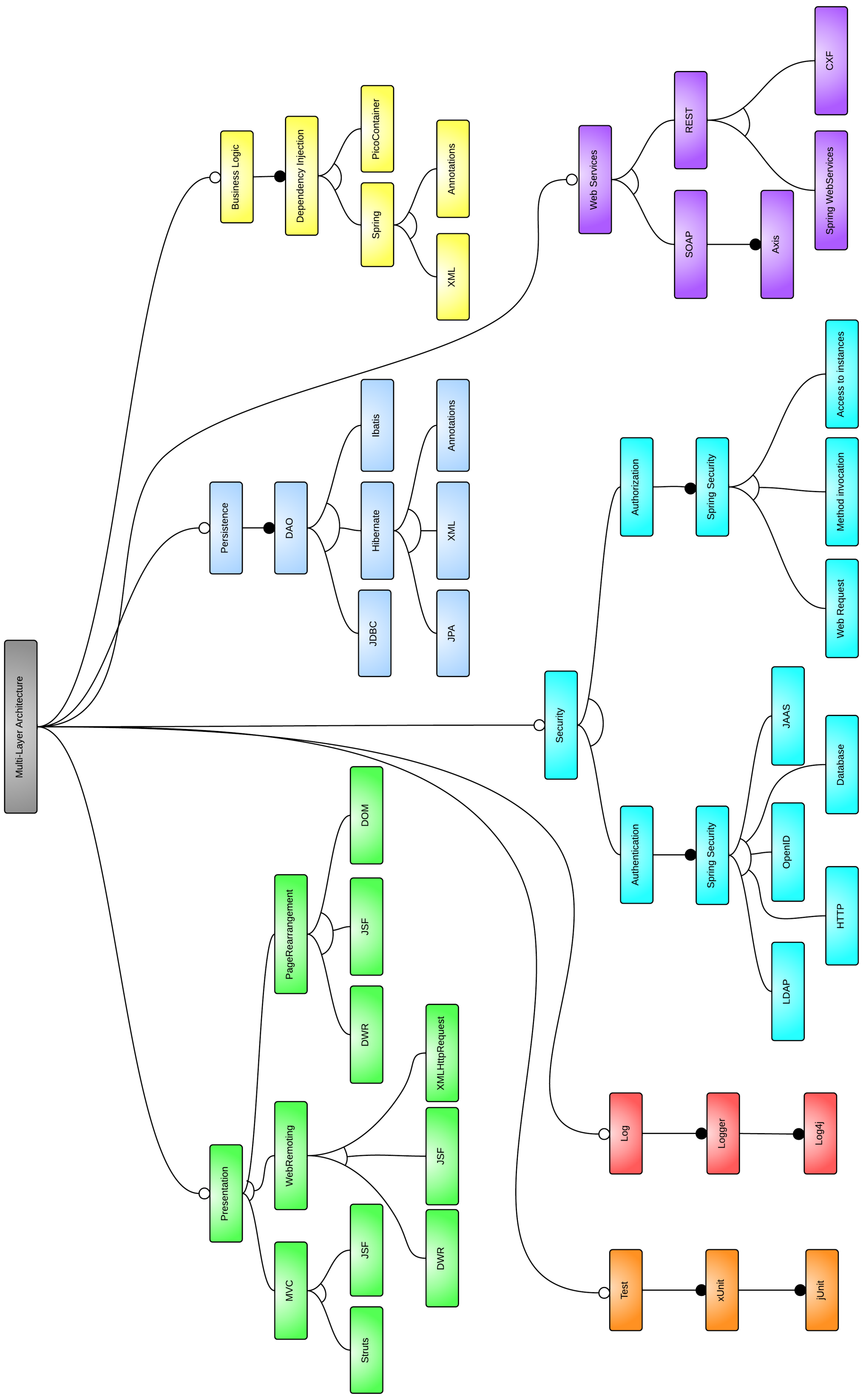


Figure 3.15: Example of a possible architectural decisions repository

repository. The purpose of this information is to ease the decision-making process for the architect.

It is usual, in applications such as the ones discussed in this work, that multiple frameworks could be used to implement a functionality. Usually, such frameworks even implement the same design patterns. Therefore, the choice of one of those frameworks is not simple. Since functionality can be covered with either one, their suitability for the project should be measured by how they affect its Quality Attributes (QA).

Leaving this decision entirely in the hands of the architects requires from them an even greater knowledge about the technical details of each framework and especially of the interactions between them. Adding information to the repository about how the different framework positively or negatively affect the QAs of an application can help the architects take these decisions.

The first step adding this information to the repository is to add specific information for each framework. To accomplish this task we use a slightly modified version of the template described here (Babar, 2004). This template was designed to capture architecturally important information about design patterns and document it in such a format, which makes it readily usable during software architecture design and evaluation activities. Table 3.2 shows the modified template.

The template was modified to collect the architectural information about a design pattern implementation by a framework. Particularly relevant to this work are the affected quality attributes, that may vary from the same information on the design pattern depending on the framework implementation details.

Since frameworks usually implement more than one design pattern, there would be more than one template per framework. They detail how its implementation of each design pattern affects the quality attributes of the final system. Having different templates is needed because using a framework for one of its implemented design pattern or another can have different effects over the final system. Also, this way to include the information matches with how frameworks are represented in the feature model, appearing as many times as design patterns they can implement.

Table 3.2: Template for the architectural information of a framework

Framework name: Name of the framework	**Pattern name**: Design pattern implemented by the framework	
Brief description	A brief description of the framework	
Context	The situation for which the framework is recommended.	
Problem description	What types of problem the framework is supposed to address?	
Suggested solution	What is the solution suggested by the framework to address the problem?	
Forces	Factors affecting the problem & solution. Justification for using the framework.	
Available tactics	What tactics are used by the framework to implement the solution?	
Affected attributes	**Positively**	**Negatively**
	Attributed supported	Attributed hindered
Supported general scenarios	s1	
	s2	
	sn	
Usage Examples	Some known examples of the usage of the framework to solve the problems.	
Implemented design patterns	Other design patterns implemented by the same framework.	

Table 3.3: Template for the effects of a combination of frameworks on a system QAs

Framework A name: Name of the framework A		**Framework B name**: Name of the framework B
Communication mechanism	A brief description of the mechanism or technique used to communicate the pair of frameworks	
Affected attributes	**Positively**	**Negatively**
	Attributed supported	Attributed hindered

Having this information in the decisions repository provides two advantages. First, architects have all the relevant architectural information about the framework present when taking an architectural decision. Second, since the information about the rest of design patterns supported by a framework is included, it is easy to link the different occurrences of a framework in the repository and take advantage of the frameworks implementing more than one design pattern.

However, as mentioned in Section **??** this information is not enough to describe how different frameworks may affect the system QAs. Table 3.3 shows the template used to record the information about the effects the interactions between different technologies have in the sustem QAs. This new template does not make the complexity of the repository grow exponentially, only information about pairs of frameworks that may effect the system QAs when used together should be included. Therefore the use of this template is limited only to pairs of frameworks in the same layer or in adjacent layers, since they are the only ones exchanging information. Neither is it necessary to consider combinations of more than two frameworks since complex interactions can always be reduced to a set of interactions between pairs of frameworks.

All this information is added to the repository so it can be used by the architect but also by the model transformation suggesting the most adequate frameworks for an application.

3.3.4 Reuse of architectural decisions

An additional advantage of the use of feature models for the implementation of the architectural decisions repository is that, once the decision-making process is completed, a configuration of the feature model containing all the architectural decisions is provided. This configuration of the feature model has multiple uses.

Firstly, it is used to ensure that the decisions taken by the architect were held in the project development and properly implemented. This information is very useful, especially in the context of distributed development centers, to measure different aspects of the productivity of development teams.

Secondly, it is used to study the results obtained, especially regarding the quality attributes, in the system. An analysis of the system behavior along with the architectural decisions made during the development permits continuous adjustment in the repository that will benefit future projects.

Finally, it is used in the development of similar projects. Again, this is especially useful in distributed development environments where many similar projects are developed. A development team with little experience can take advantage of all the architectural decisions made by a more skilled team in a previously successful project.

3.4 Framework information meta-model

This section describes the meta-model that captures the implementation details of frameworks. Models based on this meta-model facilitate the introduction of technological variability in ArchLayer. These models are used by the transformations that produce a specific design adapted to whichever technologies are selected by the architect. The following subsections describe the meta-model design rationale, the details of the meta-model and its flexibility to support the modeling of new frameworks. One of the models based in this meta-model used in this thesis can be found in Appendix B.

3.4.1 Meta-model design rationale

The main goal pursued in the creation of this meta-model is to incorporate technological variability into the development of multi-layer framework based applications.

Table 3.4: Development frameworks analysed for the creation of the meta-model.

Framework	**Version**
Axis	1.4
CXF	2.2.4
DWR	2.0.3
Hibernate	3.4.0.GA
Ibatis	2.3.4
JDBC	4.1
JSF	1.2.b19
jUnit	4.7
Log4j	1.2.14
PicoContainer	1.2
Spring	3.0.0
SpringSecurity	2.0.4
SpringWS	1.5.10
Struts	1.3.10

Having a meta-model to define the technical details of a framework facilitates this task. On the one hand the meta-model defines a domain specific language to capture the technical details of the frameoworks. On the other hand, the use of standard techniques such as creating MOF-based meta-models allows easy integration of this meta-model and the corresponding models based on it with the rest of the process.

Additionally, the meta-model has to be flexible enough to incorporate the large number of existing technologies. It also has to be capable of incorporating both any new technology that might arise and the evolution of existing technologies. Indeed, this was the main criterion imposed on the creation of the meta-model.

To obtain a meta-model that meets these requirements, a bottom-up strategy was followed. In particular, a significant number of development frameworks were studied in order to extract concepts that would form the meta-model. For it to be as flexible as possible, more than 10 Java development frameworks were chosen from different developers and with different roles and goals. These frameworks were selected for being among the most commonly used within their scope. Table 3.4 lists the frameworks analyzed and their version.

To carry out this study several multi-layer applications were created. In these applications we study how to setup the frameworks for their use and how to implement the functionality offered by each of them. The first aspect to consider when using a development framework in the implementation of a multi-layer application is to integrate it with the other frameworks used in the project. Frameworks are complex tools and the use of several of them into the same project so that they can communicate and interact with each other is not a trivial task. Therefore, this is an issue which requires special attention when attempting to create a system where multiple frameworks coexist. Hence this is one of the key issues that have shaped the structure of the meta-model presented here.

Moreover, it is useless to configure multiple frameworks in a project if they can not be used conventionally. The other aspect that has led the creation of the meta-model is the utilization of the frameworks. Frameworks are commonly used by establishing framework completion code. This code instantiates domain-specific concepts provided by the frameworks. The frameworks study has focused specifically on the general characteristics of this framework completion code and how it can make use of services provided by other layers and frameworks.

Given these considerations, the frameworks of Table 3.4 were analyzed. The information extracted from that analysis is what led to the creation of the meta-model presented here.

3.4.2 Framework information meta-model

Based on the information obtained from the analysis of the frameworks previously described the meta-model showed in Figure 3.16 was created. Each model based on this meta-model contains the information required for the use of one or more frameworks. These frameworks are identified by a name and version number.

The models based on this meta-model contain very low-level information about the use of a framework. Therefore, they will not be usually created by development teams during the development of multi-layer applications, they should be created by experts in each of the frameworks or even by the frameworks developers.

A particular model for a specific framework will contain an unspecified number of framework concepts. These concepts are identified by their name and they can

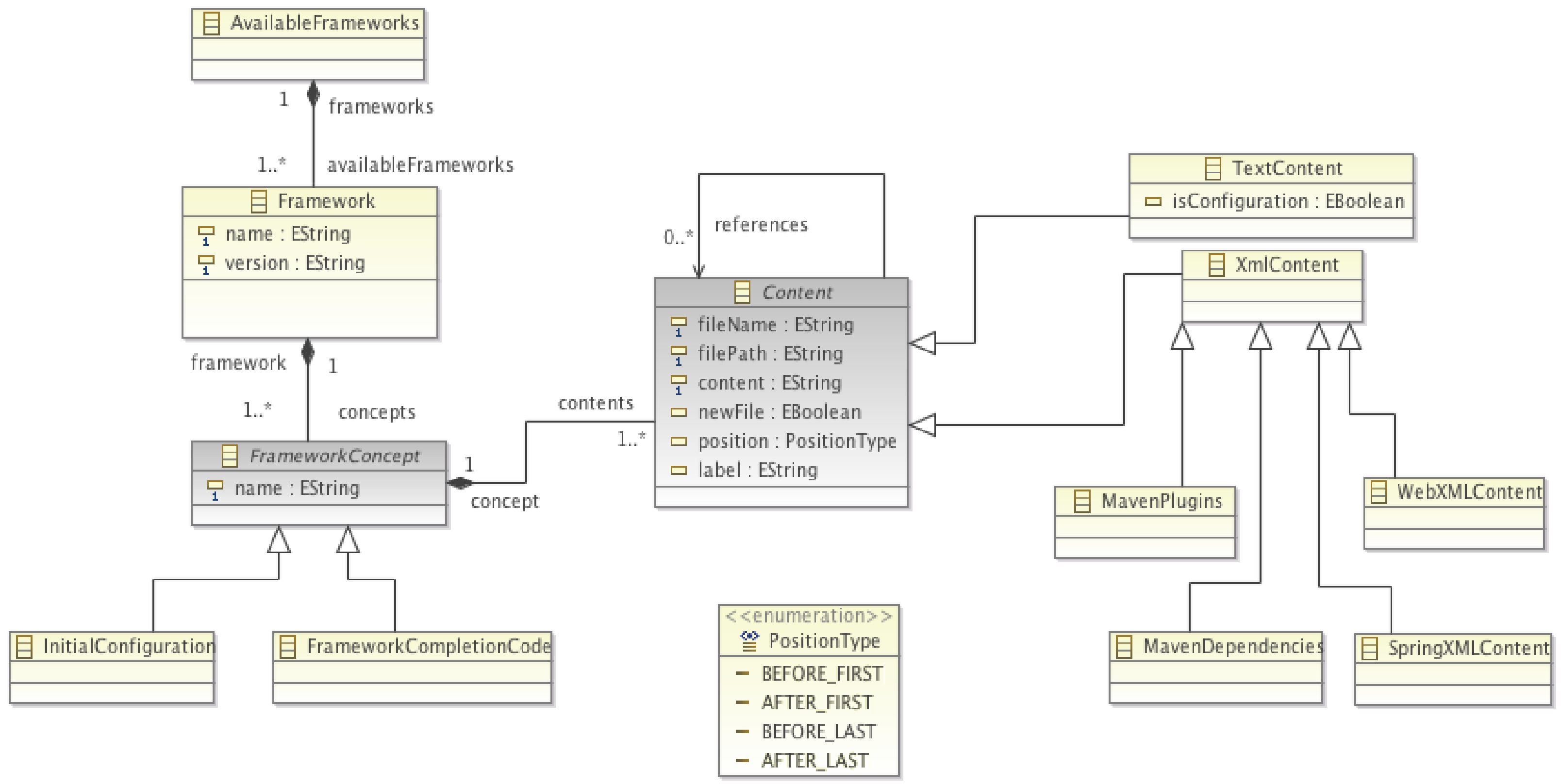


Figure 3.16: Meta-model for modeling the additional information needed about the development frameworks.

be of two possible types: initial configurations or framework completion code. The following sections provide an insight of each element of the meta-model and of how the models based on it are used in ArchLayer.

Initial configuration

As mentioned above the initial configuration is one of the most important aspects when it comes to using a framework. This configuration will enclose, in the project, all of the elements needed by the framework to work properly in conjunction with the remaining frameworks being used.

This concept is represented by the InitialConfiguration metaclass in the meta-model presented in figure 3.16. This type of elements are identified by their name. This has been designed as such because in certain situations it may be interesting to have different initial configurations for the same framework. These configurations can be used to include a framework in projects that use different sets of technologies, and therefore may require different configurations, or when the same framework can be used in different ways depending on its initial configuration. The concept of having multiple initial configurations adds another element of variability in the use of this meta-model: variability is not only present for the chosen technologies, but also as an inherent factor of the frameworks.

Each possible initial configuration of a framework is composed of a set of Contents. During the study of the frameworks presented in table 3.4 it was found that configuring these frameworks for their use in a project implied including content in several files which could be both plain text and XML configuration files. The inclusion of these contents in a project may involve creating a new file in the project or adding content to an existing file.

All these concepts are reflected in the meta-model by the properties of the Content metaclass. In these properties one can define the name of the file in which the new content will be included, the path where the file is located within the project, the content to be added, if it is a file that already exists or it must be created and in case of an existing file in which position inside the file the new content should be inserted.

To determine the position in which the new content will be included in the file a combination of the position and label attributes is used. The position attribute can take four possible values (before first, after first, before last and after last). These values refer to the position where to insert the new content in relation to the value of the label property. This property allows one to specify a label, which can be plain text or XML, for searching in the file in which the new content will be included. Content can be included before or after the first or last appearance of the label in the file. This set of properties is enough to specify the positions in which content should be added for the initial configurations of all the frameworks shown in table 3.4.

The distinction between plain text and XML contents is done simply to differentiate the types of values that can be found in the content property. This differentiation simplifies model-to-text transformations when the contents included in the model have to be inserted into the projects where they are needed.

Additionally, a specialization of the XML content in four different types is included. These types are not really necessary for the use of the meta-model, but their presence is justified in order to simplify the addition of four very common content types in the configuration of the studied frameworks. Any content of these four types can also be described using the XML content type. A brief description of these four types is included below.

- MavenDependencies. This type is used to add dependencies to a project using the Maven tool. Maven is a widely used project management tool that, among other functions, facilitates dependency management. If a different dependency management tool is used, its configuration can be included by using the basic types of content.
- MavenPlugins. Like the previous type, this type is used to simplify adding content in projects that use Maven. In this case this type is used to add Maven plugins to the project.
- WebXML. This type is used to add content to the web.xml configuration file.

This file is the deployment descriptor for web applications that is used for most of the frameworks listed in table 3.4.

- SpringXML. This type is used to add XML content to Spring configuration files. Spring is one of the frameworks studied in this work and is especially relevant because it provides different mechanisms for integration with other frameworks. These mechanisms can be exploited more easily with this type of content.

The initial configurations of all the studied frameworks can be created with the elements that have been described.

Framework completion code

In addition to the initial configuration, in order to manage the technical details regarding the use of a development framework, information about how the framework's completion code should be is required. This refers to elements in a project and not just source code, that are used to instantiate a framework provided concept, which will usually be an implementation of a design pattern.

The meta-model presented here manages these elements in a very similar way to the initial configurations. This is done with the FrameworkCompletionCode metaclass. Like the initial configurations these elements are identified by their name. It is also common to find several elements of this kind for the same framework, in this case this is due to two reasons: on one hand, a framework can provide implementations for different concepts, each of which require an element of this type, and on the other hand, it is not unusual for a framework to provide different techniques to implement the same concept. This allows the variability provided by the frameworks to be maintained in the models.

The other elements of the meta-model for specifying the content of a framework completion code match the elements that specify the content of the initial configurations. The study of the frameworks found in table 3.4, has allowed us to verify that the framework completion code required for the use of such frameworks can be specified with this meta-model. However, an important difference must be noted:

the content of the initial configuration elements in both plain text and XML, is a static content, since they can only appear once in a given project, whereas in the case of the framework completion code, it must have certain dynamic attributes. This is because in the same project it is usual to use the same framework completion code several times. For example, the DAO pattern implementation provided by the Hibernate framework will be used as many times as persistent objects exist in a project.

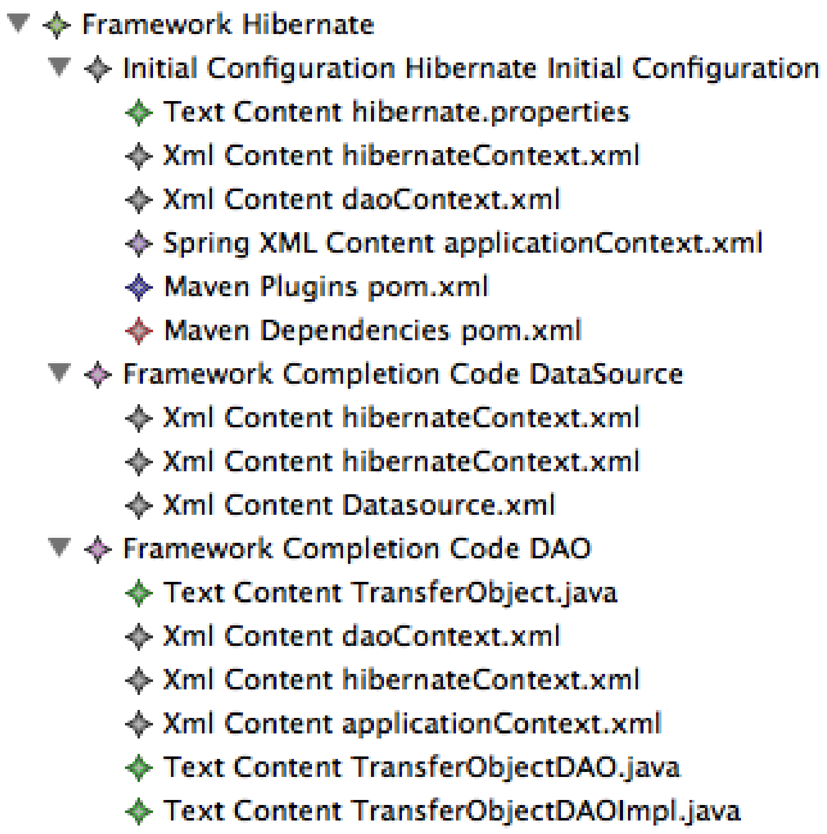
This issue is solved by the use of templates in the contents of elements of the FrameworkCompletionCode type. These templates include dinamic values for such elements in order to differentiate each implementation of a framework completion code, leaving the rest of the content static. When using these models, dynamic content can be solved manually by developers for each occurrence of one of these items or automatically using a template engine such as Velocity (*Velocity templating engine*, n.d.) or JET (*JET templating engine*, n.d.).

Figure 3.17 shows an example of a model based on the described meta-model. The model shows...

3.4.3 Flexibility of the meta-model

The main goal pursued by the creation of this meta-model is to provide a simple language to define the technological variability of multi-layer applications based on development frameworks. Therefore, the main criterion followed whilst defining the meta-model has been flexibility. This implies that the meta-model should not only serve to define the technical details of the studied frameworks, but also that it can be used for the greatest number of frameworks as possible, either existent or future.

In order to verify that the meta-model was flexible enough to describe new frameworks it was decided to model, based on the meta-model presented, the technical details of new frameworks, other than those studied. For this we chose the frameworks used in (Antkiewicz et al., 2009). This work specifies a domain specific language for each of the frameworks involved. In our case we modeled the technical details of each of these frameworks in the meta-model presented. Below are some



```
▼ Framework Hibernate
    ▼ Initial Configuration Hibernate Initial Configuration
        Text Content hibernate.properties
        Xml Content hibernateContext.xml
        Xml Content daoContext.xml
        Spring XML Content applicationContext.xml
        Maven Plugins pom.xml
        Maven Dependencies pom.xml
    ▼ Framework Completion Code DataSource
        Xml Content hibernateContext.xml
        Xml Content hibernateContext.xml
        Xml Content Datasource.xml
    ▼ Framework Completion Code DAO
        Text Content TransferObject.java
        Xml Content daoContext.xml
        Xml Content hibernateContext.xml
        Xml Content applicationContext.xml
        Text Content TransferObjectDAO.java
        Text Content TransferObjectDAOImpl.java
```

Figure 3.17: Excerpt of the Hibernate information model.

brief conclusions obtained after the creation of each framework-specific model.

- Struts. This is one of the frameworks included in our study so the model was already available. The study of the language proposed in (Antkiewicz et al., 2009) does not include any features that were overlooked in our original study.
- Applet. The creation of the specific model for this framework, based on the meta-model presented here, resulted fairly straightforward. This is mainly due to it being a relatively simple framework to use which does not require a lot of configuration elements.
- EJB. Just as in the previous case the modeling of this framework in the meta-model posed no problems. Additionally, this framework shares some common concepts with some of the frameworks included in the study which simplified the process.
- Eclipse workbench part interactions. This framework was the one that posed most difficulties in the process of describing its technical details based on the meta-model, from the frameworks included in (Antkiewicz et al., 2009). This is because this is not a web application development framework, but a framework for creating Eclipse based user interfaces. This increases the difficulty of modeling this framework since it uses different concepts than those usually presented by the studied frameworks. However, although its creation represented an increased workload, the model with the technical details of the framework was created without changing the meta-model.

The creation of these additional models shows that the meta-model presented here has enough flexibility to be used with a wide range of technologies, even those not intended for the creation of multi-layer applications.

3.5 Model to model transformations

This section describes in detail the different model transformation used throughout the process. These transformations use all the above mentioned elements, marked initial design, architectural decisions repository and framework information meta-

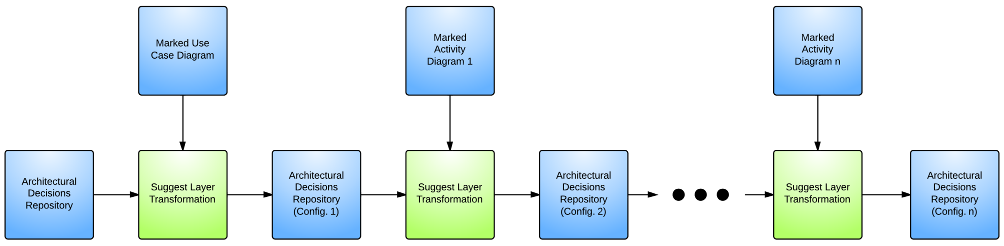

Marked Use Case Diagram
Marked Activity Diagram 1
Marked Activity Diagram n
Architectural Decisions Repository
Suggest Layer Transformation
Architectural Decisions Repository (Config. 1)
Suggest Layer Transformation
Architectural Decisions Repository (Config. 2)
Suggest Layer Transformation
Architectural Decisions Repository (Config. n)

Figure 3.18: Layer Suggestion Transformation application diagram

model, to help tailor the initial design of a system into a specific design for the architecture and technologies chosen by the architect.

Specifically, four model to model transformations are used in ArchLayer and thoroughly described in this section. As mentioned above, all the models transformations have been developed using the ATL transformations language. Extensive excerpt of the transformations source code can be found in Appendix C.

3.5.1 Layer suggestion transformation

The first model transformation used in ArchLayer is the *Layer Suggestion Transformation*. Figure 3.18 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to provide to the architect a possible set of layers to be used in the development of a system. To do this, the transformation take as input a feature model containing an Architectural Decision Repository as described in Section 3.3 and the initial design of the system to be developed marked with information about the QAs the system must fulfill as described in Section 3.2. With this information, the transformation generates a copy of the Architectural Decision Repository in which the suggested layer has been selected.

As shown in Figure 3.18, the transformation is designed in such a way that it can be applied multiple times if the initial design of the system is described in several models. Each application of the transformation generates an enriched layer suggestion that can be used as the input of the next application of the transformation. The final result obtained is the set of layers suggested to implement all the elements

```
if(f.isPersistenceConfiguration()){
    if(UML!DataStoreNode.allInstances().size()>0){
        t.state<-#USER_SELECTED;
    }
}
```

Listing 3.1: Persistence layer suggestion transformation

```
for (stereotype in UCP!Stereotype.allInstances()){
    if(stereotype.name='Authenticity'
            or stereotype.name='Security'
            or stereotype.name='Confidentiality'){
        for (useCase in UML!UseCase.allInstances()){
            if(useCase.isAnnotated(stereotype)){
                t.state<-#USER_SELECTED;
            }
        }
    }
}
for (stereotype in AP!Stereotype.allInstances()){
    if(stereotype.name='Authenticity'
            or stereotype.name='Security'
            or stereotype.name='Confidentiality'){
        for (activityPartition in
                UML!ActivityPartition.allInstances()){
            if(activityPartition.
                    isAnnotated(stereotype)){
                t.state<-#USER_SELECTED;
            }
        }
    }
}
```

Listing 3.2: Security layer suggestion transformation

contained in the different initial design models.

As mentioned above, the transformation will suggest a given layer based on specific features found in the initial design of the application or based on the marks containing information about the QAs of the system. Listing 3.1 shows a fragment of the transformation that suggest the use of a persistence layer if the initial design model contains any DataStore elements.

Listing 3.2 shows a fragment of the transformation that suggest the use of a security layer if any element of the initial design model is annotated with the given QAs.

This simple set of criteria for layers suggestion can be adapted to meet company

```
if(thisModule.halfUCAnnotated(stereotype)){
    t.state<-#USER_SELECTED;
}

helper def : halfUCAnnotated(s : UCP!Stereotype)
        : Boolean =
    if (UML!UseCase.allInstances()->
            collect(    ext | ext.extensionPoint).
            flatten()->select(exten |
            not exten.getAppliedStereotype
            (s.qualifiedName).oclIsUndefined()).
            size())/ UML!UseCase.allInstances().
            size() >= 0.5   then
        true
    else
        false
    endif;
```

Listing 3.3: Alternative security layer suggestion transformation

policies or architects preferences by enriching the transformations that suggest each of the layers. Listing 3.3 shows an alternative to the security layer suggestion that only select such layer if half or more of the use cases of the initial design are marked with the given QAs.

The final product obtained by this transformation is a configuration of the feature model containig the architectural decisions repository in which the suggested layers are selected. This model will be later use by other transformations to further advance in the development process and can also be used or modified by the software architect.

3.5.2 From initial design to layered design transformation

The next model transformation used in ArchLayer is the *Layered Design Transformation*. Figure 3.19 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to generate a design of the application tailored to the layers selected by the architect to develop the system. To do this, the transformation takes as input the set of layers selected in the form of a feature model such as the one generated by the previous transformation and the marked activity

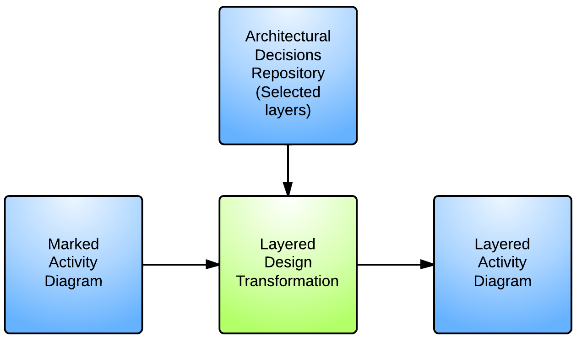

Architectural Decisions Repository (Selected layers)
Marked Activity Diagram
Layered Design Transformation
Layered Activity Diagram

Figure 3.19: Layered Design Transformation application diagram

```
helper def : layers : Sequence(UML!ActivityPartition)=
        Sequence{};

rule selectedLayer2ActivityPartition{
    from f : FMP!Feature
        (f.isSelectedLayerConfiguration())
    to t : UML!ActivityPartition (name<-f.name)
    do{
        thisModule.layers<-thisModule.layers.append(t);
    }
}
```

Listing 3.4: Activity partition inclusion for each selected layer

diagrams of the system. With this information, the transformation generates new activity diagrams in which the selected layers are present and the actions in the activity diagrams are associated to the layers in which they have presence.

This transformation is based on the UML2Copy transformation (*UML2Copy ATL transformation*, n.d.) which generates an exact copy of any UML model. In this case, the *Layered Design Transformation* generates a copy of the input activity model on which various modifications are performed.

The first modification performed to the activity diagram is the inclusion of an activity partition in the diagram for each of the selected layers in the feature model. Listing 3.4 shows the fragment of the transformation that adds the activity partition for the selected layers.

Then, the actions in the activity diagram are included in all the layer in which they have presence. The criteria used to decide in which layers is present a certain

```
if(not thisModule.getLayer('Persistence').
        oclIsUndefined()){
    if(s.outgoing->select(flow|flow.target.
            oclIsTypeOf(UML!DataStoreNode)).size() > 0
            or s.incoming->select(flow|flow.source.
            oclIsTypeOf(UML!DataStoreNode)).size() > 0)
                {
        t.inPartition<-t.inPartition.append(thisModule.
                getLayer('Persistence'));
    }
}
```

Listing 3.5: Actions present in the persistence layer transformation

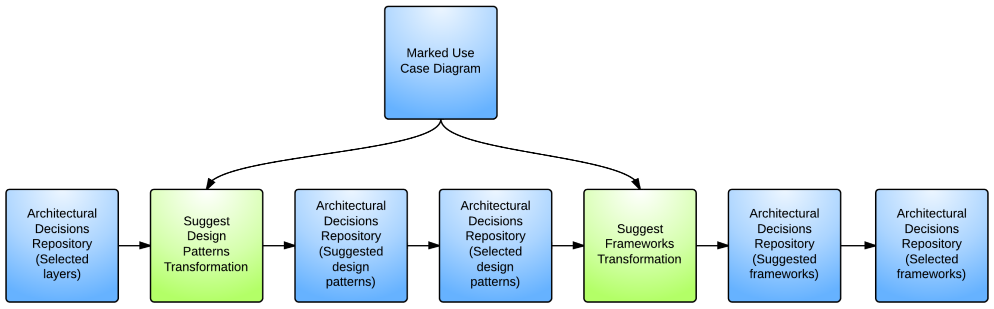

Marked Use Case Diagram
Architectural Decisions Repository (Selected layers)
Suggest Design Patterns Transformation
Architectural Decisions Repository (Suggested design patterns)
Architectural Decisions Repository (Selected design patterns)
Suggest Frameworks Transformation
Architectural Decisions Repository (Suggested frameworks)
Architectural Decisions Repository (Selected frameworks)

Figure 3.20: Design Patterns and Frameworks Suggestion Transformation application diagram

action are similar to those used to suggest the presence of a layer in the above transformation. For example, a given action will be present in the persistence layer if it is connected with a DataStore. Listing 3.5 details the fragment of the transformation that check this specific criteria.

The layered activity diagram generated by this transformation will be later used by other transformations to further advance in the development process and can also be used or modified by the software architect.

3.5.3 Design patterns and frameworks suggestion

The next model transformation used in ArchLayer is the *Design Patterns and Frameworks Suggestion Transformation*. Figure 3.20 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to provide to the architect a possible set of design patterns and frameworks to be used in the development of a system. To do this, the transformation is divided in two. The first one take as input the set of layers selected and the marked use case diagram. With this information, the transformation generates a copy of the feature model in which the suggested design pattern has been selected. The second transformation take as input the previously generated set of selected design patterns and the marked use case diagram and generates a copy of the feature model in which the suggested frameworks has been selected.

The transformation has been divided in two steps in order to give architect the opportunity to refine or validate each level of suggestion independently. Thus, the set of selected design patterns using in the second part of the transformation are not necessarily the ones automatically suggested by the transformation but the ones validated by the architect, which provides a more appropriate output.

To suggest a particular design pattern or framework the transformation uses the information about the QAs affected by them included in the architectural decisions repository, as described in Section 3.3.3. This information is checked against the QAs the system must fulfill as indicated by the marks included in the use case diagram, not forgetting the effect the combination of different design patterns and frameworks has on the final system QAs. Listing 3.6 shows a fragment of the algorithm used to suggest a framework on the basis of such information.

For each selected design pattern this prioritization algorithm suggest the framework that best helps to fulfill the system QAs based on the framework influence in the QAs and in the relations with the already selected frameworks. The final product obtained by this transformation is a configuration of the feature model containing the architectural decisions repository in which the suggested design patterns and frameworks are selected. This model will be later use by the last transformation to further advance in the development process and can also be used or modified by the software architect.

```
--Iteration over the frameworks of a selected
--design pattern
for(feature in designPatternFrameworks){
     QAsMeet<-0;
     --Calculates the priority value of the framework
     --based on the affected QAs
     properties<-feature.properties.children;
     for(property in properties){
          --Positively affected QAs added to the
          --priority value
          if(property.name='PositivelyAffectedQAs' and
               not property.typedValue.oclIsUndefined()){
               affectedQAs<-property.typedValue.
               stringValue.split(',');
               for(affectedQA in affectedQAs){
                    if(affectedQA.trim() <> ''){
                         QAsMeet<- QAsMeet + thisModule.
                         annotatedUC(thisModule.
                         getStereotype(affectedQA.trim()));
                    }
               }
          }
          --Negatively affected QAs subtracted to the
          --priority value
          ...
          --Effects of combination with previously
          --selected frameworks added to the priority
          --value
          if(property.name='CombinationAffectedQAs'
          and not property.typedValue.oclIsUndefined()){
               for(relatedFramework in relatedFrameworks){
                    --Positively affected QAs by relations
                    --increase the priority value
                    --Negatively affected QAs by relations
                    --decrease the priority value
               }
          }
     }
```

Listing 3.6: Framework suggestion transformation

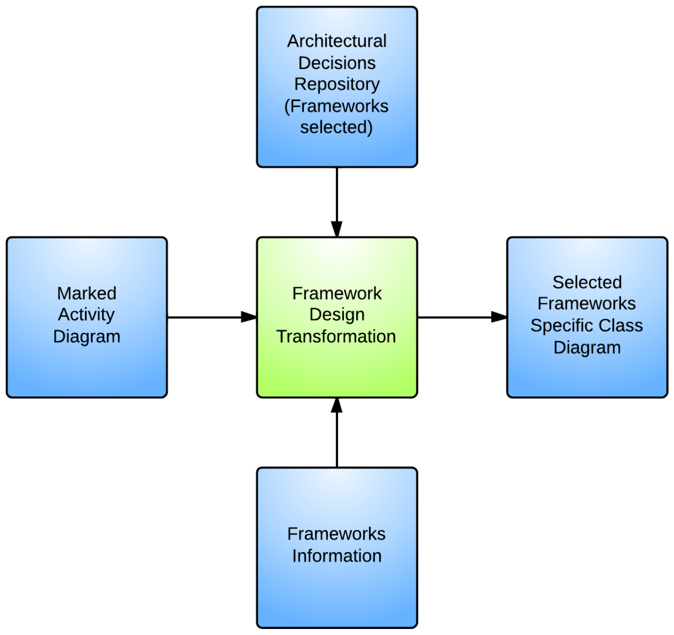

Architectural Decisions Repository (Frameworks selected)
Marked Activity Diagram
Framework Design Transformation
Selected Frameworks Specific Class Diagram
Frameworks Information

Figure 3.21: Technology Specific Design Transformation application diagram

3.5.4 From layered design to specific design transformation

The last model transformation used in ArchLayer is the *Technology Specific Design Transformation*. Figure 3.21 shows the elements of the process involved in the application of this transformation.

The goal of this transformation is to generate a design of the application tailored to the technologies selected by the architect to develop the system. To do this, the transformation takes as input the set of frameworks selected in the form of a feature model such as the one generated by the previous transformation, the marked activity diagrams of the system and the technical information of the selected frameworks in form of the framework information meta-model described in Section 3.4. With this information, the transformation generates an UML class model in which the actions in the activity diagrams are decomposed into the elements that are needed to implement such actions with the selected technologies.

To determine the elements in which each action of the activity diagram is decomposed the transformation takes into account the activity partition representing the layer in which the action is included, as provided by one of the previously described transformations. For each layer in which an action has presence a set of elements will be generated in the class model. The elements generated depend on the technologies chosen for the implementation of each layer. If only one design pattern and framework are selected to implement a given layer the transformation will use the information of such framework provided by the framework information meta-model. However, if more than one design pattern or framework are selected for a given layer, the activity diagram should be enriched as described in Section 3.1.7. the transformation will use this information to select the appropriate information from the framework information model to generate the appropriate output. Listing 3.7 shows a fragment of the described transformation.

The class diagram generated by this transformation will be later used as the basis for automatic code generation or by the projects developers as the design tailored to the architectural decisions taken by the software architect. Being a model at a low level of abstraction, which contains very detailed information about the technologies to be used, it can be used by less experienced developers.

3.5.5 Flexibility of the transformations

The main goal pursued by the creation of these transformations is to provide a mechanism to automatically transfer architectural decisions to the initial design of a multi-layer application. Therefore, as in other elements of the process, the main criterion followed whilst creating the transformation has been flexibility. This implies that the transformations should be useful for applications that use all kinds of frameworks and implement all kinds of requirements.

The flexibility of the described transformation has been verified in the development of several applications using all the frameworks described in the previous sections and implement a wide range of functionality. More details of the applications developed will be discussed later, in Chapter 5. However, due to their nature and how they have been developed the transformations flexibility has some limitations. Below these limits are described.

```
rule OpaqueAction {
    from s : UML!OpaqueAction
    do{
        for(partition in s.inPartition){
            selectedFrameworks <- thisModule.
                selectedFrameworks(partition.name);
            if(selectedFrameworks.size() = 1){
                framework <- selectedFrameworks.
                first().name;
                designPattern <- thisModule.
                    selectedPatterns(partition.name);
            }
            else{
                for(annotation in s.eAnnotations){
                    ...
                }
            }
            frameworkInformation <- thisModule.
                getFrameworkInformation(framework);
            for(c in frameworkInformation.concepts){
                for(content in c.contents){
                    ...
                    class <- UML!Class.newInstance();
                    model.packagedElement<-
                        model.packagedElement.
                        append(class);
                    for(ref in content.references){
                        dependency <- UML!Dependency.
                        newInstance();
                        ...
                    }
                }
            }
        }
    }
}
```

Listing 3.7: Specific design transformation

- All transformations are based on the use of meta-models. In this particular case in the UML meta-model, the Cardinality Based Feature Modeling meta-model and the Framework information meta-model. Any change in any of these meta-models implies changes in the transformations for them to remain effective. Likewise, transformations flexibility is limited by meta-models flexibility. If a particular framework can not be modeled using the framework information meta-model the transformations can not be applied.
- The criteria to suggest a particular layer and how it applies to the design of the application are embedded in the transformations. Therefore, if a new layer is included in the architectural decisions repository the transformations should be modified accordingly to include the layer specific information.
- Similarly, all the criteria used for suggesting the different architectural decision are embedded in the transformations. The used criteria come from the author experience and should be modified to match companies policy or software architects preferences.
- UML is a complex modeling language containing approximately 200 meta-classes. The transformations have been designed to work with Use Case diagrams and Activity diagrams. If the transformations are going to be used with other elements of UML or the same elements used differently they must be adapted accordingly.
- Finally, additional information is used by some of the transformations. Specifically, information about the QAs of a system affected by a given architectural decisions and information about the technology that shoul be applied to actions in activity diagrams. If this information is added differently than as presented in this work the transformations should be modified accordingly.

3.6 Conclusions

Explicar como las contribuciones detalladas en este capitulo permiten resolver los problemas planteados al final del capitulo dos

In this chapter ArchLayer, the process proposed in this thesis for bridging the gap between the initial design of an application and its implementation using a framework based multi-layer architecture, has been presented. First there has been a detailed description of the entire process, using a simple example to facilitate understanding. Later special attention has been paid to each of the relevant contributions of this work.

This process provides a solution to the problems identified in the development of multi-layer framework based applications stated in Section 2.5.

First, the proposed process provides a mechanism for the model-driven development of web applications that allows developers to manage the architecture and technologies with which applications are going to be implemented. The combination of the architectural decisions repository, the framework information meta-model and the model transformations let companies using this process use a wide variety of architectures and technologies rather than being constrained by an implicit architecture imposed by the process.

Second, by providing architectural suggestions the process simplify architectural decision making. The growing complexity of web applications and the increasingly important role of architects make necessary any help in this regard.

Third, the process has been designed from the beginning to support the evolution of development frameworks. Both the decisions repository and the meta-model have been designed being this one of their primary concerns.

Fourth, the use of the framework information meta-model provides a generic mechanism for dealing with different frameworks. The meta-model was designed by analyzing a significant number of frameworks and has demonstrated its usefulness for modeling more than 10 different Java frameworks.

Fifth, one of the most relevant aspects represented in the framework information meta-model is the initial configuration of the frameworks. The attention paid to this aspect make it possible to work in applications that integrate multiple frameworks.

Lastly, the combination of all the previous points make the process suitable for its use in distributed development environments with high staff rotation.

Following the presentation of this process, the next chapter will detail the JACA code generation tool. A tool that can be used as the last step of the presented process to automatically generate part of the source code of framework based multi-layer applications. The tool can also be used as a standalone tool for the development of such applications.

Chapter 4

JACA Code Generation Tool

I need you to be clever. I need you to think of solutions to problems we haven't seen yet. I want you to try things that no one has ever tried because they're absolutely stupid.

Ender's Game, Orson Scott Card.

JACA, from the spanish acronym *Java para Aplicaciones Corporativas de la Administración* or Java for Corporate Applications of the Administration is a tool developed to serve as reference architecture and set of good practices in the development of multi-layer framework based applications for the regional government of Extremadura. The tool was developed by the research group of the author of this thesis by request of the regional government to be used as the software architecture of all the applications developed by and for the regional government. Its development caused the start of this thesis, where it is integrated as the code generation tool that automatizes the last step of ArchLayer.

This chapter describes in details the tool developed to automatically generate part of the source code of multi-layer framework based web application based on the process described in the previous chapter. The description of the tool is organized as follows. Section 4.1 details the reasons behind the development of this tool and how its creation served as the main inspiration for the development of this thesis. Section 4.2 is focused on the integration between the tool described in this chapter and ArchLayer. Section 4.3 describes in detail how the tool can be used to automatically generate the initial configuration of a project based on the architectural decisions. Section 4.4 explains how the tool can be used to automatically generate the source code of the different concepts supported by the chosen development frameworks. In Section 4.5 the use of the tool outside of the environment of this thesis is described and the different materials provided alongside the tool are detailed. Finally, Section 4.6 summarize this chapter focusing on the benefits provided by the tool.

4.1 Motivation

In 2008 the regional government of Extremadura was immersed in the development of the PCDAI project, from the spanish acronym Plataforma Corporativa de Desarrollo de Aplicaciones Informáticas or Corporate Platform for IT Application Development. This project aimed to establish a common platform for the development of IT applications for and by the government of Extremadura, for which an application methodology will be defined and a general architecture will be configured to address development projects whatever their size and coverage.

One of the key elements of this project was the definition of the reference software architecture for the development of multi-layer applications. Due to the previous experience of the Quercus Software Engineering Group in developing this kind of applications, the regional government contacted this thesis supervisor to collaborate on the development of such architecture.

At that time, the use of reference architectures in the development of applications for the government was fairly widespread. Figure 4.1 shows a Spain map in which the various existing reference architectures for regional governments are shown.

Extremadura is shown on the map in a different color because, at that time, it did not had its own reference architecture yet. However, Indra Software Labs, one of spanish bigger IT companies had a development center in Extremadura were they have developed their own reference architecture, ArqOS, from the spanish acronym Arquitectura Orientada a Servicios or Service Oriented Architecture. The supervisor of this thesis had previously collaborated with Indra in different projects so it

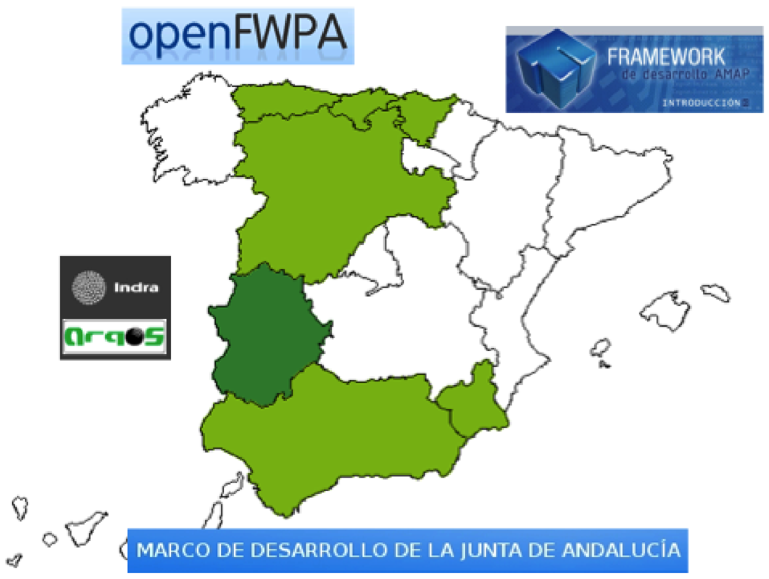

openFWPA
FRAMEWORK de desarrollo AMAP
INTRODUCCIÓN
Indra
ArqOS
MARCO DE DESARROLLO DE LA JUNTA DE ANDALUCÍA

Figure 4.1: Map showing the spanish regions with its own reference architecture.

was decided to incorporate the company to the developing of the regional reference architecture.

The development of the Extremadura reference architecture was based on the ArqOS architecture mainly for two reasons. First, for not starting from scratch which would allow the developers to take the project beyond a standard reference architecture and make a more complex product. And second, because, at the time, Indra was developing the most important regional project and therefore the defined architecture would be compatible with that project from the beginning.

Starting from this point, the development of JACA started with the following objectives:

- JACA should manage the creation of new projects, automatically integrating the development frameworks that will be used in the project in a way transparent to developers.
- JACA should focus on the most widespread frameworks in the industry.
- JACA should be as flexible as possible so the integration of new technologies that arise in this area is simplified. Such, the maintenance costs of keeping it updated should not cause a significant overload negating the benefits provided

by having a reference architecture.

- JACA should provide the developers of projects for the regional government of Extremadura a standardized mechanism for framework usage. Thus, by usign such mechanism all projects for the regional government will follow common implementation techniques improving inter-operation and staff rotation.
- JACA should provide training material for any developer about the standardized mechanism for framework usage.
- JACA should help to create a repository of corporate material such that code reuse is simplified in projects for the regional government.

The following sections describe in detail the different aspect of this tool and its role in ArchLayer. Nonetheless, is important to stress here the fundamental role played in this thesis by the development of this project.

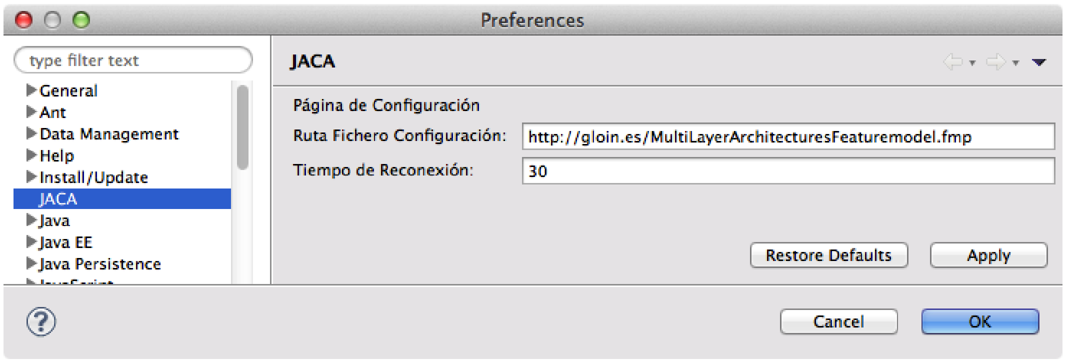
The development of this tool, of which this thesis author is software architect and lead developer, allowed the author to experience firsthand the problems in the development of framework based multi-layer application. All the problems in this context, as described in Section 1.3, were experience during the development of JACA in a real environment with real projects for both the regional government of Extremadura and the Indra development center.

This experience helped to confirm that new techniques were necessary to avoid such problems. Specifically, software engineering techniques like model driven development or cardinality based feature modeling could help to greatly mitigate them.

This situation led to the beginning of the thesis presented here and helped focus the JACA tool so it could be easily integrated into a process such as the one described in the previous chapter.

4.2 Integration with ArchLayer

Besides serving as inspiration for the work done in this thesis, JACA is integrated into ArchLayer. Specifically, as an automatic code generation tool. This section



Página de Configuración

Ruta Fichero Configuración: http://gloin.es/MultiLayerArchitecturesFeaturemodel.fmp

Tiempo de Reconexión: 30

Figure 4.2: JACA preferences page.

describes such integration.

The tool itself is developed as a set of plugins for the Eclipse IDE. The tool was developed in such way because integration with a development environment provides several advantages. First, IDEs are one of the fundamental tools of software developers. Therefore, a new tool integrated with their usual environment simplifies the process of adaptation to it. Second, many IDEs, especially Eclipse, provide mechanisms for creating plugins that greatly simplify the integration with them, but they also provide advanced techniques that allows plugin developers to build powerful applications. And third, thanks to its flexibility and power Eclipse has become the de facto standard for model-driven development techniques.

JACA integration with ArchLayer passes through the use of the same models described in the previous chapter. Figure 4.2 shows the preferences page used for the configuration of the JACA plugins.

On this page two parameters are provided to the tool, the path to the configuration file and the reconnect time. The first parameter must be a URL pointing to a model containing an architectural decisions repository as described in Section 3.3. The second parameter is the amount of time, in minutes, the tool would wait until trying to reconnect with the model if the first connection fails.

These parameters provide a particularly important behavior for the tool role. Reading the configuration file from a URL provided the regional government of Extremadura a simple mechanism for controlling the architecture of all its applications, both developed internally and contracted to third parties.

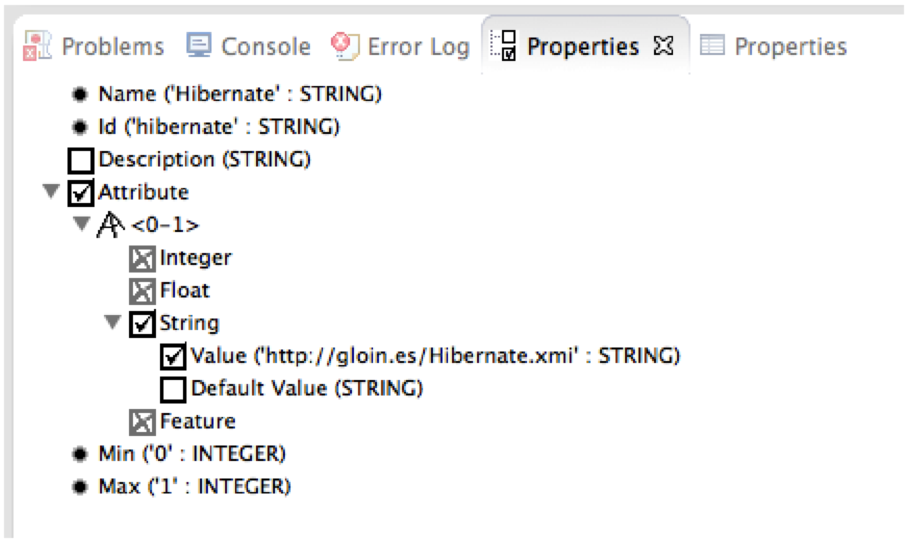
The architectural decisions repository contains the specific design patterns and frameworks that can be used in a development. By requiring developers to point to a specific repository controlled by them, the regional government ensures control over the software architectures used while it has a mechanism to include new architectural elements, as a new version of a framework, quickly and easily by updating the model pointed by the URL. It is even possible to have different models for different types of projects with different sets of architectural decisions.

To avoid the need for the tool to be always connected, whenever a repository is read from the indicated URL a local copy is stored. If at a given time the connection fails, the tool will use the local copy of the model until it can reconnect, waiting the time specified in the second parameter before trying to connect again.

Besides the architectural decisions repository, in order to automatically generate source code JACA need access to the frameworks information models, represented as described in Section 3.4, of all the frameworks included in the architectural decisions repository. To provide access to these models and simplify the synchronization between them and the evolving architectural decision repository a link has been added to the features representing frameworks in the repository. Figure 4.3 shows the properties view of a feature of the architectural decisions repository representing the Hibernate development framework and the link to its framework information model.

The tool works with these models in the same way as it works with the architectural decisions repository. Each time a framework information model is accessed through the link provided in the repository a local copy is stored. If at a given time the connection fails, the tool will use the local copy of the model until it can reconnect. This mechanism allows the organization controlling the different models to quickly and transparently make changes in the code generated by the tool and spread them to all users.

By using these models, JACA has all the information needed to fulfill the role of model to text transformation tool of the process presented in this thesis. The following sections describe in details such automatic source code generation.



```
● Name ('Hibernate' : STRING)
● Id ('hibernate' : STRING)
☐ Description (STRING)
▼ ☑ Attribute
  ▼ <0-1>
      ☒ Integer
      ☒ Float
    ▼ ☑ String
        ☑ Value ('http://gloin.es/Hibernate.xmi' : STRING)
        ☐ Default Value (STRING)
      ☒ Feature
● Min ('0' : INTEGER)
● Max ('1' : INTEGER)
```

Figure 4.3: Link between a framework on the architectural decisions repository and its framework information model.

4.3 Initial configuration generation

As mentioned above, JACA should manage the creation of new projects, automatically integrating the development frameworks that will be used in the project in a way transparent to developers. This is not only one of the main objectives pursued in the development of the tool, but it also addresses one of the major problems in using development frameworks, integrating multiple of them in the same project, as described in Section 1.3.

To fulfill this function the JACA plugins provide a new project wizard in Eclipse. Figure 4.4 show the *New JACA Project Wizard* in an Eclipse instance with the JACA plugins installed. This wizard will allow developers to easily create framework based multi-layer applications in their development environment that will be the basis for the rest of the tool functionality.

The first step to create a project using the JACA wizard is to provide some basic information about the application to be created. Figure 4.5 shows the first page of the wizard asking for the basic project information. Specifically, three information element are required: groupID, artifactID and version. This elements correspond to the information used by Maven to identify dependencies and are used here with

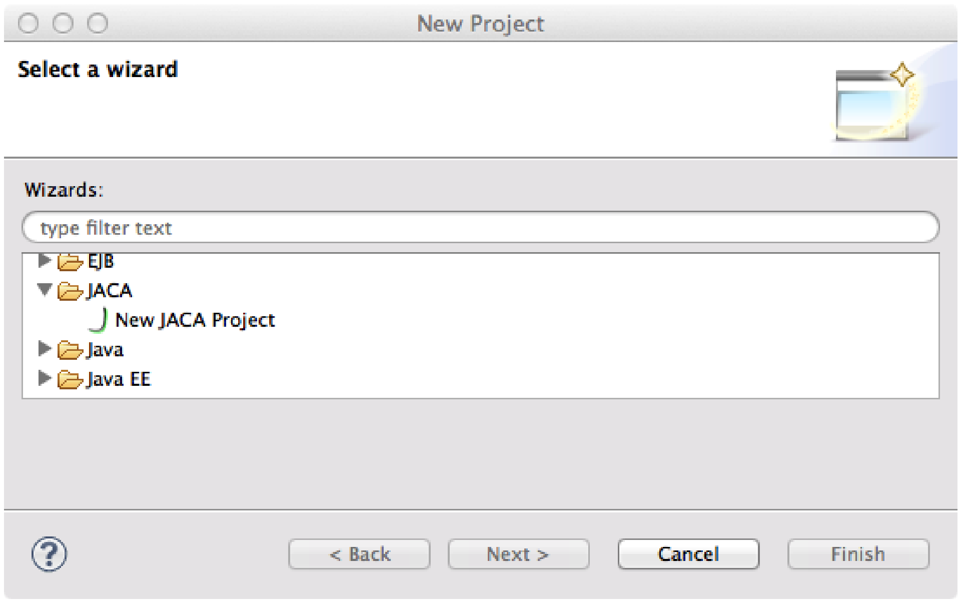


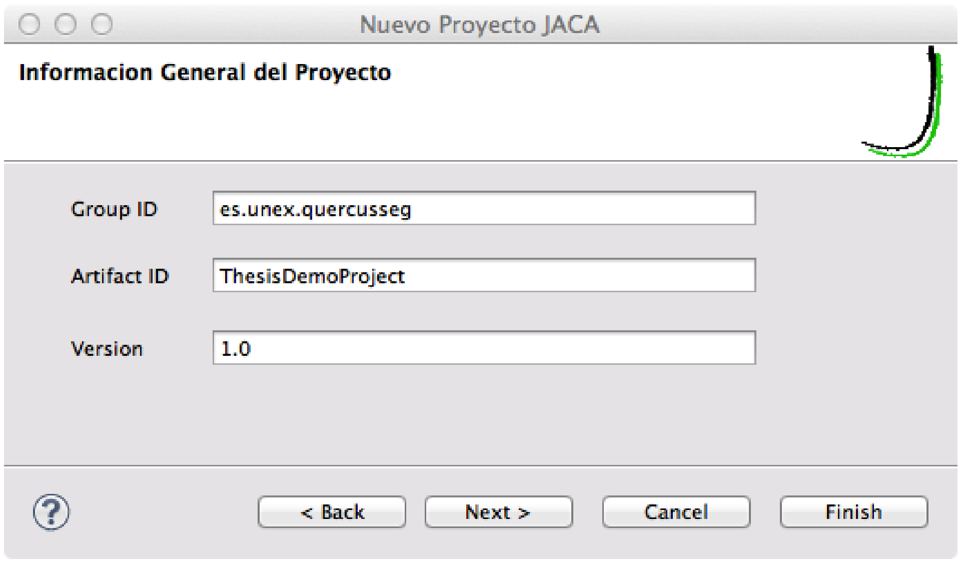
Figure 4.4: New JACA Project Wizard in an Eclipse instance with the JACA plugins installed.

the same meaning, to uniquely identify a project. More information about Maven naming conventions can be found in (*Maven guide to naming conventions*, n.d.).

At this point, the tool will read the architectural decisions repository, as described in the previous section. The information contained in the repository is used to shape the following wizard pages. Specifically, three pages will be shown to the JACA user containing the three levels of decisions in the repository. The first page shown is the layer selection page. Figure 4.6 shows an example of this page containing a check-box for every layer described in the repository. In this page the user will choose the layers in which the application to be developed will be divided.

Based on the layers selected by the user, the next page of the wizard shows the design patterns that can be used in the development of the chosen layers. Figure 4.7 shows an example of such wizard page.

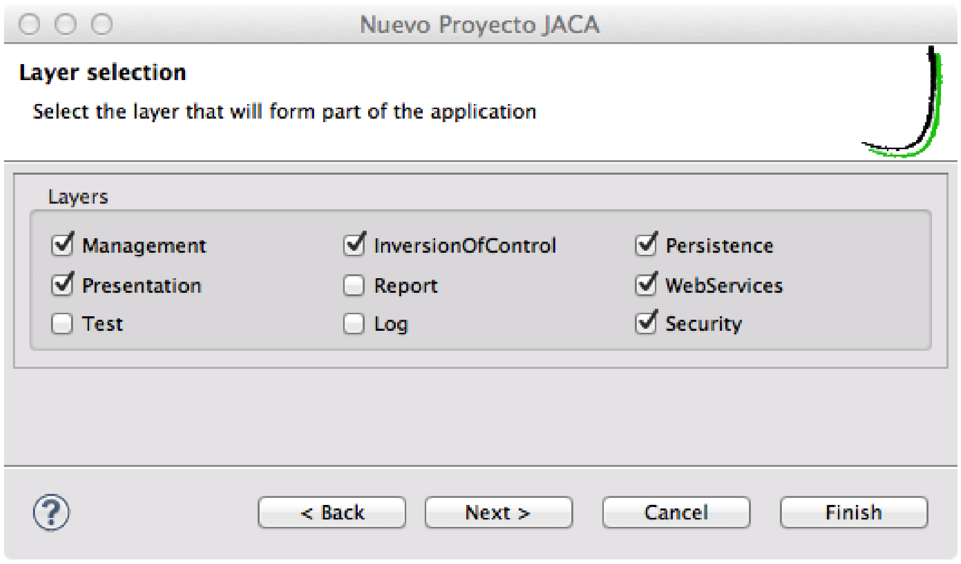
Finally, based on the design patterns selected by the user, the last page of the wizards shows the frameworks that can be used to implement each of the chosen design patterns. Figure 4.8 shows an example of such wizard page. The figure shows two automatically selected frameworks because they are marked as mandatory in the architectural decisions repository.



Informacion General del Proyecto

Group ID	es.unex.quercusseg
Artifact ID	ThesisDemoProject
Version	1.0

Figure 4.5: Basic project information page of the JACA wizard.



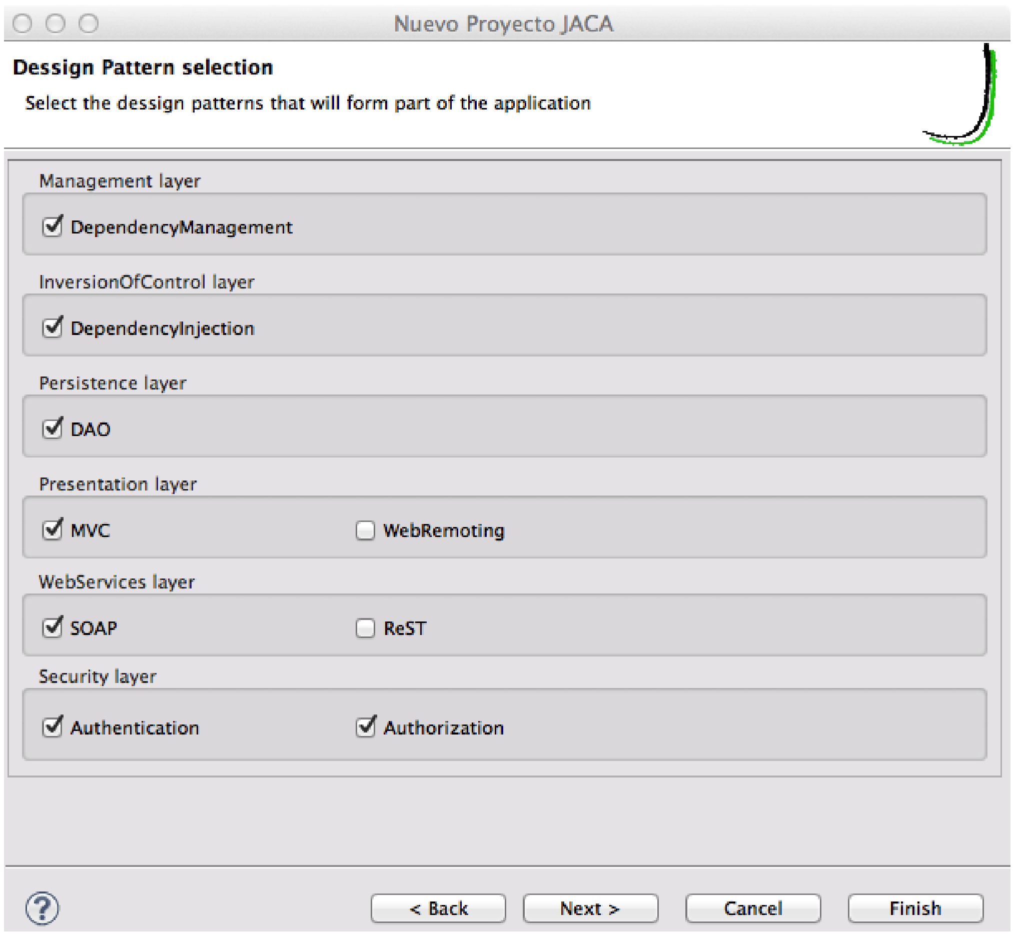
Layer selection

Select the layer that will form part of the application

Layers

✓ Management	✓ InversionOfControl	✓ Persistence
✓ Presentation	☐ Report	✓ WebServices
☐ Test	☐ Log	✓ Security

Figure 4.6: Layer selection page of the JACA wizard.


Nuevo Proyecto JACA

Dessign Pattern selection

Select the dessign patterns that will form part of the application

Management layer

[✓] DependencyManagement

InversionOfControl layer

[✓] DependencyInjection

Persistence layer

[✓] DAO

Presentation layer

[✓] MVC [ ] WebRemoting

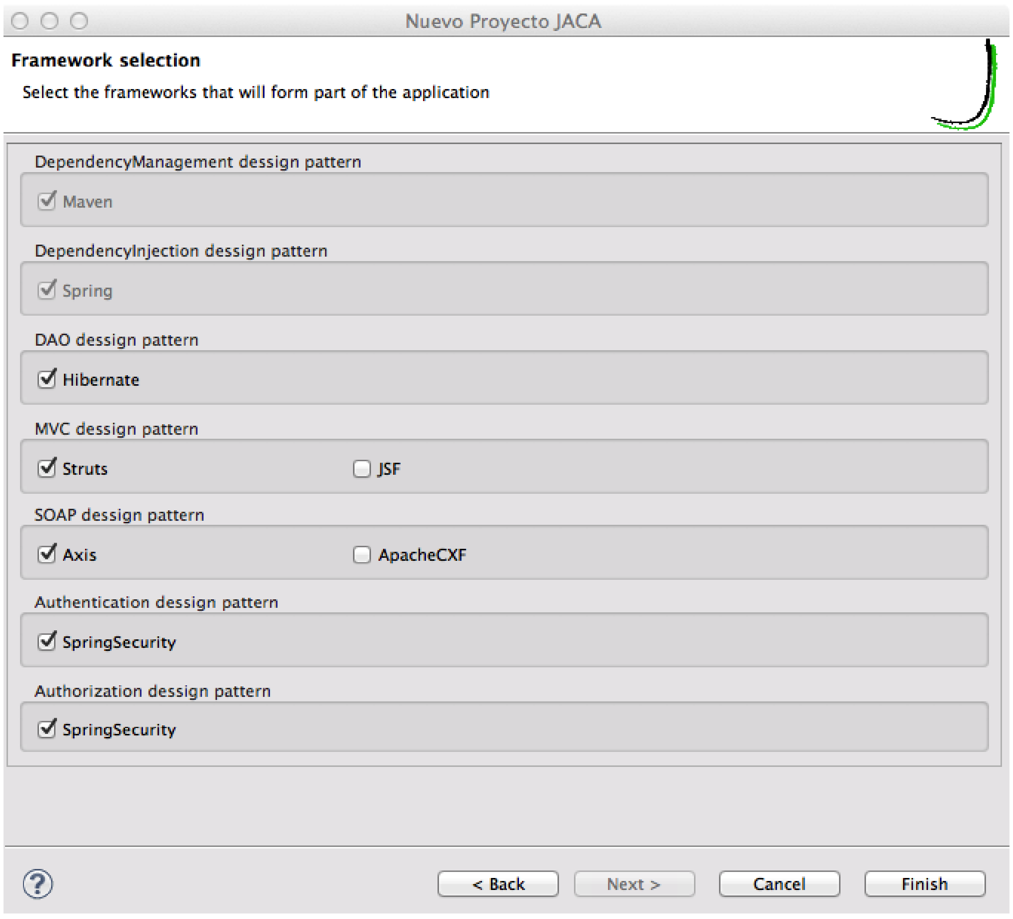
WebServices layer

[✓] SOAP [ ] ReST

Security layer

[✓] Authentication [✓] Authorization

Figure 4.7: Design pattern selection page of the JACA wizard.



Nuevo Proyecto JACA

Framework selection

Select the frameworks that will form part of the application

DependencyManagement dessign pattern

[✓ Maven]

DependencyInjection dessign pattern

[✓ Spring]

DAO dessign pattern

[✓ Hibernate]

MVC dessign pattern

[✓ Struts] JSF

SOAP dessign pattern

[✓ Axis] ApacheCXF

Authentication dessign pattern

[✓ SpringSecurity]

Authorization dessign pattern

[✓ SpringSecurity]

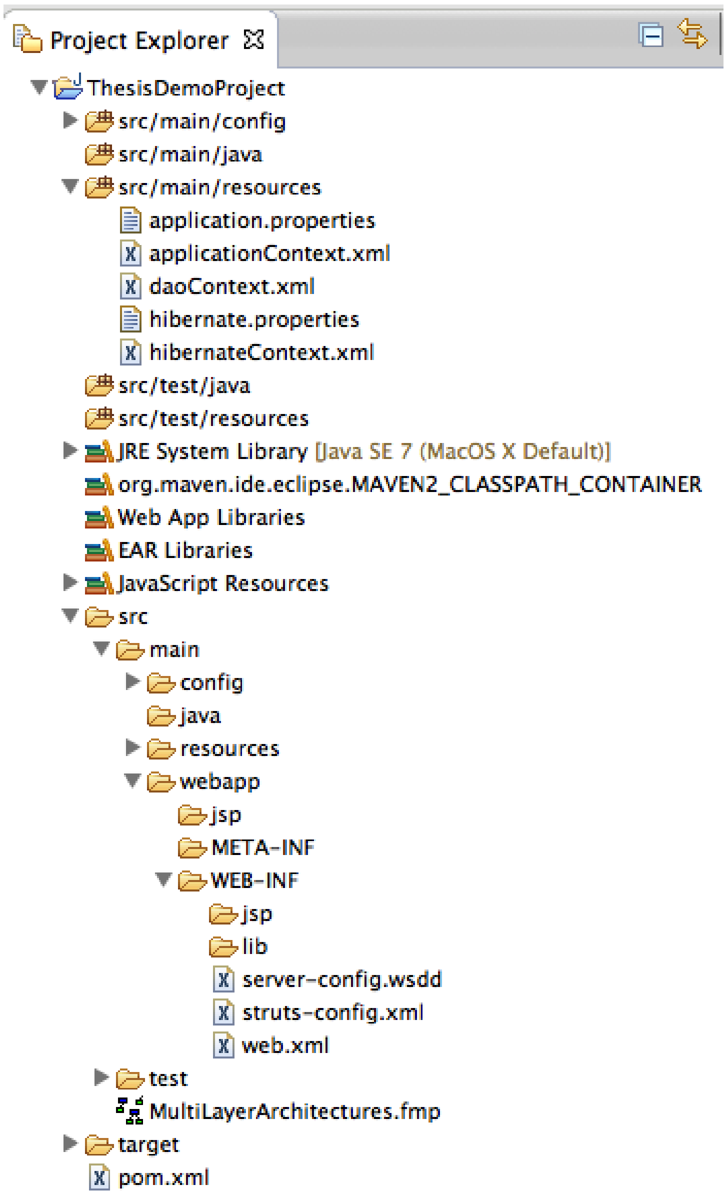
Figure 4.8: Framework selection page of the JACA wizard.

The use of these three pages serves a dual purpose in the tool. On the one hand, the complexity of the feature model is hidden to the tool users. As described in the previous chapter, the architectural decisions repository is contained in a feature model. By using this wizard, JACA users can create configurations of the feature model without knowing anything about it. On the other hand, these wizard pages allow the tool to become independent of ArchLayer. If JACA uses a feature model containing a configuration automatically generated by the model transformations described in Section 3.5, then the wizard pages will be automatically filled with the information found in the model and the tool will be used as the last step of ArchLayer. However, if JACA uses a feature model with no configuration, then the tool will be used independently as a code generation tool simplifying the development of multi-layer framework based applications but the architectural decision will have to be taken by the architect without any help from the tool.

At this point, the tool will read the framework information models pointed from the architectural decisions repository as described in the previous section. Specifically, the model to text transformation containing the initial configuration of each of the frameworks to be included in the new project. As described in Section 3.4, the initial configuration of a framework included in these models contain all of the elements needed by the framework to work properly in conjunction with the remaining frameworks being used.

From this information the tool will generate the complete project structure with the selected framework perfectly integrated and ready to begin the development. Figure 4.9 shows an example of a project structure automatically generated from the architectural decisions shown in the previous figures.

The generated project include all the dependencies and all the configuration files needed from the selected frameworks to work together. It also includes a version of the architectural decisions repository with a configuration showing the architectural decisions used to generate the project. This configuration can be later used to trace the architectural decisions in the source code of the developed project or to improve the repository itself, if significant differences are found between the information of how the different decisions affect the QAs of the system and its real behavior.



```
Project Explorer
▼ ThesisDemoProject
  ▶ src/main/config
    src/main/java
  ▼ src/main/resources
      application.properties
      applicationContext.xml
      daoContext.xml
      hibernate.properties
      hibernateContext.xml
    src/test/java
    src/test/resources
  ▶ JRE System Library [Java SE 7 (MacOS X Default)]
    org.maven.ide.eclipse.MAVEN2_CLASSPATH_CONTAINER
    Web App Libraries
    EAR Libraries
  ▶ JavaScript Resources
  ▼ src
    ▼ main
      ▶ config
        java
      ▶ resources
      ▼ webapp
          jsp
          META-INF
        ▼ WEB-INF
            jsp
            lib
            server-config.wsdd
            struts-config.xml
            web.xml
    ▶ test
      MultiLayerArchitectures.fmp
  ▶ target
    pom.xml
```

Figure 4.9: Example of a JACA multi-layer framework based project structure.

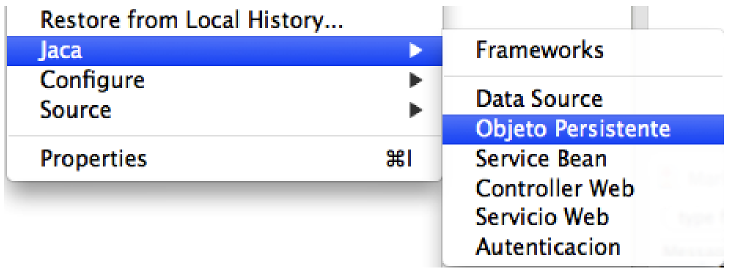


Figure 4.10: Available framework concepts implementation wizards for a set of frameworks.

The tool will keep working with this project structure to automatically generate part of the source code of the application being developed. Next section describes in detail how this code is generated.

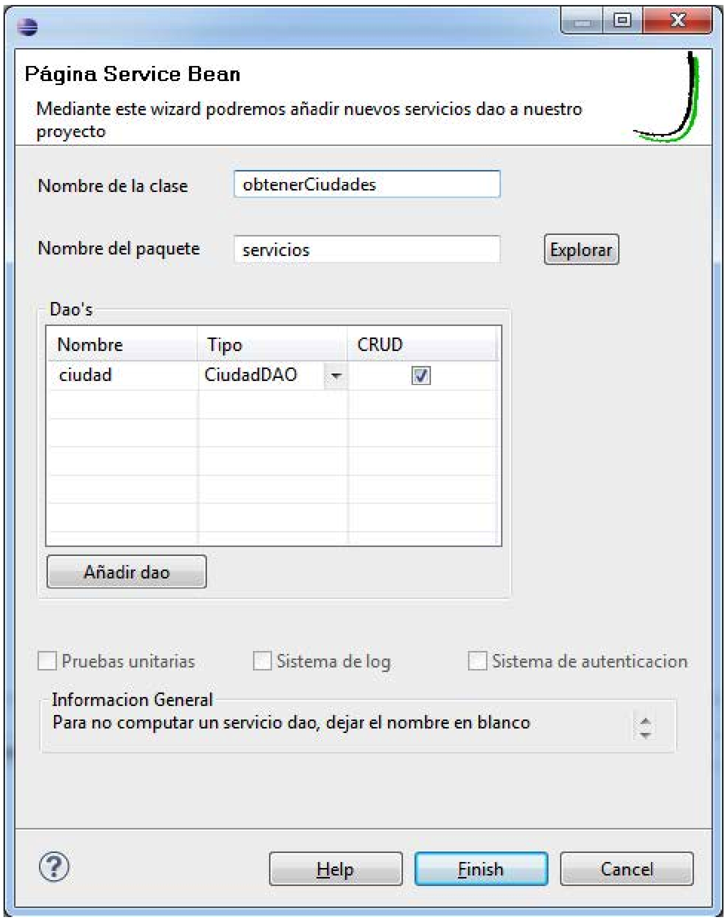
4.4 Concept implementation generation

From the project structure generated as described in the previous section, JACA can automatically generate part of the project source code. For this, the tool need the information contained in the framework information models. As described in Section 3.4, these model contains the templates used to generate the source code of the different concepts that can be implemented by each framework.

For each of the concepts that can be implemented by the frameworks selected for the development of the project, JACA provides the users a new wizard. Figure 4.10 shows the available wizards for the sample project shown in the previous section. Each time one of these wizards are executed, the source code of an instance of the corresponding framework concept will be automatically generated into the project. Also, an additional wizards is offered to the users allowing them to include a new framework for the development of the project, if it was not included from the start.

Each of the wizards presents to the user one or more pages to gather the information necessary to particularize the generated code for each instance of a concept. For example, Figure 4.11 shows the page of the wizard used to generate a new service in the business logic layer of the project.

This page gather the basic information of the service to be generated like the


Página Service Bean

Mediante este wizard podremos añadir nuevos servicios dao a nuestro proyecto

Nombre de la clase	obtenerCiudades
Nombre del paquete	servicios

Dao's

Nombre	Tipo	CRUD
ciudad	CiudadDAO	✓

Informacion General

Para no computar un servicio dao, dejar el nombre en blanco

Figure 4.11: JACA wizard for the generation of services in the business logic layer of a project.

service's name or the code package where it should be generated. Using this information, and the model to text transformations contained in the framework information model used to implement this layer, the tool will generate the source code of the new service as well as the configuration elements needed for the new element to be completely operational in the project.

To increase the utility of the code generated by these wizards, their functionality has been increased in two additional areas.

First, to generate a richer code wizards allow developers to link the new elements created with other existing elements in the same layer or in the lower layer. For example, the wizard to generate instances of the DAO design pattern allows the developer to connect the new instance with other existing DAO instances in the source code of the project. This allows developers to automatically generated the source code supporting the complex relations between tables in a relational database. Obviously, the code generation templates included in the framework information models should support this complex behavior in order for the tool to be able to provide it to its users.

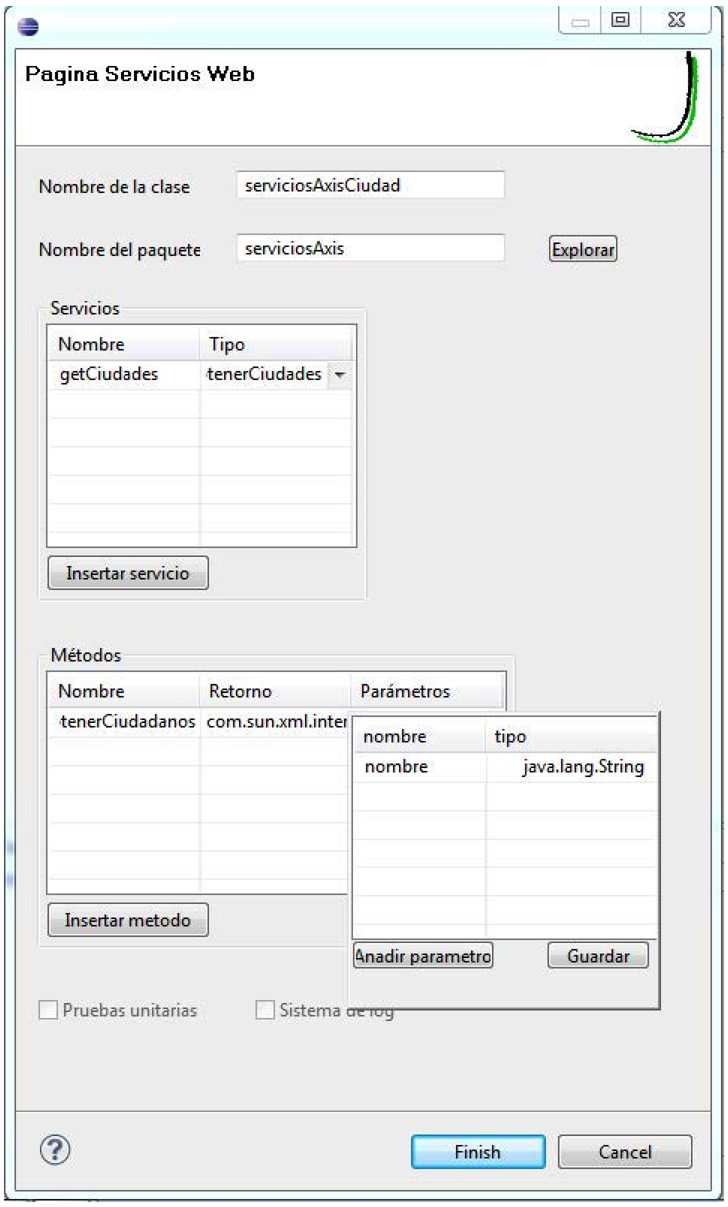
Other example of this relation between elements is shown in Figure refJacaBean-Wizard. The services of the business layer generated through this wizard could make use of the existing instances of the DAO pattern in the persistence layer. The wizard not only generates the association between the business service and the selected DAO instances and all the configuration needed for it to work in the project. It also allows developers to automatically transfer the CRUD operations of the DAOs to the business service. This is a very common operation in this type of project that allows these operations to be exposed to higher layers without breaking the multi-layer architecture design.

To implement these relationships, wizards sought in the project existing code to offer its users the existing elements that can be bound with the new element. The search of the wizards for possible elements to link is not limited to the items created automatically by the JACA wizards. It also recognizes elements created manually by the developers, as long as they have been developed following the same criteria used by the wizards. This provides great flexibility to the tool and makes the wizards useful in any development, regardless of whether it has followed the model driven process proposed in this thesis.

And second, some layers in multi-layer architectures can be considered as cross-cutting. These layers provide services or add functionality to the elements of all the other layers of the architecture. Example of this layers can be a test layer or a log layer. They do not provide any functionality of the project by themselves but complement the elements of the rest of the architecture.

JACA wizards support this kind of layers by complementing the elements generated for the functional layers. For example, the wizard for the generation of a web service shown in Figure 4.12 supports this feature. In addition to the basic information of the web service to be generated, the elements of the business logic layer with which the web service will be related and the methods that will be exposed as web services, the wizard provides users the ability to generate the corresponding elements for the testing and log layers. This possibility is shown as check-boxes in the bottom of the wizard page, as can be seen in the figure. If any of the check-boxes are selected by the user, the wizard will automatically generate the corresponding elements in the project. Again, the code generation templates included in the framework information models should support this complex behavior in order for the tool to be able to provide it to its users.

Using the different JACA wizards users can generate a significant portion of the source code of framework based multi-layer applications. Especially, the configuration and instantiation code of the different concepts implemented by frameworks. This greatly reduces the need for developers to have deep technical knowledge about the frameworks, it facilitates staff rotation and all this while keeping the software architecture flexibility.



Pagina Servicios Web

Nombre de la clase: serviciosAxisCiudad

Nombre del paquete: serviciosAxis

Servicios

Nombre	Tipo
getCiudades	tenerCiudades

Métodos

Nombre	Retorno	Parámetros
tenerCiudadanos	com.sun.xml.inter	

nombre	tipo
nombre	java.lang.String

Figure 4.12: JACA wizard for the generation of web services.

4.5 Additional material and use of the tool

As described above, JACA was designed as a standalone tool that can be used without having to follow ArchLayer. As such, the tool was developed for the regional government of Extremadura and integrated into its development environment.

To simplify the use of the tool in this manner, a relevant amount of additional material is provided accompanying. More information about the additional material can be found in Appendix D. Specifically, the additional material provided alongside the tool is the following:

- **Instruction manuals.** A set of instruction manuals are provided detailing in depth the use and maintenance of the tool. Specifically, three instruction manuals are provided:
  - *User manual.* This document includes all the information a user may need to correctly install and use JACA in the development of multi-layer framework based applications.
  - *Admin manual.* This document details the information needed to create and manage the models read by the tool, so an admin can remotely modify the tool behavior by changing the models contents or creating new models.
  - *Sysadmin manual.* This document is directed to the sysadmins who will maintain projects developed using JACA. It includes the relevant information about server configuration and other aspects needed for the correct functioning of the generated projects.
- **Frameworks guides.** For each of the development frameworks included in the list of approved technologies by the regional government of Extremadura, a best practice guide is provided. These guides detail technical information about the use of the frameworks and how they should be used in projects for the regional government.
- **General guides.** An additional set of guides is provided detailing general

aspects of the development of multi-layer applications for the government of Extremadura such as a coding conventions guide or a recommended development environment guide.

- **Models.** Finally, a set of models is provided. These models make JACA able to generate code for the approved frameworks following the guides mentioned above and meeting the requirements of the general guides.

With this material, the tool can be easily used by any company for the development of projects for the government of Extremadura. Although the tool is not a mandatory requirement for the development of applications for the government of Extremadura, its use simplifies them greatly. This is due to the application developments publicly hired by the regional government requiring the use of the frameworks integrated in the tool in the way described in the above mentioned guides.

Examples of this type of public hiring can be found in (Consejería Administración Pública, 2013) or (Centro de Información Cartográfica y Territorial de Extremadura, 2013). In both examples of technical specifications for hiring software development by the regional government, two annexes are included specifying the need for the use of the PCDAI and the Java development standard of the government of Extremadura. As described above, JACA is based and completely support such technical specifications.

4.6 Conclusions

In this chapter the tool developed for automatically generating part of the source code of framework based multi-layer web applications has been reviewed. This review has covered both possibilities when using the tool, the use as a last step of ArchLayer or the use as an independent tool for code generation. Later special attention has been paid to the additional material provided alongside the tool and how it can be used for the development of applications for the government of Extremadura.

The presented tool, as part of ArchLayer, helps to mitigate many of the addressed problems. By being integrated with the architectural decisions repository and the framework information models, JACA is able to generate a significant part of the source code of framework-based application without constraining the software architecture of the project. This integration with ArchLayer models also allows the tool to adapt itself to the technological evolution of developing frameworks.

Additionally, the centralized behavior of the tool when reading ArchLayer models and its code generation capabilities, both for the initial configuration of the frameworks and for the implementation of various concepts, make it suitable for its use in distributed development environments with high staff rotation.

The following chapter discusses the results obtained after applying the tool in conjunction with ArchLayer to commercial projects.

Chapter 5

Validation

You killed more people than anybody in history.

Be the best at whatever you do, that's what my mother always told me.

Speaker for the Dead, Orson Scott Card.

Throughout this thesis, ArchLayer, a process for the development of multi-layer framework based application, has been defined. This process was designed to be used by software development companies. Specially, by those companies facing the problems caused by distributed development and high staff rotation.

This chapter describes the validation performed to evaluate ArchLayer usefulness. To that end, it has been applied to two industrial projects. As a result of these projects, qualitative and quantitative information has been obtained on the use of the defined contributions regarding three different aspects: their feasibility, their completeness and the effort required to apply them. The rest of the chapter is organized as follows. Section 5.1 details the context in which ArchLayer was validated. Section 5.2 describes the characteristics, sub-characteristics and metrics defined to validate ArchLayer. Section 5.3 presents the industrial projects in which it has been applied. Section 5.4 specifies the results of the defined metrics. Section 5.5 details a summary of the results, the validity of the results and the lessons learned. Finally, Section 5.6 summarize this chapter focusing on the benefits provided by ArchLayer.

5.1 Validation context

This chapter tries to detail the usefulness of the model-driven, variability management and architectural decisions techniques presented in this thesis to facilitate the development of multi-layer framework based applications in the context of distributed development centers with high staff rotation. To validate the process and the contributions detailed in this thesis, they have been applied to two industrial project. Industrial projects were used instead of other validation methods (such as Controlled Experiments or Slice of Life (Shaw, 2002)) since, to properly ensure the impact and benefits of ArchLayer, reasonably large projects were needed. Therefore, it was deemed more appropriate to use real projects and validate ArchLayer in an environment as similar as possible to that in which it has been designed to work, rather than to validate it in an artificial environment or using a toy example.

The two projects used for the validation of ArchLayer were developed by Gloin. As stated in Section 1.6, Gloin is a company co-founded by the author of this thesis, its supervisor and another research partner. As such, one of the company's goals is to bring to the market the research advances. Therefore, two commercial projects were used to validate the process without causing any inconvenience to the company.

The two projects involved the development of a multi-layer framework based application, which fits perfectly with the context of this work. Otherwise, although Gloin is not a big development company, it also suffers the problems of high staff rotation and distributed development. High staff rotation because, due to its proximity to the university, the company has a scholarship program for final year students where computer science students spend a year learning in the company and then move on to other jobs. This scholarship program has the same effect in the company that high staff rotation has in bigger companies. And distributed development problems because most of the company clients are located outside of Extremadura and, usually, outside of Spain.

In each of the projects in which the contributions of this thesis have been applied the following characteristics, or validation goals, have been evaluated: their feasibility, completeness and the effort required to apply them.

- With the feasibility characteristics, the author of this thesis evaluated whether it is possible to use the developed repositories, models, transformations and tools to bridge the gap between the initial design and the implementation of framework based multi-layer applications in real projects with medium/large complexity in a company suffering staff rotation. Special attention is paid in the evaluation of these characteristic to the improvement provided over other existing techniques reviewed in Chapter 2.

- The completeness characteristics were defined to evaluate whether the application of ArchLayer, along with the tools that support it, allows developers to obtain a specific design of the application to be developed and a significant portion of its source code.

- The effort characteristics were defined to assess the amount of work needed to apply ArchLayer and the amount of effort saved during the design and development of the systems. For the evaluation of these characteristic estimates of the effort required by the projects were used. Although these estimates may not be totally accurate, they are based on historical company information on projects of similar complexity and size developed in the same context using a traditional development process.

Due to the generality, abstraction, and difficulty of measuring of the above characteristics, a set of more focused, specific and measurable sub-characteristics have been defined to better identify qualitative or quantitatively their compliance. These sub-characteristics are detailed in the following section.

5.2 Validation characteristics and sub-characteristics

In this section, the characteristics used to validate ArchLayer are refined into sub-characteristics more specific and easy to evaluate and verify their proper satisfaction. To perform this refinement, the Goal, Question, Metric (GQM) methodology was used (Basili, Caldiera, & Rombach, 1994; Van Solingen & Berghout, 1999).

GQM (Goal, Question, Metric) is an approach to software metrics that defines a measurement model on three levels:

- *Conceptual level (Goal).* A goal is defined for an object, for a variety of reasons, from various points of view and relative to a particular environment.
- *Operational level (Question).* A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.
- *Quantitative level (Metric).* A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

In this regard, the *Goals* are the validation characteristics previously defined, namely feasibility, completeness and the amount of effort required for application of ArchLayer. The *Questions* form the sub-characteristics defined to qualitatively or quantitatively evaluate the Goals compliance. Finally, *Metrics* have been defined to quantitatively measure the questions/sub-characteristics. The questions and metrics defined for each characteristic/goal are detailed in the following subsections.

5.2.1 Validation goal: feasibility

This goal tries to assess the feasibility of using the defined contributions in medium/large complex industrial projects by distributed development companies with high staff rotation.

To evaluate the feasibility of the contributions related to the architectural decisions that should be taken during the design of a framework based multi-layer architecture, the following questions and metrics were defined:

- *Question 1.1:* Is it possible to capture the architectural decisions involved in the design of a multi-layer framework based application in the architectural decisions repository?
  - *Metric 1.1.1:* Percentage of architectural decisions that can be captured with the presented contributions.
- *Question 1.2:* Is it possible to generate a set of the architectural decisions

taken during the development of a multi-layer application?

  - *Metric 1.2.1:* Percentage of architectural decisions that can be stored for future use with ArchLayer.

- *Question 1.3:* Does the management of architectural decisions provided by ArchLayer help the knowledge transfer in case of staff rotation?
  - *Metric 1.3.1:* Yes/No.
- *Question 1.4:* Does the management of architectural decisions provided by ArchLayer entail improvements over existing techniques?
  - *Metric 1.4.1:* Yes/No.

To evaluate the feasibility of the framework information model created to capture the technical knowledge about development frameworks, the following questions and metrics were detailed:

- *Question 1.5:* Can the knowledge needed to start using a framework in a multi-layer application be modeled?
  - *Metric 1.5.1:* Percentage of initial configuration knowledge that can be modeled.
- *Question 1.6:* Can the knowledge needed to implement a given concept using the different mechanism provided by a development framework be modeled?
  - *Metric 1.6.1:* Percentage of concept implementation knowledge that can be modeled.
- *Question 1.7:* Does the management of framework information provided by ArchLayer help the knowledge transfer in case of staff rotation?
  - *Metric 1.7.1:* Yes/No.
- *Question 1.8:* Does the management of framework information provided by ArchLayer entail improvements over existing techniques?
  - *Metric 1.8.1:* Yes/No.

With the aim of assessing the feasibility of using the defined model transformations for helping the architect in the design of a specific architecture, the following questions and metrics were specified:

- *Question 1.9:* Is it possible to suggest a feasible set of architectural decisions from the annotated initial design?
  - *Metric 1.9.1:* Percentage of architectural decisions that can be suggested.
- *Question 1.10:* Can a specific design be generated from the annotated design and the set of architectural decisions to be used?
  - *Metric 1.10.1:* Yes/No.
- *Question 1.11:* Does the model transformations provided by ArchLayer help the knowledge transfer in case of staff rotation?
  - *Metric 1.11.1:* Yes/No.
- *Question 1.12:* Does the model transformations provided by ArchLayer entail improvements over existing techniques?
  - *Metric 1.12.1:* Yes/No.

To evaluate the feasibility of the code generation tool developed to automatically generate framework specific code, the following questions and metrics were detailed:

- *Question 1.13:* Is it possible to automatically generate part of the source code of multi-layer applications from the specific design and the set of architectural decisions?
  - *Metric 1.13.1:* Yes/No.

Finally, in order to evaluate the feasibility of using the whole proposed process, the following questions and metrics were defined:

- *Question 1.14:* Do the users state that the process and its associated tools and techniques are usable?
  - *Metric 1.14.1:* Average usability of the process (from 0 to 10) according to the users.

5.2.2 Validation goal: completeness

This goal tries to asses whether the presented contributions facilitate the development of framework based multi-layer applications without restricting the software architect.

In order to evaluate whether the architectural decisions managed by ArchLayer are complete and can be used to generate a suitable architecture, the following questions and metrics were defined:

- *Question 2.1:* Are all the architectural decisions involved in the development of a multi-layer application available for the architect?
  - *Metric 2.1.1:* Percentage of architectural decision that are available.
- *Question 2.2:* Are all the architectural decisions involved in the development of a multi-layer application correctly suggested?
  - *Metric 2.2.1:* Percentage of architectural decision that are correctly suggested.

To assess if the framework technical knowledge stored in the framework information models is complete and can be used to design and develop framework based applications, the following questions and metrics were specified:

- *Question 2.3:* Is all the technical knowledge needed for the design and development of a framework based application stored in the framework information models?

  Metric 2.3.1: Percentage of technical knowledge stored in the available framework information models.
- *Question 2.4:* Is all the technical knowledge needed for the design and development of a framework based application automatically transferred to the design by the transformations?

  Metric 2.4.1: Percentage of technical knowledge transferred to the application specific design.

In order to evaluate whether the source code generation is complete and can be used to automatically generate framework based applications, the following questions and metrics were defined:

- *Question 2.5:* What amount of the application source code can be automatically generated?
  - *Metric 2.5.1:* Percentage of the application source code that can be automatically generated.

Finally, to validate the completeness of the whole process, the following question and metric was defined:

- *Question 2.6:* Is any additional information necessary to design and develop a multi-layer framework based application?
  - *Metric 2.6.1:* Yes/No.

5.2.3 Validation goal: effort

The goal tries to validate the effort required to apply ArchLayer and the amount of effort that can be saved in the design and development of framework based applications. To this end, the following questions and metrics were defined:

- *Question 3.1:* How much effort is needed for including new architectural decisions not available in the process?
  - *Metric 3.1.1:* Increase in effort (in man-hours and percentage) with respect to the estimations.
- *Question 3.2:* How much effort is needed for modeling the framework technical knowledge of a not available framework?
  - *Metric 3.2.1:* Increase in effort (in man-hours and percentage) with respect to the estimations.
- *Question 3.3:* How much effort is needed for adapting the process model transformations to new architectural decisions or frameworks added to the process?

  - *Metric 3.3.1:* Increase in effort (in man-hours and percentage) with respect to the estimations.

- *Question 3.4:* How much additional effort is needed for the overhead caused by using ArchLayer?

  - *Metric 3.4.1:* Increase in effort (in man-hours and percentage) with respect to the estimations.

- *Question 3.5:* How much effort is saved in the design and development of a framework based multi-layer application by using ArchLayer?

  - *Metric 3.5.1:* Decrease in effort (in man-hours and percentage) with respect to the estimations.

- *Question 3.6:* What is the return on investment (ROI) of applying ArchLayer?

  - *Metric 3.6.1:* M3.1.1 + M3.2.1 + M3.3.1 + M3.4.1 + M3.5.1.

5.3 Industrial projects

This section briefly outlines the two projects in which ArchLayer has been applied. The two projects are industrial applications of medium/large size and with a medium/high development complexity due to the complexity of their architectures and the technologies used.

5.3.1 BeeFun

BeeFun[1] is a project developed by Gloin applying the latest research made by employees of the company in Cloud Computing and Mobile Development. BeeFun is an application for mobile devices that, in contrast with the existing ones, allows its users to send messages to non static group of users, where the users in the group depend on their contextual information (such as where they are now or what are their preferences). Thus, making group messaging more efficient and direct. Additionally, the same app and its infrastructure can can be used by mobile marketing

[1]http://beefunapp.com/

companies to send ads to users based on their contextual information. Thus, these ads will only reach the users interested on them; in return, the users can receive different rewards for each received message.

This product is divided into two applications, a mobile app and a server application. ArchLayer was applied to the later, since it is a framework based multi-layer application. To develop this application, the company had to gain a significant amount of knowledge about mobile applications and mobile marketing. All this knowledge had to be included to the different artifacts involved in the process, as will be shown later.

Following are the most important requirement that the system ought to meet: the application should manage the information concerning the mobile users connected to the server, it should allow communication between those users by employing predefined message templates and geolocated messages, and it should rank the mobile application users based on the points obtained by their use of the services provided by the server.

The following are the most important business goals identified: the users/mobile devices contextual information is private and never had to come out of the mobile devices and the system had to be efficient in the use of the mobile resources. The main business processes identified are: to gather the contextual information, to create predefined messages or ads, to send a message to a group of users and to get the rewards.

Additionally, some quality attributes were defined for the application. First, the server application should be reliable since it would be used by a commercial product and, therefore, should have a high up-time. Second, the server application should be secure since it would manage users' private information. And third, it should be efficient since mobile application users tend to be very impatient with anything but fast response times from servers.

Based on these requirements and QAs the application was developed using the contributions of this thesis as follows:

- The initial design of the system included the functional requirements of the system and the QAs affecting them in an UML model ready to be used in ArchLayer.

- This initial design was used in combination with the architectural decisions repository, the model transformation and the framework information model to obtain a specific design of the system tailored to the software architecture and development technologies chosen by the architect
- The obtained specific design was used to automatically generate part of the source code of the application and as the guide for the rest of the development.
- Finally, the set of decisions taken by the architect in the project was used in conjunction with the results obtained to improve the company historical data and to be used in future developments.

5.3.2 NimBees

NimBees[2] is another project developed by Gloin aimed at developing an API facilitating the implementation of a new kind of mobile devices apps. This API implements a new service provisioning model known as *People as a Service (PeaaS)* (Guillén et al., 2014). This new model transforms mobile devices into Cloud Platforms providing services on the contextual information of the mobile devices and their owners (such as their personal data, the visited web pages, the places they have been, when they have been there, etc.) or inferences obtained from it (such us, for example, their preferences depending on the visited web pages). Thus, the applications or services interested in these data can easily access them calling one of the services already deployed in the cloud enabled mobile device. In exchange for this information, owners gain economic or social benefits (such as a better control of sick/elder people through their mobile device). In any case, the mobile device owners always can manage what information they want to provide and to whom.

This product is also divided into two applications, a mobile API to be used in the development of mobile apps and a server application. ArchLayer was applied to the later, since it is a framework based multi-layer application. Additionally, in the

[2]http://www.nimbees.com/

Table 5.1: Summary of the project features.

Project	BeeFun	NimBees
Use Cases	40	35
Non-Functional Requirements	6	4
Use Case Points (UCP)	566	377
Architecture Design	1412	880
Implementation and Tests	3960	2640

development of this product the company could reuse most of the knowledge about mobile development acquired during the development of BeeFun.

Following are the most important requirement that the system ought to meet: it should gather data from the mobile device sensors and its usage, users preferences should be infered from the information gathered and only authorized services or applications should access such information. The QAs that the system should meet are similar to the ones of the previous application, specifically: the server application should be reliable, it also should be secure and efficient.

The results of this thesis were used in the development of this system in the same way they were used in BeeFun. ArchLayer was used to develop the application starting from the initial design including the QAs information.

5.3.3 Features of the industrial projects

Table 5.1 shows a summary of the features of each project. It details the number of use cases and non-functional requirements per project. It also includes the number of Use Case Points (UCP) (Clemmons, 2006) of each project in order to show their complexity and to calculate estimations of the effort required to develop them (as UCP is the technique used in the company to calculate these estimations). Finally, the table also include the real effort in man-hours devoted in the architecture design and implementation and test stages.

5.4 Validation results

In this section, an analysis of the contributions of this thesis is done with respect to the feasibility of applying them, the completeness of the generated information and artifacts, and the effort that entails their utilization. This analysis has been made applying the questions and metrics listed above on the described industrial projects.

5.4.1 Validation goal: feasibility

Below the results of applying the questions assessing the feasibility of the contributions are detailed.

Question 1.1. Is it possible to capture the architectural decisions involved in the design of a multi-layer framework based application in the architectural decisions repository?

Metric 1.1.1. Percentage of architectural decisions that can be captured with the presented contributions.

All the architectural decisions taken during the development of both applications were reflected in the architectural decisions repository. Thanks to the flexibility of the repository, every layer, design pattern and framework used in the development could be included in it, even in those cases that had not been contemplated during its creation. The development of these applications implied the extension of the repository with a new layer for mobile communications, including design patterns and technologies for sending push notifications and text messages. Also additional patterns and technologies were added to the existent security layer to support OAuth authentication, an open standard for authorization widely used in mobile applications. All these new elements were added to the architectural decisions repository, and consequently to the rest of the process, without making any changes to it.

Question 1.2. Is it possible to generate a set of the architectural decisions taken during the development of a multi-layer application?

Metric 1.2.1. Percentage of architectural decisions that can be stored for future *use with the detailed proposal.*

Once all the architectural decisions were taken and the initial configuration code of the projects were generated, a set of all the architectural decisions were automatically generated. As described previously, this set of decisions takes the form of a configuration of the feature model that composes the architectural decisions repository. The generated configuration included all the architectural decisions taken during the project development, including the new elements added to the repository, as described in the previous question. This architectural decision set is stored alongside the project with two main goals, serve as documentation of the development of the project and be used to improve the repository itself if the behavior of the application does not match the expected.

Question 1.3. Does the management of architectural decisions provided by ArchLayer help the knowledge transfer in case of staff rotation?

Metric 1.3.1. Yes/No.

Between the completion of the first and second project a couple of significant staff changes happened in the company. First, one of the main software architects involved in the development of the first project left the company after finishing it and before starting the second project. Also, due to the company scholarship program part of the development team was different in the two projects. During the development of the second project both, the architectural decision repository and the set of architectural decisions taken during the first project, greatly helped the architects to understand the rationale behind the software architecture of the first project and to take advantage of it for the second project.

Question 1.4. Does the management of architectural decisions provided by ArchLayer entail improvements over existing techniques?

Metric 1.4.1. Yes/No.

Most of the existing techniques for the development of multi-layer applications do not provide any mechanism for the management of architectural decisions. Architectural decisions in these proposals are inherent to the models used and can not be affected by the architect. Only the WebSA proposal (Meliá & Gómez, 2006) provides support for architectural variability. In this case, ArchLayer represents an improvement over WebSA due to the depth of the architectural decisions managed, WebSA does not include the technologies to be used in the development as architectural decisions, and to the possibility of storing the decisions taken for future reuse which, as stated above, helps to reduce the effects of staff rotation.

Question 1.5. Can the knowledge needed to start using a framework in a multi-layer application be modeled?

Metric 1.5.1. Percentage of initial configuration knowledge that can be modeled.

All the information needed to automatically generate the initial configuration of all the frameworks used in both projects was modeled using the framework information meta-model presented in this thesis. Even the information of the new frameworks included specifically for these project, that were not considered during the process definition, could be modeled. Thanks to the flexibility of the framework information meta-model, the initial configuration of both project could be automatically generated with all the frameworks involved perfectly integrated.

Question 1.6. Can the knowledge needed to implement a given concept using the different mechanism provided by a development framework be modeled?

Metric 1.6.1. Percentage of concept implementation knowledge that can be modeled.

All the information needed to automatically generate the implementation of the different concepts supported by the frameworks used in both project was modeled using the framework information meta-model. Even the information of the new frameworks included specifically for these projects could be modeled. Thanks to the flexibility of the framework information meta-model, a significant amount of the code needed to implement the concepts supported by all the frameworks involved could be automatically generated and integrated in the system architecture, even using the different mechanism which a framework could have to implement a given concept.

Question 1.7. Does the management of framework information provided by ArchLayer help the knowledge transfer in case of staff rotation?

Metric 1.7.1. Yes/No.

By modeling the knowledge needed to use development frameworks, ArchLayer greatly helps the knowledge transfer in case of staff rotation. Mainly, the modeled information simplifies the process of learning how to use a given framework to the people who joined the company to develop the second project. But also, all contributions added to the framework information models during the first project could be easily reused in the second, even when the original author did not continue working for the company.

Question 1.8. Does the management of framework information provided by ArchLayer entail improvements over existing techniques?

Metric 1.8.1. Yes/No.

Most of the existing techniques for simplifying the use of development framework are focused on the developers learning process. Techniques like (Antkiewicz et al., 2009), that includes support for the automatic generation of framework completion code, are surpassed by ArchLayer since they only focus on a single framework while ArchLayer provides support for multiple frameworks in the same project.

Question 1.9. Is it possible to suggest a feasible set of architectural decisions from the annotated initial design?

Metric 1.9.1. Percentage of architectural decisions that can be suggested.

Using the model transformations that are part of ArchLayer, every layer, design pattern or framework that is included in architectural decision repository can be automatically suggested to be used in the development of a project. Transformations had to be updated to include the new architectural decisions included in the repository, as described in Question 1.1. The feasibility of the suggested decisions depends on the criteria used in the transformation to automatically suggest each one of them. Every decisions automatically suggested by the transformation during the development of both projects was suitable and, therefore, carried out. Otherwise, some architectural decisions that were taken during the development were not automatically suggested by the model transformation. This may indicate that the selection criteria are too strict. However, a broader criteria may lead to false positives.

Question 1.10. Can a specific design be generated from the annotated design and the set of architectural decisions to be used?

Metric 1.10.1. Yes/No.

Using the model transformations that are part of ArchLayer, a specific design tailored to the software architecture and the set of technologies chosen by the architect was automatically generate for both projects. The generated design was obtained from an initial design, including the annotations relating the systems QAs with the funcitional requirements, and the set of architectural decisions. The specific design obtained included all the information needed for the initial configuration of the project and for the different implementations of all the frameworks supported concepts. This design can be used to automatically generate a significant part of the code and as a very detailed documentation for developers.

Question 1.11. Does the model transformations provided by ArchLayer help the knowledge transfer in case of staff rotation?

Metric 1.11.1. Yes/No.

By automatically suggesting a set of feasible architectural decisions and by generating a specific design for such decisions, the model transformations provided reduce the training cost of the new staff that joined the company for the development of the second project. Additionally, the knowledge stored in the transformation, like the layer suggestion criteria, is easily reused even when the original author leaves the company.

Question 1.12. Does the model transformations provided by ArchLayer entail improvements over existing techniques?

Metric 1.12.1. Yes/No.

Most model-driven techniques for the development of multi-layer applications only take as input of their model transformations the design of the application. ArchLayer, by additionally taking as input of the transformation the architectural decisions and the information about the framework, can generate a more varied output considering the variability of architectures and technologies. The transformations of the WebSA proposal (Meliá & Gómez, 2006) also take as input the architectural decisions, however they do not take into account the technological variability and are limited to the development of multi-layer web applications and RIAs, while ArchLayer can be used for the development of any multi-layer application.

Question 1.13. Is it possible to automatically generate part of the source code of multi-layer applications from the specific design and the set of architectural decisions?

Metric 1.13.1. Yes/No.

In both analyzed projects, a significant part of the applications source code could be automatically generated using the specific design obtained through the process and the code generation tool. Specifically, all the initial configuration of the project and a large part of the framework concept implementation code needed to implement the system requirements was automatically generated.

Question 1.14. Do the users state that the process and its associated tools and techniques are usable?

Metric 1.14.1. Average usability of the process (from 0 to 10) according to the users.

In every project the development team was surveyed to obtain its opinion about the process usability. During the development of the first project the process received a lower score on this aspect that during the development of the second project. The lower score was caused by the need to update most process artifacts due to the inclusion of new architectural decisions. This made the use of the process cause a bigger overhead on the development, hence the worse punctuation. During the development of the second project this overload did not occur, since all the necessary information was included in the process during the first project, hence the better punctuation. The fact that the process was used for the first time by most of the team during the development of the first project also influenced the outcome. Regarding the tools, all developers indicated that the tools were usable and that they facilitated the implementation of the process. However, some usability complains were raised about the different tools, specially the modeling tools. These complains can be attributed to the use of research tools that does not have the stability and usability of the commercial tools commonly used by developers.

Summary

Table 5.2 shows a summary of the metrics obtained applying the feasibility questions on the two industrial projects.

5.4.2 Validation goal: completeness

Below the results of applying the questions assessing the completeness of the contributions are detailed.

Question 2.1. Are all the architectural decisions involved in the development of a multi-layer application available for the architect?

Metric 2.1.1. Percentage of architectural decision that are available.

The answer to this question is entirely dependent on the project being developed and the technologies that the architect choose to use. For the analyzed projects, the percentage of architectural decisions available was quite high. During the development of the first project, over 80% of the architectural decisions where available. The decisions missing were those related to the communication with mobile application. During the second project all the architectural decisions were available, since those related with mobile communication where added during the first project. However, due to the nature of ArchLayer, this percentage can reach zero, as software architectures and technologies evolve and different kinds of projects are addressed with this architectures. Therefore, for the completeness of the process, it may be considered more relevant the ease to include new decisions, which has already been discussed, than the actual architectural decisions available.

Question 2.2. Are all the architectural decisions involved in the devel-

opment of a multi-layer application correctly suggested?

Metric 2.2.1. Percentage of architectural decision that are correctly suggested.

As stated above, the suggested decisions depends on the criteria included in the model transformation. ArchLayer has been designed in a way that allow these criteria to be tailored to the need and preferences of each company or development team using the process. For the projects studied, in the first case lower results were obtained for two main reasons. First, as has been already mentioned, a new set of architectural decisions had to be added to the process related to the communication with mobile devices which caused a smaller number of correct suggestions. And second, since this was the first time that the Gloin development team used ArchLayer, the model transformation were not fine tailored to their architects preferences. Better results were obtained in the second problem since these problems were mitigated.

Question 2.3. Is all the technical knowledge needed for the design and development of a framework based application stored in the framework information models?

Metric 2.3.1. Percentage of technical knowledge stored in the available framework information models.

All the technical knowledge about the frameworks used in the development of the two projects was stored in the framework information models. Evidently, new models had to be created for the new technologies that were not originally included in the process. However, the flexibility of the framework information meta-model allowed the company architect to model all the technical knowledge needed about the initial configuration and concept implementation of the new frameworks.

Question 2.4. Is all the technical knowledge needed for the design and development of a framework based application automatically transferred to the design by the transformations?

Metric 2.4.1. Percentage of technical knowledge transferred to the application specific design.

Likewise the previous question, all the technical knowledge about the frameworks used in the development of the two projects were automatically transferred to the specific design by the model transformations. In this case, the transformation to automatically generate a layered design from the initial design need to be changed to accommodate the new layer added to the process. However, it was not necessary to modify the transformation to obtain the specific design from the layered design, since all the technical knowledge was already included in the framework information models and the transformation get all the information needed from them.

Question 2.5. What amount of the application source code can be automatically generated?

Metric 2.5.1. Percentage of the application source code that can be automatically generated.

The code generation tool included in ArchLayer was used in the two projects to automatically generate a significant part of the applications source code. Specifically, around 30% of the applications code was automatically generated. This value may seem small compared to other proposals for automatic code generation. However, there are two important aspects to be taken into account when interpreting this data. First, the amount of code that can be automatically generated is usually a trade off between technological variability and automation level. ArchLayer was designed focused on variability, greater amounts of code could be generated in exchange for losing some flexibility. And second, the code automatically generated was focused on framework implementation concepts and the initial configuration and integration of the frameworks. This kind of code causes many of the more commons problems when working with frameworks, as was seen in some of the works discussed in Section 2.3. Therefore, although the amount of generated code is not very large, it is really relevant for the project developers.

Question 2.6. Is any additional information necessary to design and develop a multi-layer framework based application?

Metric 2.6.1. Yes/No.

ArchLayer focus on the architectural decisions that need to be taken during the development of a framework based multi-layer application and the technical knowledge needed to use such frameworks. Keeping this in mind, much more information is needed for the development of one of this applications. Especially, a lot of information is needed about the functionality to be provided by the application. A significant part of this information is included in the design of the application used by the process, however additional low-level information is needed for the final implementation of the application.

Summary

Table 5.3 shows a summary of the metrics obtained applying the completeness questions on the two industrial projects.

5.4.3 Validation goal: effort

Below the results of applying the questions evaluating the effort of using ArchLayer, as well as the return of investment obtained, are detailed.

The increase/decrease of effort in each development stage is calculated by comparing the real effort devoted to the development of each project (using the process) with the estimation of the effort that should have been spent according to the historical data of the company (without using the process). These data show that the productivity by UCP is 14 hours, of which 2,8 hours correspond to architecture design and 7 hours to implementation and testing.

Question 3.1. How much effort is needed for including new architectural decisions not available in the process?

Metric 3.1.1. Increase in effort (in man-hours and percentage) with respect to the estimations.

The need to add new architectural decisions to the decision repository led to an increase in the effort spent during the development of the first project. The effort needed to include the mobile communication decisions was of approximately 60 man/hours. This supposed an increment of slightly more than 1% over the company estimations for the design and development phases of the project. Since all the decisions needed where included in the process during the first project, no additional effort was required for this matter during the second project.

Question 3.2. How much effort is needed for modeling the framework technical knowledge of a not available framework?

Metric 3.2.1. Increase in effort (in man-hours and percentage) with respect to the estimations.

As has been stated before, new frameworks were to be added to the process for the development of the first project. The effort needed to include the three frameworks that were needed for the first project was of approximately 250 man/hours. This supposed an increment of 4.5% over the company estimations for the design and development phases of the project. Again, since all the frameworks needed where included in the process during the first project, no additional effort was required for this matter during the second project.

Question 3.3. How much effort is needed for adapting the process model transformations to new architectural decisions or frameworks added to the process?

Metric 3.3.1. Increase in effort (in man-hours and percentage) with respect to the estimations.

Adapting the model transformations that are part of the process to the new architectural decisions and frameworks added during the development of the first project meant an increment of approximately 20 man/hours. Because the majority of the information is not hardcoded on the transformations, their adaptation is quite simple. This increase represented less than 0.5% over the company estimations for the design and development phases of the project. Again, since all the transformations where adapted during the first project, no additional effort was required for this matter during the second project.

Question 3.4. How much additional effort is needed for the overhead caused by using ArchLayer?

Metric 3.4.1. Increase in effort (in man-hours and percentage) with respect to *the estimations.*

The use of ArchLayer in this thesis involves an overhead on the development time. This overhead is due mostly to the time spent by the development team in learning how to use the process and its associated tools, but it is also due to some of the task imposed by the process, such as having to relate the initial design of the application with the architectural decisions. For the analyzed project the overhead was 120 and 80 hours respectively. An increase of just over 2% of the initial estimations in both cases. The overhead was lower in the second project mostly due to the experience on using the process the development team acquired during the first project.

Question 3.5. How much effort is saved in the design and development of a framework based multi-layer application by using ArchLayer?

Metric 3.5.1. Decrease in effort (in man-hours and percentage) with respect to the estimations.

Due to the benefits provided by ArchLayer and the techniques and tools used, a significant amount of effort can be saved during the design and development of multi-layer applications. For the studied projects the effort saved was more than 8% of the estimations. Specifically, 475 hours were saved during the first project, over 8.5%, and almost 320 during the second project, over 8.5%.

Question 3.6. What is the return on investment (ROI) of applying ArchLayer?

Metric 3.6.1. M3.1.1 + M3.2.1 + M3.3.1 + M3.4.1 + M3.5.1.

Considering all the above aspects, the return on investment achieved by the application of ArchLayer in the design and development of the two applications analyzed resulted in savings of 25 and over 200 man/hours respectively. These data translate to a savings of almost 0.5% of the budget for the first project and over 6% for the second project. Applying the usual rates of the company, the savings resulting from the application of process were almost 1.000 € in the first project and almost 9.000 € in the second project.

Summary

Table 5.4 shows a summary of the metrics obtained applying the effort questions on the three industrial projects.

5.4.4 Further observations

This section describe additional interesting findings that where observed during the application of ArchLayer in the industrial projects.

First, it is interesting to note that, due to the size and complexity of the projects developed during the validation of ArchLayer, it is not possible to provide quantitative comparisons with other proposals. As stated above, the effort needed to develop any of the projects exceeds the three thousand hours of work, making it unviable to duplicate the work using other proposals. As an alternative to this lack of quantitative comparison, a qualitative comparison was performed highlighting the advantages provided by ArchLayer over other existing proposals in the context of framework based multi-layer applications.

Another important observation was the reduction in the number of bugs detected during the test phase of the development. The achieved reduction was over 10% with respect to historical data obtained from other projects. An analysis of this information showed that a significant part of the bug reduction can be attributed to the automatically generated framework code and the more detailed design provided to developers by using the process tools.

Additionally, interviews were conducted with the software architects and developers involved in the projects halfway through the development and at their end in order to obtain qualitative information about the proposal. These interviews consisted of open-ended questions with the following objectives:

- To identify whether ArchLayer provides relevant information to the development team.
- To identify whether the use of ArchLayer provides any relevant improvement in the design and development of multi-layer applications in distributed development centers with high staff rotation over other methods previously used.

- To get information about the proposal's user experience.
- To obtain some suggestions and improvements that might be incorporated into the work.

In these interviews, software architects indicated that ArchLayer was sufficiently relevant since it provides important information and simplify the task of tailoring the initial design of applications to a framework based multi-layer design, which is usually hard for them specifically in the fast evolving world of development frameworks. Also, they stated that the use of the process helped them to design more appropriate architectures thanks to the specific information about how the frameworks affect system quality attributes and the provided model transformations. Furthermore, in those cases in which the architectural elements to be used were not in the repository, the additional work of incorporating them into the process earned the developers a better understanding of them, and their documentation will make reuse simpler in future projects.

Developers noted that the architecture designed was very detailed and easy to implement, specially taking into account the generated code integrating the different frameworks of a project. However, they had some complaints about some of the techniques and tools imposed by the process since they had not previous experience working with them and they are not commercial quality tools.

These observations show the benefits provided by ArchLayer. They also shown that more efforts have to be devoted to improve the stability and usability of the tools.

5.5 Discussion

This section summarizes the results obtained, the threats to the validity of the results and the lessons learned during the application of ArchLayer in the industrial projects.

5.5.1 Summary of results

The aim of the validation previously presented was to assess the feasibility, the completeness and the effort needed for the application of ArchLayer. For each of these validation goals, statistical data has been used to confirm the validity and the benefits of the proposal.

The results obtained evaluating the feasibility of the proposal were very positive. Every architectural decision involved in the development of a multi-layer framework based application was correctly modeled, and all the decision taken were stored to be used as input for future projects.

Additionally, all the technical knowledge needed to correctly use development frameworks was also correctly modeled. Furthermore, all this information was used to suggest a set of architectural decision, generate a specific design for such decisions and generate a significant amount of source code.

Finally, all this contributions helped to reduce the drawbacks caused by the staff rotation that happened during the development of the two project and they improved the existing techniques for the development of framework-based multi-layer applications.

The only feasibility drawbacks were found on the usability of the process and its associated tools and techniques. Some of the detected problems were fixed, but additional effort is needed in that regard.

In general, the data collected from questions **Q1.1 to Q1.14 strongly support that the elements of ArchLayer are feasible**, i.e., the process can be applied to real-life examples by averagely trained personnel.

The results obtained assessing the completeness of ArchLayer were encouraging. The validation results showed that the elements involved in the process were complete and very useful.

Most architectural decisions were available for the architect, and a significant amount of them were automatically suggested. All the technical knowledge was available in the process models and automatically transferred to the specific design of the application. Also, a significant amount of the application source code was generated from the rest of the artifacts involved in the process.

As discussed before, the goal of ArchLayer never was to include the complete range of development frameworks, but provide a mechanism flexible enough to admit all of them. This flexibility has been proved by the inclusion of a complete new layer and several new development frameworks that were not initially taken into account.

However, this flexibility has some disadvantages, namely the project artifacts will never be complete because there will always be a new technology to add and there is trade off between the supported flexibility and the amount of source code that can be generated. ArchLayer can be fine tuned to find the balance preferred by each company between these two aspects.

Summarizing, the data collected for questions **Q2.1 to Q2.6 strongly support that the process elements are complete**. The process facilitates the use of a broad range of architectural decisions and development frameworks, which is very useful for the development multi-layer applications.

The results obtained assessing the effort required to use ArchLayer are very promising. The use of the process itself causes a small overhead in the effort needed, but its effect is diluted in the time saved during development.

Additional effort are needed when new architectural decision or technologies have to be included into the process. These additional effort can be a major drawback using the process, as in the first developed project where the benefits provided by the process are almost outweighed by the additional efforts needed.

However, taking into account the context for which the process has been developed, distributed development centers with a large number of projects, this risk is mitigated. The potential benefits of the process are shown more clearly in the second project, where the additional efforts done during the first one could be reused.

Concluding, the data collected for questions **Q3.1 to Q3.6 strongly support the effort needed characteristic**, indicating that the use of ArchLayer reduces the total effort spent in the design and development of multi-layer applications.

5.5.2 Threats to validity

ArchLayer was evaluated in two industrial projects. Data was collected to evaluate its feasibility, completeness and the effort needed to apply it. In this section, the possible threats to validity are discussed according to the four types of possible threats reported by (Wohlin et al., 2000).

Construct validity

Construct validity is concerned with the relation between a theory and its observation. Threats to construct validity refer to the extent to which the setting of an empirical study actually corresponds to the construct under study. In this thesis, this validity is related to the historical data of the company, the artifacts generated during the projects development and the researcher's observation. The main source of information used during the validation was the historical data on the productivity of the development teams. Therefore, the threat to the validity of this data was solved. The artifacts generated during the projects development, and from which the results of the metrics have been obtained, have been used for the documentation and implementation of the two systems. Therefore, it can be implied that these artifact are valid and correct. Finally, researcher's expectancies are considered to have been properly addressed because both positive aspects and negative aspects of the methodological approach were discovered and reported as result of the validation.

Conclusion validity

Conclusion validity is concerned with the relationship between a treatment and the conclusions drawn from it. Threats to conclusion validity refer to the ability to draw correct conclusions about relationships between the treatments and the results of an empirical study. In this thesis, this validity is related to the metrics and data obtained answering the questions. The data obtained during validation are objective since they are real values acquired by analyzing the generated artifacts. Only the questions measuring the effort of applying the process and the return on investment obtained are based on the comparison of the real effort data with estimations. The accuracy of such estimations may involve a threat to the validity of the results. Nevertheless, large deviations of the estimated effort are unlikely, due to experience of the company in making this kind of estimations.

Internal validity

Internal validity is concerned with the causal relationship between a treatment and its results. Threats to internal validity refer to discovery of a causal relationship in an empirical study that does not exist. In this thesis, these threats are related to truth of the metrics obtained, and the application of the methodology. The results obtained during the validation of this methodology have been derived from analysis of the artifacts generated during the development process. The generation of these results may have been influenced by the support provided by the researchers to the development team during the application of the process. Nevertheless, this support diminished in the second project and the results obtained in this project are better than in the first one.

External validity

External validity is concerned with the generalization of the conclusions of the validation. Threats to external validity refer to the ability to generalize the results and conclusions beyond the setting of the study. In this thesis, this validity is related to the kind and complexity of the industrial projects. The two industrial projects were of medium/large size and complexity, but with some differences. BeeFun required that a lot of new elements were added to the process. NimBees was a medium sized project based on a known domain with technologies already used.

5.5.3 Lessons learned

The validation of ArchLayer in two industrial projects revealed its strengths and weaknesses. The main strengths have been already discussed: control over the architectural decisions without restricting the architect, less technical knowledge required about the development frameworks, more detailed specific design tailored to the architectural decisions and development frameworks chosen by the architect and automatic code generation. The main weakness identified were: the effort needed to include new architectural decision or development framework to the process, the effort needed to keep all the elements of the process synchronized when one of this elements is included and the need to kept the process constantly updated.

ArchLayer was designed from the begging focused on flexibility. One of the main goals was always to support any new architectural decision or technology that may arise. The fulfillment of this goal has been demonstrated during the validation presented in this chapter, in which a complete new layer with all its related technologies was added to the problem without any modification. However, this flexibility is obtained in exchange for an increase in process complexity. Adding new elements requires, not only a deep knowledge of the elements being added but a intimate knowledge about the process artifacts. This makes the effort to include new elements to the project just equals to the effort reduction that it brings, making only profitable for the company to include elements that are going to be used in more than one project.

Furthermore, the close connection between the different artifacts of the process further complicate the inclusion of new elements. Each modification in any artifact of the process has to be followed by modification to all the related artifact for the process to keep working. For example, if a new technology is added to the architectural decisions repository then the models transformations have to be modified accordingly to be able to suggest it. But, then the information model is needed to generate the specific design based on the new technology and templates are needed in order to automatically generate source code for it. Keeping all these elements synchronized is a difficult and error prone task. Again, this problem is mitigated by the fact that the modifications should be done only once and they will be shared by all future projects.

Finally, and due to the constant technological evolution, the process has to be continuously updated to include new technologies or new versions of existing ones if it is not to get outdated. This problem is inherent to the nature of the technologies used and it is mitigated by the flexibility of the process to support new elements.

5.6 Conclusions

This chapter has presented the validation of the contributions of this dissertation. They have been applied to two real industrial projects in which the feasibility, the completeness, and the effort required to apply ArchLayer was evaluated.

This validation demonstrated that variability management and model driven engineering techniques can be applied to improve the development of framework based multi-layer applications. Specifically, ArchLayer allows developers to model the architectural decisions involved in the development of one of this applications and the technical knowledge needed to implement it using a set of development frameworks. This information was used to generate a specific design of the application and significant amount of its source code was automatically generated. Applying this process, a reduction of more than 6% of the estimated effort was achieved in a development were all the technologies and decisions were already included in the process.

Finally, this validation also has highlighted the weaknesses of the methodology. The main identified weaknesses are: the effort needed to include new architectural decision or development framework to the process, the effort needed to keep all the elements of the process synchronized when one of this elements is included and the need to constantly update the process to keep the pace of technological evolution. Solutions to these weaknesses should be incorporated in future works, making the process more robust.

Table 5.2: Summary of the results for the feasibility validation goal.

Project / Question	BeeFun	NimBees
Q1.1. Is it possible to capture the architectural decisions involved in the design of a multi-layer framework based application in the architectural decisions repository?	100%	100%
Q1.2. Is it possible to generate a set of the architectural decisions taken during the development of a multi-layer application?	100%	100%
Q1.3. Does the management of architectural decisions provided by ArchLayer help the knowledge transfer in case of staff rotation?	NA	Yes
Q1.4. Does the management of architectural decisions provided by ArchLayer entail improvements over existing techniques?	Yes	Yes
Q1.5. Can the knowledge needed to start using a framework in a multi-layer application be modeled?	100%	100%
Q1.6. Can the knowledge needed to implement a given concept using the different mechanism provided by a development framework be modeled?	100%	100%
Q1.7. Does the management of framework information provided by ArchLayer help the knowledge transfer in case of staff rotation?	NA	Yes
Q1.8. Does the management of framework information provided by ArchLayer entail improvements over existing techniques?	Yes	Yes
Q1.9. Is it possible to suggest a feasible set of architectural decisions from the annotated initial design?	100%	100%
Q1.10. Can a specific design be generated from the annotated design and the set of architectural decisions to be used?	Yes	Yes
Q1.11. Does the model transformations provided by ArchLayer help the knowledge transfer in case of staff rotation?	NA	Yes
Q1.12. Does the model transformations provided by ArchLayer entail improvements over existing techniques?	Yes	Yes
Q1.13. Is it possible to automatically generate part of the source code of multi-layer applications from the specific design and the set of architectural decisions?	Yes	Yes
Q1.14. Do the users state that the process and its associated tools and techniques are usable?	6	7

Table 5.3: Summary of the results for the completeness validation goal.

Project / Question	BeeFun	NimBees
Q2.1. Are all the architectural decisions involved in the development of a multi-layer application available for the architect?	85%	100%
Q2.2. Are all the architectural decisions involved in the development of a multi-layer application correctly suggested?	66%	85%
Q2.3. Is all the technical knowledge needed for the design and development of a framework based application stored in the framework information models?	100%	100%
Q2.4. Is all the technical knowledge needed for the design and development of a framework based application automatically transferred to the design by the transformations?	100%	100%
Q2.5. What amount of the application source code can be automatically generated?	30%	30%
Q2.6. Is any additional information necessary to design and develop a multi-layer framework based application?	Yes	Yes

Table 5.4: Summary of the results for the effort validation goal.

Project / Question	BeeFun	NimBees
Q3.1. How much effort is needed for including new architectural decisions not available in the process?	60 hours 1%	0 hours 0%
Q3.2. How much effort is needed for modeling the framework technical knowledge of a not available framework?	250 hours 4.5%	0 hours 0%
Q3.3. How much effort is needed for adapting the process model transformations to new architectural decisions or frameworks added to the process?	20 hours 0.35%	0 hours 0%
Q3.4. How much additional effort is needed for the overhead caused by using ArchLayer?	120 hours 2.1%	80 hours 2.1%
Q3.5. How much effort is saved in the design and development of framework based multi-layer application by using ArchLayer?	-475 hours -8.5%	-316 hours -8.5%
Q3.6. What is the return on investment (ROI) of applying ArchLayer?	-25 hours -0.45%	-236 hours -6.4%

Chapter 6

Conclusion

Humanity does not ask us to be happy. It merely asks us to be brilliant on its behalf.

Ender's Game, Orson Scott Card.

The main conclusions and reflections drawn after development of this thesis are presented in this chapter. It summarizes the main contributions presented, how they improve different aspects of the design and development of framework based multi-layer applications and how they impact to other areas and companies. Finally, it also details how the development of this thesis has contributed to the professional development of its author.

This chapter is organized as follows. Section 6.1 sums up the contributions that have been made in this thesis, how they improve different areas of the development of multi-layer applications and the conclusions drawn. Section 6.2 details the papers that have been written in the scope of this research. Section 6.3 introduces some near and future work. Finally, Section 6.4 presents the author's final reflections about the whole process behind the writing of this thesis.

6.1 Conclusions

This thesis has addressed some of the problems that occur in distributed development centers with high staff rotation. Specifically, the problems faced during the development of framework based multi-layer applications in such centers, because such developments are the most widespread.

Different approaches exist that face this kind of developments. However, they have some limitations. Some of the existing approaches restrict architectural variability by setting a default software architecture that can not be modified or adapted to the specific needs of each project. Others do not take into account the fast pace of evolution of development frameworks, making unfeasible to keep them updated, or are not prepared to be used in distributed development center where the experience of the staff is limited due to a high staff rotation. To the best of the author knowledge, there are not comprehensive proposals managing the development of this kind of applications in the context this thesis is focused.

In this thesis, an architectural decisions repository was defined. The repository contains an organized set of common architectural and technological decisions for the development of multi-layer applications and it is used to help the software architect define the architecture best suited for each project.

To help the architect perform this task, a set of model transformations were defined in this thesis. These transformations take the initial design of the application and automatically suggest a set of architectural decisions, from the decision repository, appropriate to meet the functional and non functional requirements of the application being developed.

Furthermore, a mechanism to store the architectural decisions used for the development of an application was presented. The stored decisions can be used to maintain traceability between the software architecture designed and the final development of the project. They also can be used as the basis for decision-making in future projects and to improve ArchLayer.

Another set of model transformations were defined to help the architect obtain a specific design of the application being developed, tailored to the specific software architecture. Additional technical information about the development frameworks is needed to perform these transformations. This information is provided in framework information models following the meta-model defined in this thesis.

Finally, a tool for automatic code generation was developed. This tools generate a significant amount of the code of multi-layer application using the specific design tailored to the architecture and the technical information about the development frameworks used. This tool, was developed as a major element of the standard used by the regional government of Extremadura for software development.

These contributions, and the techniques and tools supporting them, have been validated in two industrial projects. In every project their feasibility, their completeness and the effort needed to use them were analyzed. The overall results were satisfactory and a ROI was obtained in both projects, even when the process needed to be adapted to new architectural elements. The problems and shortcomings identified during this validation were addressed or are going to be address in the future to improve ArchLayer.

In this regard, ArchLayer is already benefiting companies in the context in which it was developed. First, Gloin is using all the results of this thesis and obtaining a substantial return, as has been detailed above. But also, other companies and organizations are benefiting from this work. The regional government of Extremadura based their standard for the development of Java applications in part of technical work behind this thesis. Thus, not only the government benefits from this work, but any company can easily use the contributions of this thesis in government contracted developments. Moreover, part of the work of this thesis was developed in collaboration with Indra. The standard software architecture of the company was used as the basis for much of the work done in this thesis, so the integration of the process in the company developments can be done in a very direct way.

From these facts, along with the good results obtained in the validation of the process, it can be inferred that this works has fulfilled the main goal set by the author of this thesis and his advisor at its very begging, contribute to research in an area that can be easily applied by companies and where they can exploit the results obtained.

6.2 Publications

All the contributions of this thesis, the works resulting from the application of the knowledge acquired in other areas, and the collaborations with other companies and researchers have been published in prestigious scientific forums. All the written papers are detailed in the following sections: the first one details those that were accepted and the second one specifies the papers that are in review process when this thesis was deposited. This information complements to the one detailed in Table 1.1, which specifies a summary of the published papers.

6.2.1 Published Papers

Below the published papers, sorted chronologically, directly related with this thesis are detailed:

- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "Zentipede: Una contribución a la renovación de la gestión del proceso software" 13th Conference on Software Engineering and Databases. Gijón, Spain, 2008. Pp 391-396.
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "Making software process management agile" Revista Española de Innovación, Calidad e Ingeniería del Software vol. 4 2008. Pp 122-133.
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "Documentation Center: Simplifying the documentation of software project" Third Workshop on Wikis for software engineering. Oporto, Portugal 2008.
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "ParallelJ: Entorno de desarrollo y simulación de programas paralelos" XIV Jornadas de Enseñanza Universitaria de la Informática. Granada, Spain, 2008. Pp 571-578.
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "Decisiones arquitectónicas y tecnológicas como líneas de producto en el desarrollo dirigido por modelos". VII Taller sobre Desarrollo de Software Dirigido por Modelos. Valencia, España, 2010.

- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. “Java para Aplicaciones Corporativas de la Administración”. 15th Conference on Software Engineering and Databases 2010. Valencia, España, pp:263-266.
- **Garcia-Alonso, J.**, Guillén, J., Berrocal, J., Murillo, J.M. “Modelado de la variabilidad en arquitecturas multicapa”. 16th Conference on Software Engineering and Databases 2011. A Coruña, España. 5-7 Septiembre 2011, pp: 895-900. **Awarded as best short paper of the conference track**.
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. “Architectural Variability Management in Multi-Layer Web Applications Through Feature Models”. 4th International Workshop on Feature-Oriented Software Development 2012. Dresden, Germany. 24-25 September 2012, pp: 29-36.
- **Garcia-Alonso, J.**, Berrocal, J., Hernández, F. Murillo, J.M. “GEPRODIST: Gestión de proyectos y desarrollo de software de forma distribuida”. Jornada de Ciencia e Ingeniería de Servicios, 2013.
- **Garcia-Alonso, J.**, Guillen Melo, J., Miranda, J., Berrocal, J., Hernandez, F., and Murillo, J.M..“The PMO Tool: An Information Source for Dashboards”. ISSN 1946-7338. Agile Product and Project Management Executive Update, Vol. 14 (2013), No. 4, pp 1-7.
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. “Architectural Decisions in the Development of Multi-Layer Applications” 8th International Conference on Software Engineering Advances. ISBN: 978-1-61208-304-9. October 27 - November 1, 2013. Venice, Italy (**CORE C**).
- Guillén, J., Miranda, J., Berrocal, J., **García-Alonso, J.**, Murillo, J.M., Canal, C. “People as a Service: a mobile-centric model for providing collective sociological profiles” IEEE Software, 21 Nov. 2013. IEEE computer Society Digital Library. IEEE Computer Society (**JCR, 1,616**).
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. “Technological variability by means of a framework metamodel” 2014 12th International Conference on Software Engineering Research, Management and Applications Studies in Compu-

tational Intelligence, (**CORE C, to be published**).

- **Garcia-Alonso, J.**, Miranda, J., Berrocal, J., Murillo, J.M., Canal, C. "People as a Service: a mobile-centric model for providing collective sociological profiles". 19th Conference on Software Engineering and Databases 2014, (**to be published**).

6.2.2 Pending Papers

This section contains the papers that were going through a review process when this thesis was deposited.

- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "Architectural decisions in the development of framework-based multi-layer applications: A Project Management Office approach" 2014 23nd International Conference on Information Systems Development (ISD2014) Varazdin, Croatia, September 2-4. (**CORE A**).
- **Garcia-Alonso, J.**, Berrocal, J., Murillo, J.M. "Model Transformations for the Automatic Suggestion of Architectural Decisions in the Development of Multi-Layer Applications" 9th International Conference on Software Engineering Advances (**CORE C**).

6.3 Future Works

Throughout the development of this thesis, and especially during its application in real industrial projects, certain areas have been identified that could be further developed in order to make the contributions more robust and to provide a greater benefit to development companies. The following are the most important areas identified:

- The validation on industrial projects has shown that using ArchLayer about 30% of the source code of the application can be automatically generated. This is an area where significant improvements can be achieved, as have been show by others proposal that achieve a much higher percentage of automatic code generation but lose architectural and technological flexibility. The integration of ArchLayer with some of these approaches, especially the standard IFML, will help to advance the code generation without losing flexibility.

- Another aspect that can be improved to increase the interest of the industry for ArchLayer is the technique used to relate the architectural decisions with the initial design of the application. Now, this task has to be done manually, which represent an additional effort in the development and leads to the possibility of errors being introduced in the design. This task could be at least partially automated by a new set of model transformations. These transformations will need additional information to be added to the architectural decisions repository, so the right decisions will be related to the design in each case. Also, additional information will be needed in the initial design to get a more precise transformation.
- As has been mentioned above, one of the weaknesses of the proposal is the close relationship between the different elements that compose it and the difficulty of keeping them synchronized when new elements need to be included. This problem can be approached in two ways. On one hand, the elements of the process could be made more independent, making their synchronization simpler. This approach may be difficult, due to the way the process is designed and would involve major changes to it. On the other hand, this close relationship between the different elements of the process may be exploited to simplify the synchronization. Automatic propagation of changes could be performed on the related elements or at least a set wizards could be developed to guide the steps of developers when modifying the process.
- In a different research area, some of the ideas and techniques presented in this thesis are already being successfully applied. Specifically, in the cloud computing area a similar approach is being applied to the development of multi-clouds applications. These are applications which components will be deployed in multiple cloud providers. The variability present in the different cloud providers is being managed, in a similar way as architectural variability

is managed in this thesis, to develop cloud agnostic applications that can be automatically deployed in different clouds.

- Finally, a new paradigm has been proposed for enabling mobile devices as cloud providers, so they can both consume and provide information using the services deployed on them. Based on this paradigm, a new system may be composed, among other elements, of services deployed in different mobile devices. In this context, some of the ideas presented in this thesis are being applied in the development of these new kind of services to manage the architectural and technological variability present.

6.4 Final Reflections

The work leading to the writing of this thesis has taken place during the last seven years. During these years I got not only the deep technical and scientific knowledge needed to produce a thesis, but also I grew as a person and as a professional.

Young researchers in Spain tend to have a rather precarious situation. In my particular case, I have been fortunate enough to have regular funding that allowed me to pursue a quality research work.

Anyway, the lack of stability has led me to perform tasks that have enormously contributed to my personal growth.

During these years I have actively participated in the application and development of research projects of different scope, including regional, national, European and private research projects. From these experiences I got the skills needed to obtain funds from both public and private research. I also learned how to work in collaboration with other research entities and it gave me the opportunity to work in close collaboration with some of the most important development companies in the country and to learn the intricacies of transferring research results to the industry.

In this context, I was a founding partner of Gloin alongside my supervisor and a colleague sharing my same situation. The knowledge and skills gained along the way of creating and exploiting the company are endless. Some of the most important lessons I learned during this journey are how a software development company works, including all the details from the most mundane to the most glamorous, how to manage a group of intelligent and creative people trying to get the best of them or how to negotiate with private and business customers.

Furthermore, as a Gloin founder, I participated in a start-up acceleration program called Launchpad Denmark. A program of the Danish Ministry of Business and Growth to attract world class entrepreneurs to grow their business in Denmark. Participating in this program provided me the training and the mindset to work in and run a start-up company, which requires a very different skill set than a traditional company.

Finally, in these years I have had the opportunity to teach at the University of Extremadura and at training courses for very important software development companies. The variety of subjects that I had to teach and the diversity of students have allowed me to gain valuable experience both in teaching and in dealing with all types of people.

All this variety of situations that I have faced in recent years have allowed me to be better prepared for the future and have taught me to face all kinds of unexpected and complicated situations.

Appendix A

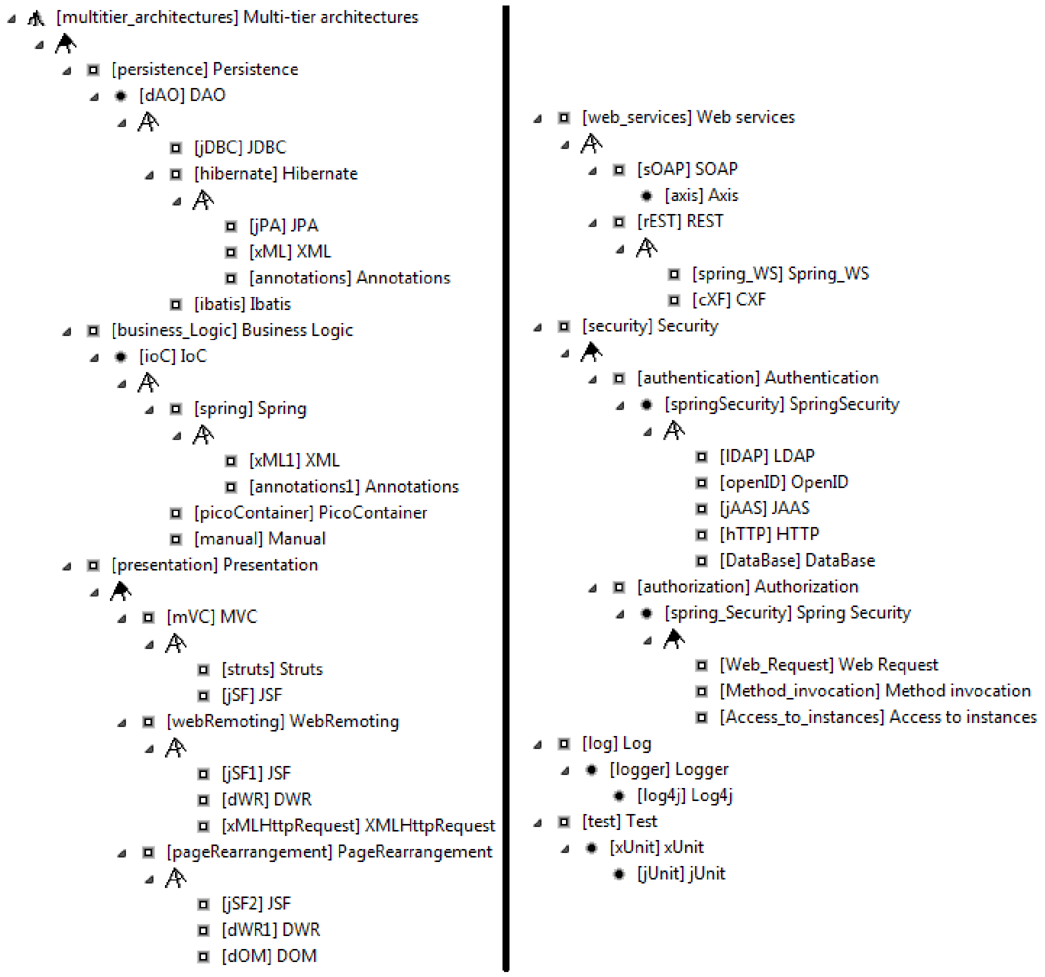
Architectural Decisions Repository

This appendix shows the complete architectural decisions repository. The repository was built as part of the research work done during this thesis and it is based on the architectural decisions and development frameworks detailed in Section 3.3. The complete model is shown in Figure A.1.

Technically, the repository is built as a feature model. To manipulate and edit the feature model the fmp Eclipse plugin developed by the Generative Software Development Lab [1] was used, specifically the version 0.6.6.

A digital version of the architectural decisions repository presented here would be available for download alongside the other additional material of this thesis detailed in Appendix D.

[1]`http://gsd.uwaterloo.ca`



```
[multitier_architectures] Multi-tier architectures
    [persistence] Persistence
      [dAO] DAO
          [jDBC] JDBC
          [hibernate] Hibernate
              [jPA] JPA
              [xML] XML
              [annotations] Annotations
          [ibatis] Ibatis
    [business_Logic] Business Logic
      [ioC] IoC
          [spring] Spring
              [xML1] XML
              [annotations1] Annotations
          [picoContainer] PicoContainer
          [manual] Manual
    [presentation] Presentation
        [mVC] MVC
            [struts] Struts
            [jSF] JSF
        [webRemoting] WebRemoting
            [jSF1] JSF
            [dWR] DWR
            [xMLHttpRequest] XMLHttpRequest
        [pageRearrangement] PageRearrangement
            [jSF2] JSF
            [dWR1] DWR
            [dOM] DOM
    [web_services] Web services
        [sOAP] SOAP
          [axis] Axis
        [rEST] REST
            [spring_WS] Spring_WS
            [cXF] CXF
    [security] Security
        [authentication] Authentication
          [springSecurity] SpringSecurity
              [lDAP] LDAP
              [openID] OpenID
              [jAAS] JAAS
              [hTTP] HTTP
              [DataBase] DataBase
        [authorization] Authorization
          [spring_Security] Spring Security
              [Web_Request] Web Request
              [Method_invocation] Method invocation
              [Access_to_instances] Access to instances
    [log] Log
      [logger] Logger
        [log4j] Log4j
    [test] Test
      [xUnit] xUnit
        [jUnit] jUnit
```

Figure A.1: Feature model containing the architectural decisions repository.

Appendix B

Framework information model

This appendix shows an example of a complete framework information model. For readability reasons, the code generation templates that should be included in the *content* element have been omitted . This model was developed as part of the research work done during this thesis and it is based on the framework information detailed in section 3.4. The complete code of the model is shown in Listing **??**

The model detailed here is based in the framework information meta-model presented in this thesis. A digital version of this model, including the code generation templates, and the framework information meta-model would be available for download alongside the other additional material of this thesis detailed in Appendix D.

```
<?xml version="1.0" encoding="ASCII"?>
<Frameworks:Framework
     xmi:version="2.0"
     xmlns:xmi="http://www.omg.org/XMI"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:Frameworks="Frameworks"
     xsi:schemaLocation="Frameworks␣../metamodel/
         Frameworks.ecore"
     name="Hibernate"
     version="3.4.0.GA">
␣␣<concepts
        xsi:type="Frameworks:InitialConfiguration"
        name="Hibernate␣Initial␣Configuration">
␣␣␣␣<contents
          xsi:type="Frameworks:TextContent"
          fileName="hibernate.properties"
          filePath="/src/main/resources/"
          content=""
          label=""
          newFile="true"/>
␣␣␣␣<contents
          xsi:type="Frameworks:XmlContent"
          fileName="hibernateContext.xml"
          filePath="/src/main/resources/"
          content=""
          label=""
          newFile="true"/>
␣␣␣␣<contents
          xsi:type="Frameworks:XmlContent"
          fileName="daoContext.xml"
          filePath="/src/main/resources/"
          content=""
          label=""
          newFile="true"/>
␣␣␣␣<contents
          xsi:type="Frameworks:SpringXMLContent"
          fileName="applicationContext.xml"
          filePath="/src/main/resources/"
          content=""/>
␣␣␣␣<contents
          xsi:type="Frameworks:MavenPlugins"
          fileName="pom.xml"
          filePath="/"
          content=""/>
␣␣␣␣<contents
          xsi:type="Frameworks:MavenDependencies"
          fileName="pom.xml"
          filePath="/"
          content=""/>
␣␣</concepts>
␣␣<concepts
        xsi:type="Frameworks:FrameworkCompletionCode"
        name="DataSource">
␣␣␣␣<contents
          xsi:type="Frameworks:XmlContent"
          fileName="hibernateContext.xml"
          filePath="/src/main/resources/"
          content=""/>
␣␣␣␣<contents
          xsi:type="Frameworks:XmlContent"
          fileName="hibernateContext.xml"
          filePath="/src/main/resources/"
          content=""/>
```

```
␣␣␣␣<contents
        xsi:type=”Frameworks:XmlContent”
        fileName=”${objeto.dataSourceName}−ds.xml”
        filePath=”/src/main/config/server/”
        content=””
        label=””
        newFile=”true”/>
␣␣</concepts>
␣␣<concepts
      xsi:type=”Frameworks:FrameworkCompletionCode”
      name=”DAO”>
␣␣␣␣<contents
        xsi:type=”Frameworks:TextContent”
        fileName=”${objeto.nombreClaseMayuscula}.java”
        filePath=”/src/main/java/$objeto.paquete/”
        content=””/>
␣␣␣␣<contents
        xsi:type=”Frameworks:TextContent”
        fileName=”${objeto2.tipo}.java”
        filePath=”/src/main/java/$objeto2.paquete/”
        content=””
        position=”BEFORE_LAST”
        label=”}”/>
␣␣␣␣<contents
        xsi:type=”Frameworks:TextContent”
        fileName=”${objeto2.tipo}.java”
        filePath=”/src/main/java/$objeto2.paquete/”
        content=””
        position=”AFTER_FIRST”
        label=”;”/>
␣␣␣␣<contents
        xsi:type=”Frameworks:XmlContent”
        fileName=”daoContext.xml”
        filePath=”/src/main/resources/”
        content=””
        position=”BEFORE_LAST”
        label=”&lt;/beans>”/>
␣␣␣␣<contents
        xsi:type=”Frameworks:XmlContent”
        fileName=”hibernateContext.xml”
        filePath=”/src/main/resources/”
        content=””
        position=”BEFORE_FIRST”
        label=”&lt;/list>”/>
␣␣␣␣<contents
        xsi:type=”Frameworks:XmlContent”
        fileName=”applicationContext.xml”
        filePath="/src/main/resources/"
        content=""
        label="&lt;context:component-scan base-package=&quot;"/>
    <contents
        xsi:type="Frameworks:TextContent"
        fileName="/${objeto.nombreClaseMayuscula}DAO.java"
        filePath="/src/main/java/$objeto.paquete/dao/interfaz/"
        content=""
        label=""
        newFile="true"/>
    <contents
        xsi:type="Frameworks:TextContent"
        fileName="${objeto.nombreClaseMayuscula}DAOImpl.java"
        filePath="/src/main/java/$objeto.paquete/dao/impl/"
        content=""
        label=""/>
    <contents
        xsi:type="Frameworks:TextContent"
        fileName="${objeto.nombreClaseMayuscula}Test.java"
        filePath="/src/test/java/${objeto.paquete}/"
        content=""
        label=""
        newFile="true"/>
  </concepts>
</Frameworks:Framework>
```

Listing B.1: Hibernate framework information model

Appendix C

Model transformations

This appendix shows the ATL model transformations that support the process presented in this thesis and that have been described in Section 3.5. A digital version of the model transformations presented here would be available for download alongside the other additional material of this thesis detailed in Appendix D.

C.1 Layer suggestion transformation

The model transformation that suggest a set of layers based on the initial design of an applications is showed in Listing C.1

```
-- @path UCP=/Transformations/Models/UCProfile.profile.
   uml
-- @nsURI FMP=http:///fmp.ecore
-- @nsURI UML=http://www.eclipse.org/uml2/4.0.0/UML

module LayerSuggestionTransformation;

create FeaturesOUT : FMP from FeaturesIn : FMP,
    UMLDiagramIn : UML, UCProfileIn : UCP, ADProfileIn :
    AP;

rule projectCopy{
     from f : FMP!Project
     to t : FMP!Project (model<-f.model, metaModel<-f.
         metaModel, metaMetaModel<-f.metaMetaModel)
}
```

```

rule featureCopy{
    from f : FMP!Feature
    to t : FMP!Feature
        (max<-f.max,
            min<-f.min,
            id<-f.id,
            children<-f.children,
            confs<-f.confs,
            origin<-f.origin,
            state<-f.state,
            clones<-f.clones,
            prototype<-f.prototype,
            name<-f.name,
            valueType<-f.valueType,
            describedNode<-f.describedNode,
            properties<-f.properties,
            typedValue<-f.typedValue,
            constraints<-f.constraints,
            configurations<-f.configurations,
            references<-f.references)
    do{
        if(f.isLayerConfiguration()){
            if(f.isManagementConfiguration()){
                t.state<-#USER_SELECTED;
            }
            if(f.isInversionOfControlConfiguration()){
                t.state<-#USER_SELECTED;
            }
            if(f.isPersistenceConfiguration()){
                if(UML!DataStoreNode.allInstances().size
                    ()>0){
                    t.state<-#USER_SELECTED;
                }
            }
            if(f.isPresentationConfiguration()){
                for (actor in UML!Actor.allInstances()){
                    if (not actor.name.startsWith('LS:'))
                        {
                        t.state<-#USER_SELECTED;
                    }
                }
            }
            if(f.isReportConfiguration()){
                --Never automatically suggested
            }
            if(f.isWebServicesConfiguration()){
                for (actor in UML!Actor.allInstances()){
                    if (actor.name.startsWith('LS:')){
                        t.state<-#USER_SELECTED;
                    }
                }
            }
            if(f.isTestConfiguration()){
                for (stereotype in UCP!Stereotype.
                    allInstances()){
                    if(stereotype.name='Testability' or
                        stereotype.name='Correctness'){
                        for (useCase in UML!UseCase.
                            allInstances()){
                            if(useCase.isAnnotated(
                                stereotype)){
                                t.state<-#USER_SELECTED;
                            }
                        }
                    }
                }
                for (stereotype in AP!Stereotype.
                    allInstances()){
                    if(stereotype.name='Testability' or
                        stereotype.name='Correctness'){
                        for (activityPartition in UML!
                            ActivityPartition.allInstances
                            ()){
                            if(activityPartition.
                                isAnnotated(stereotype)){
                                t.state<-#USER_SELECTED;
                            }
                        }
                    }
                }

            }
            if(f.isLogConfiguration()){
                for (stereotype in UCP!Stereotype.
                    allInstances()){
                    if(stereotype.name='Maintainability'
                        or stereotype.name='Accountability
                        ' or stereotype.name='
                        Analysability'){
                        for (useCase in UML!UseCase.
                            allInstances()){
                            if(useCase.isAnnotated(
                                stereotype)){
```

```
                    t.state<-#USER_SELECTED;
                }
            }
        }
    }
    for (stereotype in AP!Stereotype.
       allInstances()){
        if(stereotype.name='Maintainability'
           or stereotype.name='Accountability
           ' or stereotype.name='
           Analysability'){
            for (activityPartition in UML!
               ActivityPartition.allInstances
               ()){
                if(activityPartition.
                   isAnnotated(stereotype)){
                    t.state<-#USER_SELECTED;
                }
            }
        }
    }
}
if(f.isSecurityConfiguration()){
    for (stereotype in UCP!Stereotype.
       allInstances()){
        if(stereotype.name='Authenticity' or
           stereotype.name='Security'  or
           stereotype.name='Confidentiality')
           {
            for (useCase in UML!UseCase.
               allInstances()){
                if(useCase.isAnnotated(
                   stereotype)){
                    t.state<-#USER_SELECTED;
                }
            }
            -- Alternative - Security layer
               is suggested if half or more
               or the use cases are annotated
                with one of the stereotypes
            --if(thisModule.halfUCAnnotated(
               stereotype)){
                --t.state<-#USER_SELECTED;
            --}
        }
    }
```

```
                        for (stereotype in AP!Stereotype.
                            allInstances()){
                            if(stereotype.name='Authenticity' or
                                stereotype.name='Security' or
                                stereotype.name='Confidentiality')
                                {
                                for (activityPartition in UML!
                                    ActivityPartition.allInstances
                                    ()){
                                    if(activityPartition.
                                        isAnnotated(stereotype)){
                                        t.state<-#USER_SELECTED;
                                    }
                                }
                            }
                        }
                    }
                }
            }
}

rule typedValueCopy{
    from f: FMP!TypedValue
    to t: FMP!TypedValue
        (integerValue<-f.integerValue,
            stringValue<-f.stringValue,
            floatValue<-f.floatValue,
            featureValue<-f.featureValue)
}

rule constraintCopy{
    from f: FMP!Constraint
    to t: FMP!Constraint (text<-f.text)
}

rule referenceCopy{
    from f : FMP!Reference
    to t : FMP!Reference
        (max<-f.max,
            min<-f.min,
            id<-f.id,
            children<-f.children,
            confs<-f.confs,
            origin<-f.origin,
            state<-f.state,
            clones<-f.clones,
            prototype<-f.prototype,
                feature<-f.feature)
}

rule featureGroupCopy{
     from f : FMP!FeatureGroup
     to t : FMP!FeatureGroup
          (max<-f.max,
               min<-f.min,
               id<-f.id,
               children<-f.children,
               confs<-f.confs,
               origin<-f.origin)
}

helper context FMP!Feature def: isLayerConfiguration():
    Boolean =
     if (self.isManagementConfiguration() or
               self.isInversionOfControlConfiguration() or
               self.isPersistenceConfiguration() or
               self.isPresentationConfiguration() or
               self.isReportConfiguration() or
               self.isWebServicesConfiguration() or
               self.isTestConfiguration() or
               self.isLogConfiguration() or
               self.isSecurityConfiguration()) then
          true
     else
          false
     endif;

helper context FMP!Feature def: isManagementConfiguration
    (): Boolean =
     if (self.name='Management' and not self.origin.
         oclIsUndefined()) then
          true
     else
          false
     endif;

helper context FMP!Feature def:
    isInversionOfControlConfiguration(): Boolean =
     if (self.name='InversionOfControl' and not self.
         origin.oclIsUndefined()) then
          true
     else
          false
     endif;
```

```

helper context FMP!Feature def:
    isPersistenceConfiguration(): Boolean =
     if (self.name='Persistence' and not self.origin.
        oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def:
    isPresentationConfiguration(): Boolean =
     if (self.name='Presentation' and not self.origin.
        oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def: isReportConfiguration():
    Boolean =
     if (self.name='Report' and not self.origin.
        oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def:
    isWebServicesConfiguration(): Boolean =
     if (self.name='WebServices' and not self.origin.
        oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def: isTestConfiguration():
    Boolean =
     if (self.name='Test' and not self.origin.
        oclIsUndefined()) then
         true
     else
         false
     endif;

```

```
helper context FMP!Feature def: isLogConfiguration():
    Boolean =
     if (self.name='Log' and not self.origin.
         oclIsUndefined()) then
          true
     else
          false
     endif;

helper context FMP!Feature def: isSecurityConfiguration()
    : Boolean =
     if (self.name='Security' and not self.origin.
         oclIsUndefined()) then
          true
     else
          false
     endif;

helper context UML!UseCase def : isAnnotated(s : UCP!
    Stereotype) : Boolean =
     if self.extensionPoint->select(exten | not exten.
         getAppliedStereotype(s.qualifiedName).
         oclIsUndefined()).size()>0 then
          true
     else
          false
     endif
;
helper context UML!ActivityPartition def : isAnnotated(s
    : AP!Stereotype) : Boolean =
     if not self.getAppliedStereotype(s.qualifiedName).
         oclIsUndefined() then
          true
     else
          false
     endif
;

helper def : halfUCAnnotated(s : UCP!Stereotype) :
    Boolean =
     if (UML!UseCase.allInstances()->collect(ext | ext.
         extensionPoint).flatten()->select(exten | not
         exten.getAppliedStereotype(s.qualifiedName).
         oclIsUndefined()).size())/ UML!UseCase.
         allInstances().size() >= 0.5    then
          true
     else
```

```
            false
        endif
;
```

Listing C.1: Layer suggestion transformation

C.2 Layered design transformation

An excerpt of the model transformation that generates a design adapted to the layers selected by the architect is showed in Listing C.2

```
-- @path AP=/Models/ADProfile.profile.uml
-- @nsURI UML=http://www.eclipse.org/uml2/4.0.0/UML
-- @nsURI FMP=http:///fmp.ecore

module LayeredActivityDiagramTransformation;
create ActivityOUT : UML from FeatureIn : FMP, ActivityIn
    : UML, ActivityProfileIn : AP;

helper def : layers : Sequence(UML!ActivityPartition)=
    Sequence{};

rule selectedLayer2ActivityPartition{
    from f : FMP!Feature (f.isSelectedLayerConfiguration
        ())
    to t : UML!ActivityPartition (name<-f.name)
    do{
        thisModule.layers<-thisModule.layers.append(t);
    }
}

rule Model {
    from s : UML!Model
    to t : UML!Model (
        name <- s.name,
        packagedElement <- s.packagedElement)
}

rule OpaqueAction {
    from s : UML!OpaqueAction
    to t : UML!OpaqueAction (
        name <- s.name,
        isLeaf <- s.isLeaf,
        outgoing <- s.outgoing,
        incoming <- s.incoming,
        inPartition <- s.inPartition)
    do{
        if(not thisModule.getLayer('InversionOfControl').
            oclIsUndefined()){
             t.inPartition<-t.inPartition.append(
                 thisModule.getLayer('InversionOfControl'))
                 ;
        }
        if(not thisModule.getLayer('Persistence').
            oclIsUndefined()){
             if(s.outgoing->select(flow|flow.target.
                 oclIsTypeOf(UML!DataStoreNode)).size() > 0
                       or s.incoming->select(flow|flow.
                           source.oclIsTypeOf(UML!
                           DataStoreNode)).size() > 0){
                  t.inPartition<-t.inPartition.append(
                      thisModule.getLayer('Persistence'));
             }
        }
        if(not thisModule.getLayer('Presentation').
            oclIsUndefined()){

        }
        if(not thisModule.getLayer('Report').
            oclIsUndefined()){

        }
        if(not thisModule.getLayer('WebServices').
            oclIsUndefined()){

        }
        if(not thisModule.getLayer('Test').oclIsUndefined
            ()){
             t.inPartition<-t.inPartition.append(
                 thisModule.getLayer('Test'));
        }
        if(not thisModule.getLayer('Log').oclIsUndefined
            ()){
             t.inPartition<-t.inPartition.append(
                 thisModule.getLayer('Log'));
        }
        if(not thisModule.getLayer('Security').
            oclIsUndefined()){

        }
    }
}
```

```

rule ControlFlow {
    from s : UML!ControlFlow
    to t : UML!ControlFlow (
        name <- s.name,
        isLeaf <- s.isLeaf,
        source <- s.source,
        target <- s.target)
}

rule DataStoreNode {
    from s : UML!DataStoreNode
    to t : UML!DataStoreNode (
        name <- s.name,
        isLeaf <- s.isLeaf,
        isControlType <- s.isControlType,
        outgoing <- s.outgoing,
        incoming <- s.incoming)
}

rule ForkNode {
    from s : UML!ForkNode
    to t : UML!ForkNode(
        name <- s.name,
        isLeaf <- s.isLeaf,
        outgoing <- s.outgoing,
        incoming <- s.incoming)
}

rule JoinNode {
    from s : UML!JoinNode
    to t : UML!JoinNode (
        name <- s.name,
        isLeaf <- s.isLeaf,
        outgoing <- s.outgoing,
        incoming <- s.incoming)
}

rule ActivityFinalNode {
    from s : UML!ActivityFinalNode
    to t : UML!ActivityFinalNode(
        name <- s.name,
        isLeaf <- s.isLeaf,
        outgoing <- s.outgoing,
        incoming <- s.incoming)
}

```

```
rule InitialNode {
    from s : UML!InitialNode
    to t : UML!InitialNode (
        name <- s.name,
        isLeaf <- s.isLeaf,
        outgoing <- s.outgoing,
        incoming <- s.incoming,
        inPartition <- s.inPartition)
    do{
        if(not thisModule.getLayer('InversionOfControl').
            oclIsUndefined()){
            t.inPartition<-t.inPartition.append(
                thisModule.getLayer('InversionOfControl'))
                ;
        }
        if(not thisModule.getLayer('Persistence').
            oclIsUndefined()){
            if(t.outgoing->select(flow|flow.target.
                oclIsTypeOf(UML!DataStoreNode)).size() > 0
                    or t.incoming->select(flow|flow.
                        source.oclIsTypeOf(UML!
                        DataStoreNode)).size() > 0){
                t.inPartition<-t.inPartition.append(
                    thisModule.getLayer('Persistence'));
            }
        }
        if(not thisModule.getLayer('Presentation').
            oclIsUndefined()){
            t.inPartition<-t.inPartition.append(
                thisModule.getLayer('Presentation'));
        }
        if(not thisModule.getLayer('Report').
            oclIsUndefined()){

        }
        if(not thisModule.getLayer('WebServices').
            oclIsUndefined()){

        }
        if(not thisModule.getLayer('Test').oclIsUndefined
            ()){
            t.inPartition<-t.inPartition.append(
                thisModule.getLayer('Test'));
        }
        if(not thisModule.getLayer('Log').oclIsUndefined
            ()){
                t.inPartition<-t.inPartition.append(
                    thisModule.getLayer('Log'));
            }
            if(not thisModule.getLayer('Security').
                oclIsUndefined()){
                t.inPartition<-t.inPartition.append(
                    thisModule.getLayer('Security'));
            }
        }
}

rule CallBehaviorAction {
    from s : UML!CallBehaviorAction
    to t : UML!CallBehaviorAction(
        name <- s.name,
        isLeaf <- s.isLeaf,
        outgoing <- s.outgoing,
        incoming <- s.incoming)
}

rule Interaction {
    from s : UML!Interaction
    to t : UML!Interaction(
        name <- s.name,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        isActive <- s.isActive,
        fragment <- s.fragment)
}

rule InteractionUse {
    from s : UML!InteractionUse (s.oclIsTypeOf(UML!
        InteractionUse))
    to t : UML!InteractionUse (
        name <- s.name,
        refersTo <- s.refersTo)
}

rule ActivityPartition {
    from s : UML!ActivityPartition
    to t : UML!ActivityPartition(
        name <- s.name,
        isDimension <- s.isDimension,
        isExternal <- s.isExternal,
        node <- s.node,
        subpartition <- s.subpartition)
}
```

```

rule Activity {
    from s : UML!Activity
    to t : UML!Activity (
        name <- s.name,
        isLeaf <- s.isLeaf,
        isAbstract <- s.isAbstract,
        isActive <- s.isActive,
        isReadOnly <- s.isReadOnly,
        isSingleExecution <- s.isSingleExecution,
        ownedBehavior <- s.ownedBehavior,
        node <- s.node,
        edge <- s.edge,
        group <- s.group)
}

helper def: getLayer(layerName: String): UML!
    ActivityPartition =
    thisModule.layers->select(layer|layer.name=layerName)
        ->first();

helper context FMP!Feature def:
    isSelectedLayerConfiguration(): Boolean =
    if (self.isManagementConfiguration() or
            self.isInversionOfControlConfiguration() or
            self.isPersistenceConfiguration() or
            self.isPresentationConfiguration() or
            self.isReportConfiguration() or
            self.isWebServicesConfiguration() or
            self.isTestConfiguration() or
            self.isLogConfiguration() or
            self.isSecurityConfiguration()) then
        true
    else
        false
    endif;

helper context FMP!Feature def: isManagementConfiguration
    (): Boolean =
    if (self.name='Management' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
        true
    else
        false
    endif;

```

```
helper context FMP!Feature def:
    isInversionOfControlConfiguration(): Boolean =
     if (self.name='InversionOfControl' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
         true
     else
         false
     endif;

helper context FMP!Feature def:
    isPersistenceConfiguration(): Boolean =
     if (self.name='Persistence' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
         true
     else
         false
     endif;

helper context FMP!Feature def:
    isPresentationConfiguration(): Boolean =
     if (self.name='Presentation' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
         true
     else
         false
     endif;

helper context FMP!Feature def: isReportConfiguration():
    Boolean =
     if (self.name='Report' and self.state=#USER_SELECTED
        and not self.origin.oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def:
    isWebServicesConfiguration(): Boolean =
     if (self.name='WebServices' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
         true
     else
         false
     endif;

helper context FMP!Feature def: isTestConfiguration():
    Boolean =
     if (self.name='Test' and self.state=#USER_SELECTED
        and not self.origin.oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def: isLogConfiguration():
    Boolean =
     if (self.name='Log' and self.state=#USER_SELECTED and
         not self.origin.oclIsUndefined()) then
         true
     else
         false
     endif;

helper context FMP!Feature def: isSecurityConfiguration()
    : Boolean =
     if (self.name='Security' and self.state=#USER_SELECTE
        D and not self.origin.oclIsUndefined()) then
         true
     else
         false
     endif;
```

Listing C.2: Excerpt of the layered design transformation

C.3 Design patterns and framework suggestion transformations

An excerpt of the model transformation that suggest a set of design pattern based on the initial design of an applications and the set of layers selected by the architect is showed in Listing C.3

```
-- @path UCP=/Transformations/Models/UCProfile.profile.
    uml
-- @nsURI UML=http://www.eclipse.org/uml2/4.0.0/UML
-- @nsURI FMP=http:///fmp.ecore

module DesignPatternsSuggestionTransformation;
```

```
create FeaturesOut : FMP from FeaturesIn : FMP,
    UMLDiagramIn : UML, UCProfileIn : UCP;

helper def : designPatterns : Sequence(Sequence(FMP!
    Feature))=OclUndefined;
helper def : selectedDesignPatterns : Sequence(FMP!
    Feature)=Sequence{};

rule projectCopy{
     from f : FMP!Project
     to t : FMP!Project (model<-f.model, metaModel<-f.
         metaModel, metaMetaModel<-f.metaMetaModel)
}

rule featureCopy{
     from f : FMP!Feature
     using{
          layerDesignPatterns: Sequence(FMP!Feature)=
              OclUndefined;
          feature: FMP!Feature=OclUndefined;
          selectedDesignPattern: FMP!Feature=OclUndefined;
          QAsMeet: Integer=0;
          selectedQAsMeet: Integer=-1;
          properties: Sequence(MM!Node)=OclUndefined;
          property: MM!Feature=OclUndefined;
          affectedQAs: Sequence(String)=OclUndefined;
          affectedQA: String=OclUndefined;
          relatedDesignPatterns: Sequence(String)=
              OclUndefined;
          relatedDesignPattern: String=OclUndefined;
          relatedDesignPatternAffectedQAsInfo: Sequence(
              String)=OclUndefined;
          relatedDesignPatternName: String=OclUndefined;
          relatedDesignPatternsAffectedQAs: Sequence(String
              )=OclUndefined;
     }
     to t : FMP!Feature
          (max<-f.max,
               min<-f.min,
               id<-f.id,
               children<-f.children,
               confs<-f.confs,
               origin<-f.origin,
               state<-f.state,
               clones<-f.clones,
               prototype<-f.prototype,
               name<-f.name,
            valueType<-f.valueType,
            describedNode<-f.describedNode,
            properties<-f.properties,
            typedValue<-f.typedValue,
            constraints<-f.constraints,
            configurations<-f.configurations,
            references<-f.references)
    do{
        -- The first time this rule is applied the
           suggested design patterns are calculated.
        if(thisModule.designPatterns.oclIsUndefined()){
            thisModule.designPatterns<-thisModule.
               getDesignPatterns;
            -- Iteration over the selected layers
            for(layerDesignPatterns in thisModule.
               designPatterns){
                selectedQAsMeet<- -1;
                selectedDesignPattern<- OclUndefined;
                -- Iteration over the design patterns of
                   a selected layer
                for(feature in layerDesignPatterns){
                    QAsMeet<-0;
                    --Calculates the priority value of
                       the pattern based on the affected
                       QAs
                    properties<-feature.properties.
                       children;
                    for(property in properties){
                        --Positively affected QAs added
                           to the priority value
                        if(property.name='
                           PositivelyAffectedQAs' and not
                            property.typedValue.
                           oclIsUndefined()){
                            affectedQAs<-property.
                               typedValue.stringValue.
                               split(',');
                            for(affectedQA in affectedQAs
                               ){
                                if(affectedQA.trim() <> '
                                    '){
                                    QAsMeet<- QAsMeet +
                                       thisModule.
                                       annotatedUC(
                                       thisModule.
                                       getStereotype(
                                       affectedQA.trim())
                                    );
                                }
                            }
                        }
                        --Negatively affected QAs
                            subtracted to the priority
                            value
                        if(property.name='
                            NegativelyAffectedQAs' and not
                             property.typedValue.
                            oclIsUndefined()){
                             affectedQAs<-property.
                                typedValue.stringValue.
                                split(',');
                             for(affectedQA in affectedQAs
                                ){
                                 if(affectedQA.trim() <> '
                                    '){
                                     QAsMeet<- QAsMeet -
                                         thisModule.
                                         annotatedUC(
                                         thisModule.
                                         getStereotype(
                                         affectedQA.trim())
                                         );
                                 }
                             }
                        }
                        --Effects of combination with
                            previously selected design
                            patterns added to the priority
                             value
                        if(property.name='
                            FrameworkCombinationAffectedQAs
                            ' and not property.typedValue.
                            oclIsUndefined()){
                             relatedDesignPatterns<-
                                property.typedValue.
                                stringValue.split('#');
                             for(relatedDesignPattern in
                                relatedDesignPatterns){
                                 relatedDesignPatternAffectedQAsInfo
                                    <-relatedDesignPattern
                                    .trim().split(':');
                                 relatedDesignPatternName
                                    <-
                                    relatedDesignPatternAffectedQAsInfo
```

```
                                                  .first();
                                              if(thisModule.
                                                  isDesignPatternSelected
                                                  (
                                                  relatedDesignPatternName
                                                  )){
                                                   relatedDesignPatternsAffectedQAs
                                                       <-
                                                       relatedDesignPatternAffectedQAsI
                                                       .last().split(';')
                                                       ;
                                                   affectedQAs<-
                                                       relatedDesignPatternsAffectedQAs
                                                       .first().split(','
                                                       );
                                                   for(affectedQA in
                                                       affectedQAs){
                                                        if(affectedQA.
                                                            trim() <> ''){
                                                             QAsMeet<-
                                                                 QAsMeet +
                                                                 thisModule
                                                                 .
                                                                 annotatedUC
                                                                 (
                                                                 thisModule
                                                                 .
                                                                 getStereotype
                                                                 (
                                                                 affectedQA
                                                                 .trim()));
                                                        }
                                                   }
                                                   affectedQAs<-
                                                       relatedDesignPatternsAffectedQAs
                                                       .last().split(',')
                                                       ;
                                                   for(affectedQA in
                                                       affectedQAs){
                                                        if(affectedQA.
                                                            trim() <> ''){
                                                             QAsMeet<-
                                                                 QAsMeet -
                                                                 thisModule
                                                                 .
                                                                 annotatedUC
                                                                 (
                                                                 thisModule
                                                                 .
                                                                 getStereotype
                                                                 (
                                                                 affectedQA
                                                                 .trim()));
                                                        }
                                                   }
                                               }
                                           }
                                       }
                                   }
                                   --Select the Design pattern with
                                       higher priority value
                                   if(QAsMeet > selectedQAsMeet){
                                       selectedQAsMeet<-QAsMeet;
                                       selectedDesignPattern<-feature;
                                   }
                               }
                               if(not selectedDesignPattern.
                                   oclIsUndefined()){
                                   thisModule.selectedDesignPatterns<-
                                       thisModule.selectedDesignPatterns.
                                       append(selectedDesignPattern);
                               }
                           }
                       }
                       if(thisModule.selectedDesignPatterns->includes(f)
                           ){
                           t.state<-#USER_SELECTED;
                       }
                   }
}

rule typedValueCopy{
   from f: FMP!TypedValue
   to t: FMP!TypedValue
       (integerValue<-f.integerValue,
           stringValue<-f.stringValue,
           floatValue<-f.floatValue,
           featureValue<-f.featureValue)
}

rule constraintCopy{
   from f: FMP!Constraint
   to t: FMP!Constraint (text<-f.text)
}
```

```

rule referenceCopy{
     from f : FMP!Reference
     to t : FMP!Reference
          (max<-f.max,
               min<-f.min,
               id<-f.id,
               children<-f.children,
               confs<-f.confs,
               origin<-f.origin,
               state<-f.state,
               clones<-f.clones,
               prototype<-f.prototype,
               feature<-f.feature)
}

rule featureGroupCopy{
     from f : FMP!FeatureGroup
     to t : FMP!FeatureGroup
          (max<-f.max,
               min<-f.min,
               id<-f.id,
               children<-f.children,
               confs<-f.confs,
               origin<-f.origin)
}

helper context FMP!Feature def:
    isSelectedLayerConfiguration(): Boolean =
     if (self.isManagementConfiguration() or
               self.isInversionOfControlConfiguration() or
               self.isPersistenceConfiguration() or
               self.isPresentationConfiguration() or
               self.isReportConfiguration() or
               self.isWebServicesConfiguration() or
               self.isTestConfiguration() or
               self.isLogConfiguration() or
               self.isSecurityConfiguration()) then
          true
     else
          false
     endif;

helper context FMP!Feature def: isManagementConfiguration
    (): Boolean =
     if (self.name='Management' and self.state=#
         USER_SELECTED and not self.origin.oclIsUndefined()
```

```
        ) then
          true
      else
          false
      endif;

helper context FMP!Feature def:
    isInversionOfControlConfiguration(): Boolean =
      if (self.name='InversionOfControl' and self.state=#
         USER_SELECTED and not self.origin.oclIsUndefined()
         ) then
          true
      else
          false
      endif;

helper context FMP!Feature def:
    isPersistenceConfiguration(): Boolean =
      if (self.name='Persistence' and self.state=#
         USER_SELECTED and not self.origin.oclIsUndefined()
         ) then
          true
      else
          false
      endif;

helper context FMP!Feature def:
    isPresentationConfiguration(): Boolean =
      if (self.name='Presentation' and self.state=#
         USER_SELECTED and not self.origin.oclIsUndefined()
         ) then
          true
      else
          false
      endif;

helper context FMP!Feature def: isReportConfiguration():
    Boolean =
      if (self.name='Report' and self.state=#USER_SELECTED
         and not self.origin.oclIsUndefined()) then
          true
      else
          false
      endif;

helper context FMP!Feature def:
    isWebServicesConfiguration(): Boolean =
      if (self.name='WebServices' and self.state=#
         USER_SELECTED and not self.origin.oclIsUndefined()
         ) then
          true
      else
          false
      endif;

helper context FMP!Feature def: isTestConfiguration():
    Boolean =
      if (self.name='Test' and self.state=#USER_SELECTED
         and not self.origin.oclIsUndefined()) then
          true
      else
          false
      endif;

helper context FMP!Feature def: isLogConfiguration():
    Boolean =
      if (self.name='Log' and self.state=#USER_SELECTED and
          not self.origin.oclIsUndefined()) then
          true
      else
          false
      endif;

helper context FMP!Feature def: isSecurityConfiguration()
    : Boolean =
      if (self.name='Security' and self.state=#USER_SELECTE
         D and not self.origin.oclIsUndefined()) then
          true
      else
          false
      endif;

-- Return a Sequence containing all the dessign pattern
    available for the implementation of the selected
    layers
helper def : getDesignPatterns : Sequence(Sequence(FMP!
    Feature)) =
    FMP!Feature.allInstances()->select(layer|layer.
          isSelectedLayerConfiguration())->collect(feature|
          feature.children.first().children)
;

-- Return the number of UC annotated with a given
    Stereotype
```

```
helper def : annotatedUC(s : UCP!Stereotype) : Integer =
    UML!UseCase.allInstances()->collect(ext | ext.
        extensionPoint).flatten()->select(exten | not
        exten.getAppliedStereotype(s.qualifiedName).
        oclIsUndefined()).size()
;

-- Return a Stereotype by its name
helper def : getStereotype(stereotypeName : String) : UCP
    !Stereotype =
    UCP!Stereotype.allInstances()->select(stereotype|
        stereotype.name=stereotypeName)->first()
;

-- Return a DesignPattern by its name
helper def : getDesignPatternByName(designPatternName :
    String) : FMP!Feature =
    FMP!Feature.allInstances()->select(designPattern|
        designPattern.name=designPatternName and not
        designPattern.origin.oclIsUndefined())->first()
;

helper def: isDesignPatternSelected(designPattern: String
    ): Boolean =
     if (thisModule.selectedDesignPatterns->includes(
        thisModule.getDesignPatternByName(designPattern)))
          then
          true
     else
          false
     endif;
```

Listing C.3: Excerpt of the design patterns suggestion transformation

An excerpt of the model transformation that suggest a set of frameworks based on the initial design of an applications and the set of design patterns selected by the architect is showed in Listing C.4

```
-- @path UCP=/Transformations/Models/UCProfile.profile.
    uml
-- @nsURI UML=http://www.eclipse.org/uml2/4.0.0/UML
-- @nsURI FMP=http:///fmp.ecore

module FrameworkSuggestionTransformation;
create FeaturesOut : FMP from FeaturesIn : FMP,
    UMLDiagramIn : UML, UCProfileIn : UCP;

```

```
helper def : frameworks : Sequence(Sequence(FMP!Feature))
    =OclUndefined;
helper def : selectedFrameworks : Sequence(FMP!Feature)=
    Sequence{};

rule projectCopy{
     from f : FMP!Project
     to t : FMP!Project (model<-f.model, metaModel<-f.
          metaModel, metaMetaModel<-f.metaMetaModel)
}

rule featureCopy{
     from f : FMP!Feature
     using{
          designPatternFrameworks: Sequence(FMP!Feature)=
               OclUndefined;
          feature: FMP!Feature=OclUndefined;
          selectedFramework: FMP!Feature=OclUndefined;
          QAsMeet: Integer=0;
          selectedQAsMeet: Integer=-1;
          properties: Sequence(MM!Node)=OclUndefined;
          property: MM!Feature=OclUndefined;
          affectedQAs: Sequence(String)=OclUndefined;
          affectedQA: String=OclUndefined;
          relatedFrameworks: Sequence(String)=OclUndefined;
          relatedFramework: String=OclUndefined;
          relatedFrameworkAffectedQAsInfo: Sequence(String)
               =OclUndefined;
          relatedFrameworkName: String=OclUndefined;
          relatedFrameworksAffectedQAs: Sequence(String)=
               OclUndefined;
     }
     to t : FMP!Feature
          (max<-f.max,
               min<-f.min,
               id<-f.id,
               children<-f.children,
               confs<-f.confs,
               origin<-f.origin,
               state<-f.state,
               clones<-f.clones,
               prototype<-f.prototype,
               name<-f.name,
               valueType<-f.valueType,
               describedNode<-f.describedNode,
               properties<-f.properties,
               typedValue<-f.typedValue,
                constraints<-f.constraints,
                configurations<-f.configurations,
                references<-f.references)
        do{
            -- The first time this rule is applied the
                suggested frameworks are calculated.
            if(thisModule.frameworks.oclIsUndefined()){
                thisModule.frameworks<-thisModule.
                    getFrameworks;
                -- Iteration over the selected design
                    patterns
                for(designPatternFrameworks in thisModule.
                    frameworks){
                    selectedQAsMeet<- -1;
                    selectedFramework<- OclUndefined;
                    -- Iteration over the frameworks of a
                        selected design pattern
                    for(feature in designPatternFrameworks){
                        QAsMeet<-0;
                        --Calculates the priority value of
                            the framework based on the
                            affected QAs
                        properties<-feature.properties.
                            children;
                        for(property in properties){
                            --Positively affected QAs added
                                to the priority value
                            if(property.name='
                                PositivelyAffectedQAs' and not
                                 property.typedValue.
                                oclIsUndefined()){
                                 affectedQAs<-property.
                                     typedValue.stringValue.
                                     split(',');
                                 for(affectedQA in affectedQAs
                                     ){
                                      if(affectedQA.trim() <> '
                                          '){
                                           QAsMeet<- QAsMeet +
                                               thisModule.
                                               annotatedUC(
                                               thisModule.
                                               getStereotype(
                                               affectedQA.trim())
                                               );
                                      }
                                 }
```

```
}
--Negatively affected QAs subtracted to the priority value
if(property.name='NegativelyAffectedQAs' and not property.typedValue.oclIsUndefined()){
 affectedQAs<-property.typedValue.stringValue.split(',');
 for(affectedQA in affectedQAs){
   if(affectedQA.trim() <> ''){
     QAsMeet<- QAsMeet - thisModule.annotatedUC(thisModule.getStereotype(affectedQA.trim()));
   }
 }
}
--Effects of combination with previously selected design patterns added to the priority value
if(property.name='FrameworkCombinationAffectedQAs' and not property.typedValue.oclIsUndefined()){
 relatedFrameworks<-property.typedValue.stringValue.split('#');
 for(relatedFramework in relatedFrameworks){
   relatedFrameworkAffectedQAsInfo<-relatedFramework.trim().split(':');
   relatedFrameworkName<-relatedFrameworkAffectedQAsInfo.first();
   if(thisModule.isFrameworkSelected(relatedFrameworkName))
```

```
{
 relatedFrameworksAffectedQAs
    <-
    relatedFrameworkAffectedQAsInfo
    .last().split(';')
    ;
 affectedQAs<-
    relatedFrameworksAffectedQAs
    .first().split(','
    );
 for(affectedQA in
    affectedQAs){
   if(affectedQA.
       trim() <> ''){
        QAsMeet<-
            QAsMeet +
            thisModule
            .
            annotatedUC
            (
            thisModule
            .
            getStereotype
            (
            affectedQA
            .trim()));
   }
 }
 affectedQAs<-
    relatedFrameworksAffectedQAs
    .last().split(',')
    ;
 for(affectedQA in
    affectedQAs){
   if(affectedQA.
       trim() <> ''){
        QAsMeet<-
            QAsMeet -
            thisModule
            .
            annotatedUC
            (
            thisModule
            .
            getStereotype
            (
            affectedQA
```

```
                                                                    .trim()));
                                                                 }
                                                             }
                                                         }
                                                     }
                                                 }
                                             }
                                             --Select the Design pattern with
                                                 higher priority value
                                             if(QAsMeet > selectedQAsMeet){
                                                 selectedQAsMeet<-QAsMeet;
                                                 selectedFramework<-feature;
                                             }
                                         }
                                         if(not selectedFramework.oclIsUndefined()
                                             ){
                                             thisModule.selectedFrameworks<-
                                                 thisModule.selectedFrameworks.
                                                 append(selectedFramework);
                                         }
                                     }
                                 }
                                 if(thisModule.selectedFrameworks->includes(f)){
                                     t.state<-#USER_SELECTED;
                                 }
                             }
}

rule typedValueCopy{
    from f: FMP!TypedValue
    to t: FMP!TypedValue
        (integerValue<-f.integerValue,
            stringValue<-f.stringValue,
            floatValue<-f.floatValue,
            featureValue<-f.featureValue)
}

rule constraintCopy{
    from f: FMP!Constraint
    to t: FMP!Constraint (text<-f.text)
}

rule referenceCopy{
    from f : FMP!Reference
    to t : FMP!Reference
        (max<-f.max,
            min<-f.min,
                    id<-f.id,
                    children<-f.children,
                    confs<-f.confs,
                    origin<-f.origin,
                    state<-f.state,
                    clones<-f.clones,
                    prototype<-f.prototype,
                    feature<-f.feature)
}

rule featureGroupCopy{
     from f : FMP!FeatureGroup
     to t : FMP!FeatureGroup
          (max<-f.max,
               min<-f.min,
               id<-f.id,
               children<-f.children,
               confs<-f.confs,
               origin<-f.origin)
}

helper context FMP!Feature def:
    isSelectedLayerConfiguration(): Boolean =
     if (self.isManagementConfiguration() or
               self.isInversionOfControlConfiguration() or
               self.isPersistenceConfiguration() or
               self.isPresentationConfiguration() or
               self.isReportConfiguration() or
               self.isWebServicesConfiguration() or
               self.isTestConfiguration() or
               self.isLogConfiguration() or
               self.isSecurityConfiguration()) then
          true
     else
          false
     endif;

helper context FMP!Feature def: isManagementConfiguration
    (): Boolean =
     if (self.name='Management' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
          true
     else
          false
     endif;

```

```
helper context FMP!Feature def:
    isInversionOfControlConfiguration(): Boolean =
     if (self.name='InversionOfControl' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
          true
     else
          false
     endif;

helper context FMP!Feature def:
    isPersistenceConfiguration(): Boolean =
     if (self.name='Persistence' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
          true
     else
          false
     endif;

helper context FMP!Feature def:
    isPresentationConfiguration(): Boolean =
     if (self.name='Presentation' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
          true
     else
          false
     endif;

helper context FMP!Feature def: isReportConfiguration():
    Boolean =
     if (self.name='Report' and self.state=#USER_SELECTED
        and not self.origin.oclIsUndefined()) then
          true
     else
          false
     endif;

helper context FMP!Feature def:
    isWebServicesConfiguration(): Boolean =
     if (self.name='WebServices' and self.state=#
        USER_SELECTED and not self.origin.oclIsUndefined()
        ) then
          true
     else
          false
    endif;

helper context FMP!Feature def: isTestConfiguration():
    Boolean =
    if (self.name='Test' and self.state=#USER_SELECTED
        and not self.origin.oclIsUndefined()) then
        true
    else
        false
    endif;

helper context FMP!Feature def: isLogConfiguration():
    Boolean =
    if (self.name='Log' and self.state=#USER_SELECTED and
        not self.origin.oclIsUndefined()) then
        true
    else
        false
    endif;

helper context FMP!Feature def: isSecurityConfiguration()
    : Boolean =
    if (self.name='Security' and self.state=#USER_SELECTE
        D and not self.origin.oclIsUndefined()) then
        true
    else
        false
    endif;

-- Return a Sequence containing all the frameworks
    available for the implementation of the selected
    design patterns
helper def : getFrameworks : Sequence(Sequence(FMP!
    Feature)) =
    FMP!Feature.allInstances()->select(layer|layer.
        isSelectedLayerConfiguration())->collect(feature|
        feature.children.first().children)->flatten()->
        select(feature|feature.state=#USER_SELECTED)->
        collect(dessignPattern|dessignPattern.children.
        first().children)
;

-- Return the number of UC annotated with a given
    Stereotype
helper def : annotatedUC(s : UCP!Stereotype) : Integer =
    UML!UseCase.allInstances()->collect(ext | ext.
        extensionPoint).flatten()->select(exten | not
```

```
            exten.getAppliedStereotype(s.qualifiedName).
            oclIsUndefined()).size()
;

-- Return a Stereotype by its name
helper def : getStereotype(stereotypeName : String) : UCP
    !Stereotype =
    UCP!Stereotype.allInstances()->select(stereotype|
        stereotype.name=stereotypeName)->first()
;

-- Return a Framework by its name
helper def : getFrameworkByName(frameworkName : String) :
    FMP!Feature =
    FMP!Feature.allInstances()->select(framework|
        framework.name=frameworkName and not framework.
        origin.oclIsUndefined())->first()
;

helper def: iFrameworkSelected(framework: String):
    Boolean =
    if (thisModule.selectedFrameworkss->includes(
        thisModule.getFrameworkByName(framework))) then
        true
    else
        false
    endif;
```

Listing C.4: Excerpt of the framework suggestion transformation

C.4 Specific design transformation

An excerpt of the model transformation that generates a design adapted to the architecture designed by the architect is showed in Listing C.5

```
-- @path AP=/Models/ADProfile.profile.uml
-- @nsURI Frameworks=Frameworks
-- @nsURI UML=http://www.eclipse.org/uml2/4.0.0/UML
-- @nsURI FMP=http:///fmp.ecore

module ClassDiagramTansformation;
create ClassOut : UML from FeaturesIn : FMP, ActivityIn :
    UML, ActivityProfileIn : AP, FrameworksInfoIn :
    Frameworks;

```

```
helper def : model : UML!Model = OclUndefined;

rule Model {
    from s : UML!Model
    to t : UML!Model (
        name <- s.name,
        packagedElement <- Sequence{})
    do{
        thisModule.model<-t;
    }
}

rule OpaqueAction {
    from s : UML!OpaqueAction
    using{
        class: UML!Class = OclUndefined;
        informationItem: UML!InformationItem =
            OclUndefined;
        dependency: UML!Dependency = OclUndefined;
        contents: Sequence(OclAny) = OclUndefined;
        selectedFrameworks: Sequence (FMP!Feature) =
            OclUndefined;
        appliedDesignPattern: String = OclUndefined;
        appliedFramework: String = OclUndefined;
        frameworkInformation: Frameworks!Framework =
            OclUndefined;
    }
    do{
        for(partition in s.inPartition){
            appliedFramework <- OclUndefined;
            appliedDesignPattern <- OclUndefined;
            if(thisModule.partitionIsLayer(partition)){
                selectedFrameworks <- thisModule.
                    selectedFrameworks(partition.name);
                if(selectedFrameworks.size() = 1){
                    appliedFramework <-
                        selectedFrameworks.first().name;
                    appliedDesignPattern <- thisModule.
                        selectedPatterns(partition.name).
                        first().name;
                }
                else{
                    for(annotation in s.eAnnotations){
                        if(annotation.source=partition.
                            name){
                            appliedDesignPattern <-
                                annotation.details.first()
                    .key;
                  appliedFramework <- annotation.details.first().value;
                }
              }
            }
            if(not appliedFramework.oclIsUndefined()){
              frameworkInformation <- thisModule.getFrameworkInformation(appliedFramework);
              if(not frameworkInformation.oclIsUndefined()){
                for(concept in frameworkInformation.concepts){
                  if(concept.name = appliedDesignPattern){
                    contents <- Sequence{};
                    for(content in concept.contents){
                      if(content.oclIsTypeOf(Frameworks!XmlContent)){
                        informationItem <- UML!InformationItem.newInstance();
                        informationItem.name <- content.fileName;
                        thisModule.model.packagedElement <-thisModule.model.packagedElement.append(informationItem);
                        contents <- contents.append(informationItem
                                        );
                                    for(reference in
                                        content.
                                        references){
                                         dependency <-
                                              UML!
                                              Dependency
                                              .
                                              newInstance
                                              ();
                                         dependency.
                                              supplier
                                              <-
                                              informationItem
                                              ;
                                         for(c in
                                              contents){
                                               if(c.name
                                                   .
                                                   endsWith
                                                   (
                                                   reference
                                                   .
                                                   fileName
                                                   )){
                                                    dependency
                                                        .
                                                        client

                                                        <-
                                                         c
                                                        ;
                                               }
                                         }
                                         thisModule.
                                              model.
                                              packagedElement
                                              <-
                                              thisModule
                                              .model.
                                              packagedElement
                                              .append(
                                              dependency
                                              );
                                    }
                               }
                               else{
```

```
                                        if(content.isConfiguration){
                                            informationItem <- UML!InformationItem.newInstance();
                                            informationItem.name <- content.fileName;
                                            thisModule.model.packagedElement <- thisModule.model.packagedElement.append(informationItem);
                                            contents <- contents.append(informationItem);
                                            for(reference in content.references){
                                                dependency <- UML!Dependency.newInstance();
                                                dependency.supplier <- informationItem;
```

```
                                                        for(c in
                                                            contents
                                                            ){
                                                              if(c.
                                                                  name
                                                                  .
                                                                  endsWith
                                                                  (
                                                                  reference
                                                                  .
                                                                  fileName
                                                                  ))
                                                                  {
                                                                    dependency
                                                                        .
                                                                        client

                                                                        <-

                                                                        c
                                                                        ;

                                                              }
                                                        }
                                                        thisModule
                                                            .model
                                                            .
                                                            packagedElement
                                                            <-
                                                            thisModule
                                                            .model
                                                            .
                                                            packagedElement
                                                            .
                                                            append
                                                            (
                                                            dependency
                                                            );
                                                  }
                                           }
                                           else{
                                                  class <- UML!
                                                      Class.
                                                      newInstance
                                                      ();
                                                  class.name <-
                                                        s.name +
                                                      content.
                                                      fileName;
                                                  thisModule.
                                                      model.
                                                      packagedElement
                                                      <-
                                                      thisModule
                                                      .model.
                                                      packagedElement
                                                      .append(
                                                      class);
                                                  contents <-
                                                      contents.
                                                      append(
                                                      class);
                                                  for(reference
                                                        in
                                                      content.
                                                      references
                                                      ){
                                                        dependency
                                                            <-
                                                           UML!
                                                            Dependency
                                                            .
                                                            newInstance
                                                            ();
                                                        dependency
                                                            .
                                                            supplier
                                                            <-
                                                            class;
                                                        for(c in
                                                            contents
                                                            ){
                                                              if(c.
                                                                  name
                                                                  .
                                                                  endsWith
                                                                  (
                                                                  reference
                                                                  .
                                                                  fileName
                                                                  ))
                                                                  {
                                                                    dependency
                                                                        .
                                                                        client
                                                                        <-
                                                                        c
                                                                        ;
                                                              }
                                                        }
                                                        thisModule
                                                            .model
                                                            .
                                                            packagedElement
                                                            <-
                                                            thisModule
                                                            .model
                                                            .
                                                            packagedElement
                                                            .
                                                            append
                                                            (
                                                            dependency
                                                            );
                                                  }
                                              }
                                          }
                                      }
                                  }
                              }
                          }
                      }
                  }
              }
          }
  }

  helper def: getFrameworkInformation(frameworkName :
      String): Frameworks!Framework =
       Frameworks!Framework.allInstances()->select(framework
           |framework.name=frameworkName).first();

  helper def: partitionIsLayer(partition : UML!
      ActivityPartition): Boolean =
       if (FMP!Feature.allInstances()->select(feature|
           feature.name=partition.name)->size() > 0) then
           true
       else
```

```
            false
        endif;

helper def: selectedFrameworks(layerName : String):
    Sequence(FMP!Feature) =
     FMP!Feature.allInstances()->select(layer|layer.
         isSelectedLayerConfiguration(layerName))
          ->collect(feature|feature.children.first().
              children)->flatten()->select(feature|feature.
              state=#USER_SELECTED)
               ->collect(dessignPattern|dessignPattern.
                   children.first().children)->flatten()->
                   select(feature|feature.state=#USER_SELECTE
                  D);

helper def: selectedPatterns(layerName : String):
    Sequence(FMP!Feature) =
     FMP!Feature.allInstances()->select(layer|layer.
         isSelectedLayerConfiguration(layerName))
          ->collect(feature|feature.children.first().
              children)->flatten()->select(feature|feature.
              state=#USER_SELECTED);

helper context FMP!Feature def:
    isSelectedLayerConfiguration(layerName : String):
    Boolean =
      if (self.name=layerName and self.state=#USER_SELECTED
           and not self.origin.oclIsUndefined()) then
            true
        else
            false
        endif;
```

Listing C.5: Excerpt of the specific design transformation

Appendix D

Additional material

To contribute to the completeness of this thesis, a web page has been created from which all the materials related to this thesis and its contributions can be downloaded. This page can be found at `http://www.gloin.es/garcia-alonso_thesis/` and contain the following elements.

- The architectural decision repository described in Section 3.3.
- The framework information meta-model described in Section 3.4.
- The set of ATL model transformations described in Section 3.5.
- The code generation tool described in Chapter 4, along with all the additional material for the tool detailed in section 4.5

References

Ågerfalk, P. J., & Fitzgerald, B. (2008). Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy. *MIS Quarterly*, *32*(2), 385-409.

Andova, S., van den Brand, M. G. J., Engelen, L. J. P., & Verhoeff, T. (2012). Mde basics with a dsl focus. In M. Bernardo, V. Cortellessa, & A. Pierantonio (Eds.), *Sfm* (Vol. 7320, p. 21-57). Springer.

Antkiewicz, M. (2007). Round-trip engineering using framework-specific modeling languages. In R. P. Gabriel, D. F. Bacon, C. V. Lopes, & G. L. S. Jr. (Eds.), *Oopsla companion* (p. 927-928). ACM.

Antkiewicz, M., & Czarnecki, K. (2004). Featureplugin: feature modeling plug-in for eclipse. In M. G. Burke (Ed.), *Etx* (p. 67-72). ACM.

Antkiewicz, M., Czarnecki, K., & Stephan, M. (2009). Engineering of framework-specific modeling languages. *IEEE Trans. Software Eng.*, *35*(6), 795-824.

Atkinson, C., & Kühne, T. (2003). Model-driven development: A metamodeling foundation. *IEEE Software*, *20*(5), 36-41.

Avgeriou, P., & Zdun, U. (2005). Architectural patterns revisited - a pattern language. In A. Longshaw & U. Zdun (Eds.), *Europlop* (p. 431-470). UVK - Universitaetsverlag Konstanz.

Babar, M. A. (2004). Scenarios, quality attributes, and patterns: Capturing and using their synergistic relationships for product line architectures. In *Apsec* (p. 574-578). IEEE Computer Society.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. In *Encyclopedia of software engineering.* Wiley.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice, second edition.* Addison-Wesley Professional.

Benavides, D., Segura, S., & Cortés, A. R. (2010). Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, *35*(6), 615-636.

Bengtsson, P. (1998). Towards maintainability metrics on software architecture: An adaptation of object-oriented metrics. In *First nordic workshop on software architecture, ronneby.*

Bernardo, M., Cortellessa, V., & Pierantonio, A. (Eds.). (2012). *Formal methods for model-driven engineering - 12th international school on formal methods for the design of computer, communication, and software systems, sfm 2012, bertinoro, italy, june 18-23, 2012. advanced lectures* (Vol. 7320). Springer.

Bézivin, J., & Gerbé, O. (2001). Towards a precise definition of the omg/mda framework. In *Ase* (p. 273-280). IEEE Computer Society.

Boehm, B. W. (2006). A view of 20th and 21st century software engineering. In L. J. Osterweil, H. D. Rombach, & M. L. Soffa (Eds.), *Icse* (p. 12-29). ACM.

Bolchini, D., & Garzotto, F. (2008). Designing multichannel web applications as "dialogue systems": the idm model. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 193-219). Springer.

Bolchini, D., & Paolini, P. (2006). Interactive dialogue model: a design technique for multichannel applications. *IEEE Transactions on Multimedia*, *8*(3), 529-541.

Bosch, J. (2000). *Design and use of software architectures - adopting and evolving a product-line approach.* Addison-Wesley.

Bosch, J., Molin, P., Mattsson, M., & Bengtsson, P. (1997). *Object-oriented frameworks – problems & experiences.*

Brambilla, M., Comai, S., Fraternali, P., & Matera, M. (2008). Designing web applications with webml and webratio. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 221-261). Springer.

Cabot, J., & Gogolla, M. (2012). Object constraint language (ocl): A definitive guide. In M. Bernardo, V. Cortellessa, & A. Pierantonio (Eds.), *Sfm* (Vol. 7320, p. 58-90). Springer.

Capilla, R., Babar, M. A., & Pastor, O. (2012). Quality requirements engineering for systems and software architecting: methods, approaches, and tools. *Requir.*

Eng., *17*(4), 255-258.

Carmel, E., & Abbott, P. (2007). Why 'nearshore' means that distance matters. *Commun. ACM*, *50*(10), 40-46.

Centro de Información Cartográfica y Territorial de Extremadura. (2013). *Pliego de prescripciones técnicas que han de regir el servicio titulado registro cartográfico extremeño.*

Ceri, S., Brambilla, M., & Fraternali, P. (2009). The history of webml lessons learned from 10 years of model-driven development of web applications. In A. Borgida, V. K. Chaudhri, P. Giorgini, & E. S. K. Yu (Eds.), *Conceptual modeling: Foundations and applications* (Vol. 5600, p. 273-292). Springer.

Ceri, S., Fraternali, P., & Bongio, A. (2000). Web modeling language (webml): a modeling language for designing web sites. *Computer Networks*, *33*(1-6), 137-157.

Chen, L., & Babar, M. A. (2011). A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, *53*(4), 344 - 362.

Clements, P. C. (2001). On the importance of product line scope. In F. van der Linden (Ed.), *Pfe* (Vol. 2290, p. 70-78). Springer.

Clements, P. C., Kazman, R., Klein, M., Devesh, D., Reddy, S., & Verma, P. (2007). The duties, skills, and knowledge of software architects. In *Wicsa* (p. 20). IEEE Computer Society.

Clemmons, R. K. (2006). Project Estimation With Use Case Points. *Diversified Technical Services, Inc. CrossTalk - The journal of Defence Software Engineering*, 18-22.

Conejero, J. M., Rodríguez-Echeverría, R., Sánchez-Figueroa, F., Trigueros, M. L., Preciado, J. C., & Clemente, P. J. (2013). Re-engineering legacy web applications into rias by aligning modernization requirements, patterns and ria features. *Journal of Systems and Software*, *86*(12), 2981-2994.

Consejería Administración Pública. (2013). *Pliego de prescripciones técnicas que han de regir el servicio para el análisis y desarrollo del sistema de gestión de planes urbanísticos y su integración con el proyecto e-gobeex.*

Cxf development framework. (n.d.). http://cxf.apache.org.

Czarnecki, K., & Helsen, S. (2006, July). Feature-based survey of model transformation approaches. *IBM Syst. J.*, *45*(3), 621–645.

Czarnecki, K., Helsen, S., & Eisenecker, U. W. (2005a). Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, *10*(1), 7-29.

Czarnecki, K., Helsen, S., & Eisenecker, U. W. (2005b). Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, *10*(2), 143-169.

Dalgarno, M. (2009). When good architecture goes bad. *Methods & Tools*, *17*(1), 27-34.

Davis, E., Parker, A., & Shanahan, A. (2009). *Complementing India with nearshore strategies: Spotlight on Spain.* Forrester. Europe's Offshore Landscape. Retrieved from http://www.forrester.com/Complementing+India+With+Nearshore+Strategies+Spotlight+On+Spain/fulltext/-/E-RES46904

de Andalucía, J. (n.d.). *Marco de desarrollo de la junta de andalucía.* Retrieved from http://www.juntadeandalucia.es/servicios/madeja/

de Asturias, P. (n.d.). *Open framework del principado de asturias.* Retrieved from http://www.asturias.es/portal/site/OpenFWPA

de Cantabria, G. (n.d.). *Arquitectura marco para las administraciones públicas.* Retrieved from http://amap.cantabria.es/

de Extremadura Press Release, P. (2008, June). *La consultora accenture firma con las empresas xtrem y cie acuerdos.* Retrieved from http://www.elperiodicoextremadura.com/noticias/merida/consultora-accenture-firma-empresas-xtrem-cie-acuerdos_377333.html

de Murcia, R. (n.d.). *Framework javato.* Retrieved from https://www.carm.es/web/pagina?IDCONTENIDO=29980&IDTIPO=100&RASTRO=c814$m4394

Dwr development framework. (n.d.). http://directwebremoting.org/dwr/index.html.

Fatolahi, A., Somé, S. S., & Lethbridge, T. C. (2008). A model-driven approach for the semi-automated generation of web-based applications from requirements. In *Seke* (p. 619-624). Knowledge Systems Institute Graduate School.

Fatolahi, A., Somé, S. S., & Lethbridge, T. C. (2012). A meta-model for model-driven web development. *Int. J. Software and Informatics*, *6*(2), 125-162.

Fons, J., Pelechano, V., Pastor, O., Valderas, P., & Torres, V. (2008). Applying the oows model-driven approach for developing web applications. the internet movie database case study. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 65-108). Springer.

Fowler, M. (2002). *Patterns of enterprise application architecture.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Fraternali, P., Rossi, G., & Sánchez-Figueroa, F. (2010). Rich internet applications. *IEEE Internet Computing*, *14*(3), 9-12.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

García-Alonso, J., Olmeda, J. B., & Murillo, J. M. (2010). Java para aplicaciones corporativas de la administración. In E. Teniente & S. Abrahão (Eds.), *Jisbd* (p. 263-266). IBERGARCETA Pub. S.L.

García-Alonso, J., Olmeda, J. B., & Murillo, J. M. (2012). Architectural variability management in multi-layer web applications through feature models. In I. Schaefer & T. Thüm (Eds.), *Fosd@gpce* (p. 29-36). ACM.

García-Alonso, J., Olmeda, J. B., & Murillo, J. M. (2013). Architectural decisions in the development of multi-layer applications. In *Icsea 2013, the eighth international conference on software engineering advances* (p. 214-219).

García-Alonso, J., Olmeda, J. B., & Murillo, J. M. (2014a). Model transformations for the automatic suggestion of architectural decisions in the development of multi-layer applications. In *Icsea 2014, 9th international conference on software engineering advances.*

García-Alonso, J., Olmeda, J. B., & Murillo, J. M. (2014b). Technological variability by means of a framework metamodel. In *Sera (to be published).*

Garzotto, F., Paolini, P., & Schwabe, D. (1993). Hdm - a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.*, *11*(1), 1-26.

Gómez, A., & Ramos, I. (2010). Cardinality-based feature modeling and model-driven engineering: Fitting them together. In D. Benavides, D. S. Batory, & P. Grünbacher (Eds.), *Vamos* (Vol. 37, p. 61-68). Universität Duisburg-Essen.

Guillén, J., Miranda, J., Berrocal, J., García-Alonso, J., Murillo, J. M., & Canal, C. (2014). People as a service: A mobile-centric model for providing collective sociological profiles. *IEEE Software*, *31*(2), 48-53.

Harrison, N. B., & Avgeriou, P. (2010). How do architecture patterns and tactics interact? a model and annotation. *Journal of Systems and Software*, *83*(10), 1735-1758.

Harrison, N. B., Avgeriou, P., & Zdun, U. (2007). Using patterns to capture architectural decisions. *IEEE Software*, *24*(4), 38-45.

Hennicker, R., & Koch, N. (2000). A uml-based methodology for hypermedia design. In A. Evans, S. Kent, & B. Selic (Eds.), *Uml* (Vol. 1939, p. 410-424). Springer.

Heydarnoori, A., Czarnecki, K., & Bartolomei, T. T. (2009). Supporting framework use via automatically extracted concept-implementation templates. In S. Drossopoulou (Ed.), *Ecoop* (Vol. 5653, p. 344-368). Springer.

Heydarnoori, A., Czarnecki, K., Binder, W., & Bartolomei, T. T. (2012). Two studies of framework-usage templates extracted from dynamic traces. *IEEE Trans. Software Eng.*, *38*(6), 1464-1487.

Hou, D., & Li, L. (2011). Obstacles in using frameworks and apis: An exploratory study of programmers' newsgroup discussions. In *Icpc* (p. 91-100). IEEE Computer Society.

Hou, D., Wong, K., & Hoover, H. J. (2005). What can programmer questions tell us about frameworks? In *Iwpc* (p. 87-96). IEEE Computer Society.

Houben, G.-J., van der Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., & Frasincar, F. (2008). Hera. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 263-301). Springer.

ISO. (2005). *ISO/IEC 19502:2005 information technology – Meta Object Facility (MOF).* Retrieved from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32621

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Jet templating engine. (n.d.). http://www.eclipse.org/modeling/m2t/?project=jet.

Johnson, R. (2005). J2ee development frameworks. *IEEE Computer*, *38*(1), 107-

110.

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., ... Webb, P. (2013a). Integrating with other web frameworks. In *Spring framework reference documentation.* Spring Framework. Retrieved from http://static.springsource.org/spring/docs/3.2.4.RELEASE/spring-framework-reference/html/web-integration.html

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., ... Webb, P. (2013b). Integration. In *Spring framework reference documentation.* Spring Framework. Retrieved from http://static.springsource.org/spring/docs/3.2.4.RELEASE/spring-framework-reference/html/spring-integration.html

Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). Atl: A model transformation tool. *Sci. Comput. Program.*, *72*(1-2), 31-39.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., & Valduriez, P. (2006). Atl: a qvt-like transformation language. In P. L. Tarr & W. R. Cook (Eds.), *Oopsla companion* (p. 719-720). ACM.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-oriented domain analysis (foda): Feasibility study.*

Kazman, R., Asundi, J., & Klein, M. (2001). Quantifying the costs and benefits of architectural decisions. In H. A. Müller, M. J. Harrold, & W. Schäfer (Eds.), *Icse* (p. 297-306). IEEE Computer Society.

Kleppe, A. G., Warmer, J., & Bast, W. (2003). *MDA explained: The Model Driven Architecture: Practice and promise.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Klyne, G., & Carroll, J. J. (2004, February). *Resource description framework (RDF): Concepts and abstract syntax.* World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210.

Knapp, A., Koch, N., & Zhang, G. (2005). Modelling the behaviour of web applications with argouwe. In D. B. Lowe & M. Gaedke (Eds.), *Icwe* (Vol. 3579, p. 624-626). Springer.

Koch, N., Knapp, A., Zhang, G., & Baumeister, H. (2008). Uml-based web engineering - an approach based on standards. In G. Rossi, O. Pastor, D. Schwabe,

& L. Olsina (Eds.), *Web engineering* (p. 157-191). Springer.

Kruchten, P. (2008). What do software architects really do? *Journal of Systems and Software*, *81*(12), 2413-2416.

Lacity, M., Carmel, E., & Rottman, J. W. (2011). Rural outsourcing: Delivering ito and bpo services from remote domestic locations. *IEEE Computer*, *44*(12), 55-62.

Leffingwell, D., & Widrig, D. (2003). *Managing software requirements: a use case approach.* Addison-Wesley Professional.

Lung, C.-H., & Kalaichelvan, K. (2000). An approach to quantitative software architecture sensitivity analysis. *International Journal of Software Engineering and Knowledge Engineering*, *10*(1), 97-114.

Mattsson, M., Bosch, J., & Fayad, M. E. (1999). Framework integration problems, causes, solutions. *Communications of the ACM*, *42*(10), 80–87.

Maven guide to naming conventions. (n.d.). http://maven.apache.org/guides/mini/guide-naming-conventions.html.

Meliá, S., & Gómez, J. (2006). The websa approach: Applying model driven engineering to web applications. *J. Web Eng.*, *5*(2), 121-149.

Meliá, S., Gómez, J., Pérez, S., & Díaz, O. (2010). Architectural and technological variability in rich internet applications. *IEEE Internet Computing*, *14*(3), 24-32.

Miller, J., & Mukerji, J. (2003). *MDA guide version 1.0.1.*

Mohan, K., & Ramesh, B. (2003, Jan). Ontology-based support for variability management in product and families. In *System sciences, 2003. proceedings of the 36th annual hawaii international conference on* (p. 9 pp.-).

Moreno, N., Romero, J. R., & Vallecillo, A. (2008). An overview of model-driven web engineering and the mda. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 353-382). Springer.

Moreno, N., & Vallecillo, A. (2008). Towards interoperable web engineering methods. *JASIST*, *59*(7), 1073-1092.

Murugesan, S., & Deshpande, Y. (1999). Icse'99 workshop on web engineering. In B. W. Boehm, D. Garlan, & J. Kramer (Eds.), *Icse* (p. 693-694). ACM.

Nascimento, V., & Schwabe, D. (2013). Semantic data driven interfaces for web applications. In F. Daniel, P. Dolog, & Q. Li (Eds.), *Icwe* (Vol. 7977, p. 22-36). Springer.

Northrop, L. (2003). The importance of software architecture. *Software Engineering Institute, Carnegie Mellon University*. Retrieved from http://sunset.usc.edu/GSAW/gsaw2003/s13/northrop.pdf

of Extremadura Press Release, U. (2005, March). *Se abre la primera fábrica universitaria de software.* Retrieved from http://noticias.universia.es/ciencia-nn-tt/noticia/2005/03/15/608437/abre-primera-fabrica-universitaria-software.html

OMG. (2011a, January). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1.* Retrieved from http://www.omg.org/spec/QVT/1.1/

OMG. (2011b, August). *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1.* Retrieved from http://www.omg.org/spec/MOF/2.4.1

OMG. (2012, January). *OMG Object Constraint Language (OCL), Version 2.3.1.* Retrieved from http://www.omg.org/spec/OCL/2.3.1/

OMG. (2013, March). *Interaction Flow Modeling Language (IFML).* Retrieved from http://www.omg.org/spec/IFML/

Organization, I. S. (2011). *ISO/IEC 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models.* Retrieved from http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733

Paraschiv, E. (2013). *Spring security with maven.* Retrieved from http://www.baeldung.com/spring-security-with-maven

Pohl, K., Böckle, G., & van der Linden, F. (2005). *Software product line engineering - foundations, principles, and techniques.* Springer.

Preciado, J. C., Trigueros, M. L., & Sánchez-Figueroa, F. (2008). Enriching model-based web applications presentation. *J. Web Eng.*, *7*(3), 239-256.

Preciado, J. C., Trigueros, M. L., Sánchez-Figueroa, F., & Comai, S. (2005). Necessity of methodologies to model rich internet applications. In *Wse* (p. 7-13). IEEE Computer Society.

Pressman, R. S. (2000). Manager - What a tangled web we weave. *IEEE Software*, *17*(1).

Prikladnicki, R., Audy, J. L. N., Damian, D., & de Oliveira, T. C. (2007). Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring. In *Icgse* (p. 262-274). IEEE.

Prikladnicki, R., Audy, J. L. N., & Shull, F. (2010). Patterns in effective distributed software development. *IEEE Software*, *27*(2), 12-15.

Raible, M. (2007). *Comparing java web frameworks.* Apache convention. Retrieved from http://static.raibledesigns.com/repository/presentations/ComparingJavaWebFrameworks-ApacheConUS2007.pdf

Raible, M. (2012). *Comparing JVM web frameworks.* Jfokus. Retrieved from http://static.raibledesigns.com/repository/presentations/Comparing_JVM_Web_Frameworks_Jfokus2012.pdf

Release, I.-I. P. (2007, February). *Apertura del centro de innovación tecnológica de cáceres.* Retrieved from http://www.insags.com/noticias/cenit_caceres.html

Release, I. P. (2012, June). *El software lab de indra en badajoz ha generado más de 120 empleos cualificados en menos de tres años.* Retrieved from http://www.indracompany.com/noticia/el-software-lab-de-indra-en-badajoz-ha-generado-mas-de-120-empleos-cualificados-en-menos-de-

Release, I. P. (2013, November). *El presidente monago inaugura la nueva factoría de software de mérida.* Retrieved from http://www.ibermatica.com/sala-de-prensa/noticias/el-presidente-monago-inaugura-la-nueva-factoria-de-software-de-merida

Rodríguez-Echeverría, R., Pavón, V. M., Macías, F., Conejero, J. M., Clemente, P. J., & Sánchez-Figueroa, F. (2013). Generating a conceptual representation of a legacy web application. In X. Lin, Y. Manolopoulos, D. Srivastava, & G. Huang (Eds.), *Wise (2)* (Vol. 8181, p. 231-240). Springer.

Rossi, G. (2013). Web modeling languages strike back. *IEEE Internet Computing*, *17*(4), 4-6.

Rossi, G., Pastor, O., Schwabe, D., & Olsina, L. (Eds.). (2008). *Web engineering: Modelling and implementing web applications*. Springer.

Rossi, G., & Schwabe, D. (2008). Modeling and implementing web applications with oohdm. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 109-155). Springer.

Ruscio, D. D., Eramo, R., & Pierantonio, A. (2012). Model transformations. In M. Bernardo, V. Cortellessa, & A. Pierantonio (Eds.), *Sfm* (Vol. 7320, p. 91-136). Springer.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *IEEE Computer*, *39*(2), 25-31.

Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., & Bontemps, Y. (2007). Generic semantics of feature diagrams. *Computer Networks*, *51*(2), 456-479.

Schwabe, D., & Rossi, G. (1998). An object oriented approach to web-based applications design. *TAPOS*, *4*(4), 207-225.

Shan, T. C., & Hua, W. W. (2006). Taxonomy of java web application frameworks. In *Icebe* (p. 378-385). IEEE Computer Society.

Shaw, M. (2002). What makes good research in software engineering. *for Technology Transfer (STTT). Springer Berlin / Heidelberg*, *4*, 1-7.

Snell, J., Tidwell, D., & Kulchenko, P. (2001). *Programming web services with soap*. O'Reilly Media.

Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *Emf: Eclipse modeling framework 2.0* (2nd ed.). Addison-Wesley Professional.

Struts development framework. (n.d.). http://struts.apache.org/.

Taentzer, G., Ehrig, K., Guerra, E., Lara, J. D., Levendovszky, T., Prange, U., ... et al. (2005). Model transformations by graph transformations: A comparative study. In *Model transformations in practice workshop at models 2005, montego.*

Tang, A., Avgeriou, P., Jansen, A., Capilla, R., & Babar, M. A. (2010). A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, *83*(3), 352-370.

Torres, V., Pelechano, V., & Pastor, O. (2006). Building semantic web services based on a model driven web engineering method. In J. F. Roddick et al. (Eds.), *Er (workshops)* (Vol. 4231, p. 173-182). Springer.

Trigueros, M. L., Preciado, J. C., & Sánchez-Figueroa, F. (2007). Engineering rich internet application user interfaces over legacy web models. *IEEE Internet*

Computing, *11*(6), 53-59.

Troyer, O. D., Casteleyn, S., & Plessers, P. (2008). Wsdm: Web semantics design method. In G. Rossi, O. Pastor, D. Schwabe, & L. Olsina (Eds.), *Web engineering* (p. 303-351). Springer.

Troyer, O. D., & Leune, C. J. (1998). Wsdm: A user centered design method for web sites. *Computer Networks*, *30*(1-7), 85-94.

Uml2copy atl transformation. (n.d.). http://soft.vub.ac.be/viewvc/UML2CaseStudies/uml2cs-transformations/transformations/UML2Copy.atl.

Van Solingen, R., & Berghout, E. (1999). *The goal/question/metric method: A practical guide for quality improvement of software development.* McGraw-Hill Higher Education.

Vdovjak, R., Frasincar, F., Houben, G.-J., & Barna, P. (2003). Engineering semantic web information systems in hera. *J. Web Eng.*, *2*(1-2), 3-26.

Velocity templating engine. (n.d.). http://velocity.apache.org/.

Vosloo, I., & Kourie, D. G. (2008). Server-centric web frameworks: An overview. *ACM Comput. Surv.*, *40*(2).

Webber, J., Parastatidis, S., & Robinson, I. (2010). *Rest in practice. hypermedia and systems architecture.* O'Reilly Media.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2000). *Experimentation in software engineering: An introduction.* Norwell, MA, USA: Kluwer Academic Publishers.

Wu, W., & Kelly, T. (2006). Managing architectural design decisions for safety-critical software systems. In C. Hofmeister, I. Crnkovic, & R. Reussner (Eds.), *Qosa* (Vol. 4214, p. 59-77). Springer.

Zimmermann, O. (2011). Architectural decisions as reusable design assets. *IEEE Software*, *28*(1), 64-69.

Zimmermann, O. (2012). Architectural decision identification in architectural patterns. In T. Männistö, M. A. Babar, C. E. Cuesta, & J. E. Savolainen (Eds.), *Wicsa/ecsa companion volume* (Vol. 704, p. 96-103). ACM.