INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

A MDE Approach for the Development of CMS-based Web Applications

Francisco José Cabral Cardoso

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Prof. José Manuel Nunes Salvador Tribolet
Vogal:	Prof. André Ferreira Ferrão Couto e Vasconcelos
Orientador:	Prof. Alberto Manuel Rodrigues da Silva
Co-Orientador:	Eng. João Paulo Pedro Mendes de Sousa Saraiva

Outubro 2009

Acknowledgements

We all have others with whom we work, with whom we talk, with whom we despair, with whom we laugh, and with whom we dream. All of these people, one way or another, influence what we do and who we are. The work leading to this dissertation was a long journey that would not have been possible without the help of many. To thank them all, I shall give a small text that does not do justice to all the help that I got. First and foremost, I would like to thank Professor Aberto Silva, my adviser, who gave me the opportunity to do this work, for his constant availability and guidance. I would like to thank, also, Eng. João Saraiva, my co-adviser, both for having initiated me into some of the technologies required in my research, and for his support throughout all my work.

I am grateful to Altitude Software, the company where I was working in the last year, for providing me not only the means to do my dissertation in parallel, but also an healthy and enjoyable environment to work in. I am deeply indebted with all the Altitude Software team, in particular, to my project managers, for all the numerous discussions that helped me to shed light into the more obscure parts of this work, motivation and comprehension.

I couldn't have reached this stage in my life if it wasn't for my mother, Sidónia Cabral, who always supported me and provided me with all she could. No words could even begin to express how much I truly owe her; thus, I can only try to express my gratitude, and my regret for the little availability and little time devoted to her over this last year of hard work.

Finally, regardless of all the remaining help, none of this work would have happened without the enduring support, encouragement, and sacrifices of my lovingly girlfriend Andreia Medeiros. I heart-feelingly thank her for all their patience and ability to lead me through the most difficult phases of my work. I thank her, also, for her analytical reasoning that helped me countless times during my work. Discussing the most difficult problems with her allowed me, time and again, to make things clearer in my head.

Lisboa, Thursday 22nd October, 2009
Francisco Cardoso

Abstract

Content Management Systems (CMS) are typically regarded as critical software platforms for the success of organizational web sites and intranets. Although most current CMS systems allow their extension through the addition of modules, these are usually built using the typical source-code-oriented software development process, which is slow and error-prone. On the other hand, a MDE-oriented development process is centered on models, which represent the system and are used to automatically generate all the artifacts of a system, such as source-code and documentation.

To simplify the development of CMS-based web applications, this dissertation proposes a Model-Driven Engineering approach to address the design and creation of CMS-based web applications in a graphical fashion, in the form of an UML Profile, which will permit the design of a web application using typical domain concepts (e.g., user, role, permission, web page), and subsequent code generation and deployment to a target platform. The UML modeling and model transformation is supported by ProjectIT-Studio tool, which was extended in the context of this work to support the proposed language.

We also propose a simple methodology, guidelines and inter-related processes for the development of CMS-based web applications using the UML-based domain specific language, named by CMS-ML, supported by the ProjectIT-Studio CASE tool for model design and transformation.

The dissertation validation is given along the work with simple examples and complemented with a complex and complete case study, which describes one application of the proposal, a fictional CMS-based web application: the *MyWebHome*.

Resumo

Os Sistemas de Gestão de Conteúdo (CMS) são geralmente considerados como plataformas de software críticas para o sucesso de web sites e intranets organizacionais. Embora a maioria dos sistemas CMS actuais permitirem a sua extensão através da adição de módulos, estes são geralmente construídos utilizando o processo de desenvolvimento de software típico baseado em código-fonte, que é lento e sujeito a erros. Por outro lado, um processo de desenvolvimento orientado ao MDE é centrado em modelos, que representam o sistema e são utilizados para gerar automaticamente todos os artefatos de um sistema, tais como o código-fonte e documentação.

Para simplificar o desenvolvimento de aplicações web baseadas em CMS, esta dissertação propõe uma abordagem de engenharia baseada em modelos para abordar a concepção e criação de aplicações web baseadas em CMS baseado de uma forma gráfica, na forma de um perfil UML, o que irá permitir a concepção de uma aplicação web utilizando conceitos de domínio típicos (por exemplo, utilizador, papel, permissão, página web), e geração de código e a posterior instalação numa plataforma de destino.

Propomos também uma metodologia simples, orientações e processos inter-relacionados para o desenvolvimento de aplicações web baseadas em CMS utilizando uma linguagem especifica a um domínio baseada no UML, denominada por CMS-ML, apoiada pela ferramenta CASE ProjectIT-Studio para a concepção do modelo e a sua transformação .

A validação da dissertação é dada ao longo do trabalho, com exemplos simples e complementada com um caso de estudo complexo e completo, que descreve uma aplicação da proposta, uma aplicação web baseada em CMS fictícia: o *MyWebHome*.

Keywords

Model Driven Engineering
Domain Specific Languages
Platform Independent Models
Content Management Systems
Model Transformation

Palavras Chave

Engenharia baseada em Modelos
Linguagens Específicas a um Domínio
Modelos Independentes da Plataforma
Sistemas de Gestão de Conteúdos
Transformação de Modelos

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation describes the work developed during one year at the Information Systems Groups at INESC-ID Lisboa, regarding the Master Thesis in Information Systems.

The research work presented in this dissertation is concerned with the specification of a domain specific language [81, 31, 77] for the development of CMS-based web applications [11], representing these web applications in a model that is portable across platforms, easy to use and with a high degree of expressiveness.

This introductory chapter starts with the identification of the work context and motivation, followed by the problem definition that is the origin of this research work. These two sections are crucial to present the thesis statement, which is done after these. Finally, the outline of the dissertation is presented.

1.1 Context

This dissertation describes the work developed during one year at the Information Systems Group of INESC-ID Lisboa, regarding the Master Thesis in Information Systems, in strict collaboration with the ProjectIT-Studio development team, and was supervised by PhD Alberto Manuel Rodrigues da Silva.

This work focuses in the development of CMS-based web applications [11] and takes place in the context of ProjectIT-MDD, which is ProjectIT's functional component related to the area of information systems modeling and Model-Driven Engineering (MDE) [56]. This component allows the specification of models, transformations between models defined in different languages, and the automatic generation of artifacts (such as source-code and documentation) [54]. Figure 1.1 presents a high-level overview of the ProjectIT-MDD architecture; the shaded package indicates the sub-component of ProjectIT-MDD which was developed in the context of this work - the CMS-ML Meta-Model.

The ProjectIT approach is language-independent, because the UMLModeler allows the architect to specify the semantics that are inherent to a specific UML profile. The XIS2 [36] is an example of an UML2 profile for the development of software systems with the ProjectIT-MDD, section 2.2.4 introduces this language with more details.

Besides the XIS2 meta-model the ProjectIT-MDD must also support another meta-model, one that is specifically focused in the CMS-based web applications domain.

This work describes the specification of a modeling language for the development of CMS-based web applications, which aims to offer more simplify in the task of complex CMS-based web applications development and increase overall quality and productivity; along with the creation of the ProjectIT-MDD sub-component to support the language meta-model, which is primarily focused on making the user look upon the ProjectIT approach as an easy and efficient way to model CMS-based web applications.

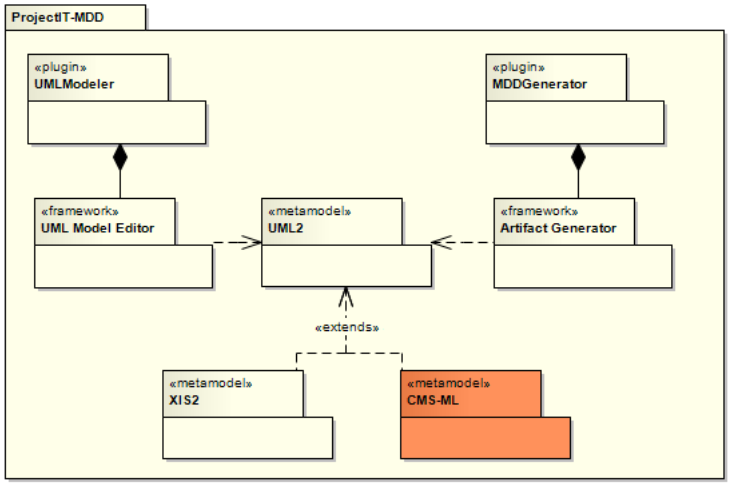


Figure 1.1: ProjectIT-MDD's architecture (adapted from [65])

1.2 Motivation

Since the born of the world wide web, several methodological approaches, such as model-driven development (MDD/MDE) [44], try to provide support for software applications by means of abstract mechanisms that make easy the conceptualization and development of applications [53]. The development approaches that enable the specification of applications independently of the target platform, have gained a large popularity in the scientific community and offers several benefits, identified in section 2.2.

Besides the MDE approach, there're several other approaches for the development of web applications, such as the classical software coding, visual web editors, web formularies, among others.

Along with the MDE approach [56], an initiative that has been gaining support recently is the usage of DSLs (Domain-Specific Languages) [77]. A DSL provides a different perspective of software development that consists of building abstractions, this way, a DSL can be viewed as a front-end to a framework, providing a different development style.

Web applications are a family of applications developed with various technologies and platforms. One of these web applications are the CMS-based web applications. These web applications are developed on top of CMS software platforms and present great potential not only because they automates site content management, but also allows nontechnical writers to contribute articles directly into the system via a web interface. These web applications also allows an easy user access management and a great level of extension by providing mechanisms that allows the addition of individual components (also known as modules) that can be bound to the web application [11, 53], section 2.1 presents a more detailed view over the CMS systems.

Surprisingly, not many proposal were presented for the development of CMS-based web applications, most of the existing proposals are based in visual web editors and web formularies. Thus there's an untapped opportunity that brings together the MDE approach and web applications based on CMS platforms, which has an associated set of advantages that combines the advantages of both sides.

1.3 Problem Definition

The previous sections described the context and the motivation of this dissertation work. This section describes the problems that originated this research work. The ultimate goal of this work is to significantly reduce most of the problems described in this section.

The development of web applications is a very complex activity, involving nontrivial issues in various areas such as technology and human resources. Despite the efforts made to overcome the issues related with this activity, these issues are still frequently found in web software development projects, with its negative consequent aspects in terms of project scope, time, budget and quality in the application and documentation.

Another issue in web development is that, nowadays, web application developers need to use many technologies, sometimes several different technologies in the same project; this influences their performance and the system quality. Also, most common development tools lack of a web application multiple views, making the application understanding and analysis a very difficult task.

The technology diversity makes development difficult in terms of reuse, portability and quality assurance, in all projects. Also, frequent changes of business needs can be observed in many cases, and web applications need to reflect such changes in a timely manner, which leads again to the development issues already mentioned.

Thus, currently are used frameworks for developing web applications, such as Django [14] and Ruby On Rails [49], which dramatically simplify and reduce the amount of developments. Although, these frameworks doesn't eliminates all the coding tasks nor does provides mechanisms for content management and extensions that allows the addition of modules to the web application.

As mentioned before, several types of web applications exist nowadays, one of them is the CMS-based web applications that are gaining more popularity in the web development community. However, there aren't relevant researches and proposals for development approaches of CMS-based web applications that follow a structured, standard and systematic approach, like the methodologies and best practices proposed by the software engineering community [63].

The current CMS platforms have difficulties to support the design and implementation of so-called data-intensive web applications, defined as web applications for accessing and maintaining large amounts of structured data, typically stored as records in a database management system, like online trading and e-commerce applications, institutional web sites of private and public organizations, digital libraries, corporate portals, and community sites.

For a large site with hundreds or thousands of pages and components that needs to be added or updated frequently with multiple authors, CMS-based web applications configuration and development becomes more difficult and complex. This means that the traditional method of browser-based development, used for the development of CMS-based web applications, loses efficiency when the site complexity increases.

Although there exists many MDE approaches for web and general applications, **currently there are no significant MDE approaches for CMS-based web applications**. Therefore, there's a possibility to explore on the development of CMS-based web applications, which led to the origin of this work.

The most critical issue to consider when developing a web application is the developer. The developers must have, not only an extensive web development experience, but must also be well-experienced with a wide variety of conventional programming languages, databases, and platforms. This experience will be essential to ensuring that:

- The web application will meet both immediate and evolving business needs;
- Provide rapid web applications development and deployment of a broad range of business and technologies;

We believe that enabling all the features and capabilities of CMS-based web applications along with a simple and powerful MDE approach for the development of these web applications can be achieved by the development of a rich domain specific language and a set of activities for the language usage. Furthermore we think that such a proposal can drive the choose of web applications to be more oriented

to CMS platforms and to a more mature development approach based on models, and applicable to a broader set of projects.

1.4 Thesis Statement

This dissertation's thesis is that **it is possible to simplify the task of complex CMS-based web applications development and improve overall quality by making a domain specific language that is coherent, portable, powerful and easy to use**.

In the context of this work, effort has been made towards to design and develop a domain specific language for CMS-based web applications, in order to study and explore the MDE approach and the model transformation properties and capabilities. Furthermore, this work proposes a development approach for using this domain specific language, with a set of well defined actors, processes, activities and artifacts.

To evaluate and validate the concepts and the approach proposed in this dissertation, one fictional case study is presented where the CMS modeling language is applied. The case study describes in detail how to develop a CMS-based web application using the domain specific language and demonstrate, by example, what are the benefits of this new CMS-based web applications development approach. To demonstrate the potential for platform independency, the case study is deployed in two distinct CMS platforms.

1.5 Outline of the Dissertation

This work comprises the following structure:

- **Introduction**. The first chapter establishes what the subject of concern of this dissertation is. Specifically, it presents the dissertation context and motivation, defines the problem that is subjacent to the work and presents this thesis statement.

- **Related Work**. Chapter 2 presents some key concepts of CMS, the benefits of using a CMS, how they are developed and some state-of-the-art CMS references. After, the focus shifts to the MDE approach for web applications, with an introduction of a small set of MDE initiaves for applications development. Finally, a discussion about the MDE initiaves is presented, along with a quantitative comparison between them, according to some relevant dimensions.

- **Domain Specific Language for CMS-based Web Applications**. Chapter 3 begins with a simple description of what is a DSL and its benefits, followed by an analysis of the CMS domain, which is the background to build the DSL. This chapter then presents a proposal of an UML Profile as a development approach that allows a rather different way of developing CMS-based web applications. The proposal includes a set of UML stereotypes for classes, associations and enumeration literals, along with a set of views for the development of CMS-based web applications; all of these are detailed in this chapter.

- **Transformation Processes an Technical Details**. Chapter 4 introduces the ProjectIT-Studio CASE tool, used in the context of the work. Then two key concepts in model transformations are presented, the PIM and PSM, furthermore, a description is given about the transformation processes required to transform the platform independent model to a platform specific model and therefore to specific platform artifacts. After defining the model transformation process, this chapter describes an important transformation characteristic, the transformation direction. Finally, the chapter ends with a detailed description of the transformation process developed for the language proposal with the ProjectIT-Studio tool.

- **Methodology for the Development of CMS-based Web Applications Using the CMS-ML Language**. In order to use the domain specific language in a systematically and consistently way, from project to project, chapter 5 presents the proposal for the development of CMS-based web applications, using the domain specific language presented in chapter 3. This chapter begins with the identification of the actors required for the development process and then proceeds with the definition of two development processes: the platform-level development and the project-level development. Each development process has different goals and activities that are defined in the respective sections.

- **Evaluation and Case Study**. To evaluate and validate the approach proposed in this dissertation, chapter 6 describes how the CMS Model Language was used in the development of one fictional web application, presenting the desired requirements, the model aligned with the requirements and a set of conclusions about the model design phase. Similarly, an evaluation is presented for the model transformation process and the supporting tool, the ProjectIT-Studio.

- **Conclusion**. Chapter 7 concludes this dissertation, summarizes the main contributions of this dissertation and discusses which new areas of research are opened in field of the CMS Modeling Language.

- **Bibliography**. Lists the bibliographical references that were found relevant either for research purposes or for the development of the proposal itself.

- **The *MyWebHome* Case Study**. Appendix A contains additional information about the *MyWebHome* case study, introduced in chapter 6.

Chapter 2

Related Work

The previous chapter stated that the ultimate goal of this work is to provide a new approach for the development of CMS-based web applications, which aspires to present more benefits than other approaches used nowadays, and presented some of the reasons for pursuing this goal. In order to hit such a high goal, an analysis to current CMS references and MDE initiaves is required to be able to define a domain specific language that is effective in its domain: the CMS-based web applications domain.

This chapter presents some concepts and products that are somehow related to this thesis work. The chapter is divided into three main sections: (1) a brief introduction to CMS, how to developed web applications with a CMS, a short analysis of some CMS references and an introduction to a related type of systems, the Enterprise Content Management systems (ECM); (2) the introduction of the paradigm of Model Driven Engineering (MDE) and a related standard, the MDA, and an analysis of some of MDE initiatives; and (3) a discussion about some MDE initiatives along with a quantitative comparison between them.

2.1 Content Management Systems - CMS

The main goal of this section is to introduce the CMS concept, describe how to develop a web application using a generic CMS and present some current CMS references.

Content Management Systems are software platforms, used to manage (create, update and delete) and publish content in a structured form. A special type of CMS are the Web Content Management Systems (WCMS) designed to simplify the management and publication of web content on web systems [7]. CMS allows users without technical knowledge to create and maintain content in a quickly and easily way, helping them in the system content organization and visual appearance.

One of the more useful feature of a CMS is its ability to separate the visual style of the content that is added to the system, this way the content and the visual style may change in time, independently of each other [11].

In a CMS approach, content is the first class entity, everything in the application focuses in content and content management.

2.1.1 Why Choose a CMS Approach?

Each CMS product has an unique combination of benefits, but all of them share the same main benefits of a generic CMS. Some of the advantages of a generic CMS are [7, 11]:

- The process of creation, edition and publish of content on a web site is much more simple and requires no technical knowledge;

- Time-to-publish is reduced, allowing the site content to be updated faster;
- An easier management of users and theirs operations;
- Consistent management of meta-data through content template structures. This delivers a significantly improved search process;
- Facilitates the management of sites with many authors and editors publishing substantial quantities of content;
- It is possible to extend the site capabilities through product specific extensions.

In addition to these advantages, there are specific advantages relative to the development and deployment of CMS-based web application, such as [11]:

- The browser-based interface and a site creation wizard makes the configuration and instantiation of the application very easy and quick;
- No technological knowledge is required for the development.

Because of these advantages and the future perspective for CMS-based web applications, this dissertation focuses on the development process of this type of web applications.

2.1.2 How to build a CMS-Based Application Nowadays

As presented in the previous section, a CMS approach is definitely a good approach for web applications development. This section briefly presents how a CMS-based application is developed and maintained, using the most common approach, a web-based formulary.

How a CMS-based web application is developed nowadays? Like any web application, the initial planning stages are the most crucial to ensure a successful design, content, usability, and much more. When the web application objectives are defined, the next step is to choose and install a CMS product. Independently of the specific CMS product chosen, the development of a web application using a specific CMS is almost standardized and are very similar for all products [39]. The majority of the web CMS uses online editing rather than push-publishing, which means it allows users to create and manage web pages and their content by using a web browser [9], this way the development of a CMS-based web application is done by only using a web browser.

The maintenance and creation of elements, such as web pages and users, on a CMS-based web application is done through an administrative web page. In order to create any element a set of information must be provided, such as the title, description, etc. For example if we want to create content in a web page, the user must provide a name and text for the content. Some elements must be related to others, such as the relation between users and roles; this relation is usually explicitly created in the formulary.

Any out-of-the-box CMS product comes with a great set of features, but sometimes they are not enough to satisfy the requirements or a very specific feature is required for a web application project, so most CMS platforms provide extension mechanisms in the form of components or modules. This way, a simple CMS platform installation can be trivially extended with the inclusion of new features in the form of modules.

The simplicity and familiarity of a browser-based web application development and maintenance makes this approach very easy and requires no technical knowledge, although it can become a complex task when the application complexity increases and the web application usually is not completely specified in the beginning of the project.

2.1.3 CMS References

The previous section defined what a CMS is, some of its advantages and described the common development method of CMS-based web applications. In this section some CMS products are analyzed to understand how CMS platforms works in practice and the way they differ from other products, with the goal to compare distinct platforms. The CMS platforms initiatives chosen were: Drupal [15], Joomla [26] and WebComfort [83].

Although there are many other CMS products currently available, this thesis presents this set because their features and user interfaces constitutes a good representation of the current state of the art of web CMS platforms.

In the last years many CMS platform products (Joomla, Drupal, WebComfort , Typo3 [79], Ez Publish [17], JBoss [24], Plone [47], and much more) have been developed. These products try to provide all the features of a basic CMS, plus some extra features, but all of them have the same goal and advantages of a generic CMS, presented in the previous section.

It's not in the scope of this work to make an exhaustive analysis of CMS, the thesis of João Carmo [11] has a good comparison between some CMS products and focus especially on the WebComfort, which is the theme of his dissertation.

A popular and updated source of CMS comparison is the CMS Matrix [73]. The CMS Matrix tries to provide an extensive comparison between CMS products through a great set of state of the art CMS features.

Drupal

Drupal is a CMS system with blog and wiki-like traits, the ability to run many sites and manage many users with variable roles and permissions [1] (chapter 1). Functionality, scalability, and customizability at every level has been prized over usability and aesthetics [32]. Its extreme flexibility and scalability requires a steep learning curve and a lot of time to realize the benefits. For that reason, initially Drupal can seem too amorphous and lacking in definition.

It's build on relevant standards, open-source technologies (PHP [46] and MySQL [71]) and aims to enhance the collaboration of individuals and groups by collectively produce, discuss, and share information and ideas [32]. With a central interest in and focus on communities and collaboration, Drupal's flexibility allows the collaborative production of online information systems and communities [1].

Drupal principles focus on [1]: modular and extensible; quality coding; standards-based; low resource demands; open source; ease of use; and collaboration.

Joomla

Joomla is a CMS developed by Mambo. It is written in PHP and runs with an Apache web server or IIS and MySQL database system.

Joomla's a very popular CMS, the primary reason it is so popular is the user interface aesthetics that the application offers to even the most novice users. Also, the vast array of extensions for Joomla makes it possible to deploy a system that can do almost anything, from chat rooms, to online auctions, to classified ads, to inventory management [32].

The installation and configuration of this product is almost done by "next" button clicks. The application development is likely very easy, simple and intuitive for basic features and basic management.

WebComfort

WebComfort is a R&D project aimed to the study, development and evaluation of CMS platform and its potential. It's promoted by SIQuant [68] and developed with Microsoft ASP.NET 2.0 technology and Microsoft SQL Server [10].

WebComfort has the capability to extend itself, this is crucial nowadays, and it's done through the addiction of modules. One particularity of this CMS is the workflow capability [53]. It is possible to define workflow and associate it with a web page or a component, and give them a sequence of related activities, thus making those component more powerful and activity oriented [53]. The authorization and security policies are also very well defined according to a flexible role-based mechanism [10]. In fact the access policies of WebComfort are very well designed, it's possible to define a role and its permissions to a web page or web component. The complexity and richness of roles and access mechanisms of WebComfort are subject for a whole chapter alone, so we will not discuss this theme in this work.

The software architecture of WebComfort is based in the classical three layer architecture [10]: data, logical and presentation. The presentation layer is based on ASP.NET and defines the system visual component. The logical layer defines a set of components and elements responsible for the business logic and rules [53]. The last layer, the data layer, is responsible for the data access operations [11].

2.1.4 Enterprise Content Management - ECM

The previous sections introduced CMS as a good approach to effectively manage content and support web applications. This section briefly presents another approach that's very popular in the management of content, in a more enterprise and document oriented way.

In many organizations, the content must be managed so that it is used to achieve business goals. Central to this strategy are the tools and technologies of the Enterprise Content Management (ECM), which manage the complete life-cycle of content, from birth to death [29].

ECM are advanced systems, may be thought as an extension to the CMS, that offers all the features of a CMS plus has the capability of document management related to organizational processes [29]. This way, ECM allows the management of organizations unstructured information [39]. The technologies used by these systems allow capturing, managing, storing, delivering, and preserving information to support business processes [39].

Typically, unstructured content enters organization IT infrastructure from a variety of sources. Regardless of how a piece of content enters, it has a life-cycle. ECM technology allows the organization to follow a document through its life-cycle [29]. Unstructured Data can be of various types and forms, physical (like paper) or electronic (like email and spreadsheet).

The installation, configuration, development and maintenance of these systems are much more complex than in the CMS described above, and **is not part of this dissertation scope**.

2.2 Model-Driven Engineering for Web Applications

This section presents the Model-Driven Architecture (MDA), which is the standard behind MDE and the advantages of adopting a MDE-based approach, followed by a description of MDE for general and web applications. Then, some MDE initiatives related to this work are presented, namely: Acceleo [2], WebML [84], UWE [80] and XIS [62].

As mentioned before, the focus of this dissertation is to develop CMS-based web applications in model-driven way. Model-Driven Engineering (MDE) is based on the systematic use of models as primary engineering entities during the solution specification [61]. MDE technologies that combine domain specific modeling languages and transformation engines [56], are a good approach to address platform complexity

and the inability of third-generation languages to alleviate this complexity and express domain concepts effectively.

The benefits afforded by MDE are productivity, quality improvements and platform abstraction. These benefits come from various sources [55, 61, 56]:

- Design models are easier and faster to produce and test;
- Some development tasks are automated;
- Design effort is focused on applications, not on platform details;
- Reuse of designs and tests between platforms or releases;
- Standardized common notations avoid retraining of engineers;
- Sometimes leads to a soft learning curve.

MDA [45] was proposed by the OMG [44] and it defines an approach to information systems specification that separates the specification of the system functionality from the specification of the implementation of functionalities on a specific technological platform and provides a set of guidelines for structuring specifications expressed as models [56].

MDA proposes two models, the Platform Independent Model (PIM) and the Platform Specific Model (PSM), to map a real-world problem to a computer system [55]. The PIM reflects the conceptual model of the target system. The PSM describes a model that is specific to the target platform, later, the PIM and PSM concepts will be described in more detail.

Another characteristic that's very popular in MDA is that it enables different applications to be integrated by explicitly relating their models; this facilitates integration and interoperability and supports system evolution (deployment choices) as platform technologies change [76].

As stated above, MDA is a powerful approach and provides an architecture that assures [8, 16, 6]:

- **Portability**: Increase application reuse and reduce the cost and complexity of application development and management.
- **Cross-platform Interoperability**: Use rigorous methods to guarantee that standards based on multiple implementation technologies all implement identical business functions.
- **Platform Independence**: Reduce the time, cost and complexity with re-targeting applications for different platforms.
- **Productivity**: Allows developers, designers and system administrators to use a language and concepts they are comfortable with, while allowing seamless communication and integration across the teams.

In the past years many methodological approaches (OOHDM [74], WebML [84], UWE [80], XIS2 [62], Acceleo [2], and much more) have been proposed for the model-driven development of software systems. These approaches try to provide support for systems development, by means of abstract mechanisms that make easy the conceptualization and development of these systems.

The following sub-sections, presents an analyses of some MDE for general or web applications approaches, in order to clarify the concepts of each initiative and the way each one differs from similar initiatives.

2.2.1 Acceleo

Acceleo is used for generating source code for heterogeneous technical platforms. It complies with the MDA approach in the way that it drives development towards models to increase the software development productivity. Acceleo is used to decrease realization time effort of software projects and to improve quality. It allows the generation of files using UML, MOF, and Eclipse Modeling Framework (EMF) modules [27].

Acceleo is based on the latest research advances and best practices and offers many advantages, such as: high customizability, interoperability, good learning curve and traceability management [6].

Modules are Acceleo plug-ins for code generation and each one is specific to some technology. There are a set of ready-to-use modules available for JEE, C#, Java, PHP and Python, but it's possible to add other modules [28]. The modules mechanism allows this language to generate applications code for, theorically, any platform, even for new technologies.

Figure 2.1 shows a simplistic macro view of the Acceleo: models, modules, acceleo engine and the process result - the code.

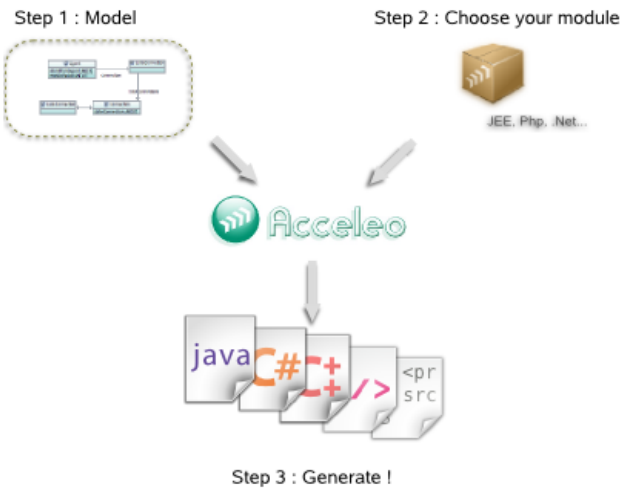


Figure 2.1: Acceleo overview diagram

2.2.2 UML-Based Web Engineering - UWE

UML-based Web Engineering (UWE) is a software engineering approach for development of applications in the web domain, based on OMG standards, such as UML, MDA, OCL, XMI and more, that focuses on models and model transformation [33]. The UWE notation is tailored for an intuitive modeling of web applications [34]. It comprises a development process and tool support for semi-automatic construction of Web applications. UWE models make early validation possible because a well designed model allows the requirements validation before the system development. One of the distinguishing features of UWE is its compliance with standards, because modeling with UWE is based on UML and on UML extension - the so-called UML profile [33]. The UWE profile is defined as an extension of the UML meta-model. Meta modeling and conformance with standards enable the use of existing tools or the development of plug-ins for tools already in use. UWE comprises [35]:

- A modeling language for the graphical representation of models of web applications;
- A method (technique) supporting semi-automatic generation;
- A meta-model for UWE modeling elements;

- A process supporting the development life-cycle of web applications;

UWE focuses on systematization, automatic generation and its main characteristic is the use of UML for all models [35]:

- "pure" UML whenever possible;
- For the web specific features, such as nodes and links of the hypertext structure, the UWE profile includes stereotypes, tagged values and constraints defined for the modeling elements.

UWE is supported by some case-tools, like ArgoUML [4] (ArgoUWE [5]) and MagicDraw [37] (MagicUWE [38]).

2.2.3 Web Modeling Language - WebML

WebML addresses the high-level, platform-independent specification of data-intensive web applications and targets web sites that require such advanced features as the one-to-one personalization of content and the delivery of information on multiple devices, like PCs, PDAs, digital televisions, and WAP phones [42].

WebML enables designers to express the core features of a site at a high level, without committing to detailed architectural details [43]. WebML concepts are associated with an intuitive graphic representation, which can be easily supported by CASE tools and effectively communicated to the non-technical members of the development team (e.g., with the graphic designers and the content producers) [42]. WebML also supports an XML syntax, which instead can be inputted to software generators for automatically producing the implementation of a web application [42].

Figure 2.2 shows the WebML meta-model package.

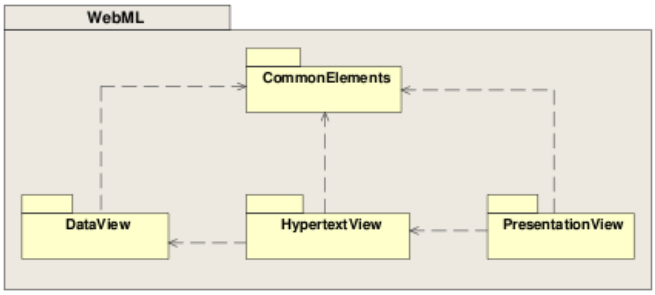


Figure 2.2: The WebML Meta-Model Package

The specification of a web application in WebML consists of four orthogonal perspectives [43, 42]:

- Structural Model: it expresses the data content of the site, in terms of the relevant entities and relationships.
- Hypertext Model: it describes one or more hypertext that can be published in the site. Each different hypertext defines a so-called site view. Site view descriptions in turn consist of two sub-models:
  - Composition Model: it specifies which pages compose the hypertext, and which content units make up a page.

- Navigation Model: it expresses how pages and content units are linked to form the hypertext. Links are either non-contextual, when they connect semantically independent pages, or contextual, when the content of the destination unit of the link depends on the content of the source unit.

- Presentation Model: it expresses the layout and graphic appearance of pages, independently of the output device and of the rendering language, by means of an abstract XML syntax. Presentation specifications are either page-specific or generic.

- Personalization Model: users and user groups are explicitly modeled in the structure schema in the form of predefined entities called User and Group. The features of these entities can be used for storing group-specific or individual content.

2.2.4 XIS2

XIS2 is a R&D project which main goal is the analysis, development and evaluation mechanisms to produce information systems in a more abstract, high-level, efficient and productive way than it is done currently [59]. XIS2 project is mainly based on three emergent best practices, namely: it is based on high-level models or specifications; it is architecture-centric; and it is based on generative programming techniques [60]. XIS2 evolved from XIS and is supported by a case-tool named ProjectIT-Studio.

The XIS2 approach is based on models, specified in UML and focuses on software architectures and development process. This approach has some crucial processes [62]:

1. The UML profile definition (XIS2/UML), with the system elements definition, their relationships and their meta-element association. This profile, along with the functional requirements, allows the system to be modeled, this model is UML-based and is transformed to XMI.

2. The definition of the XMI-to-XIS2 transformation template. This template specifies a set of rules for the transformation of XMI elements to XIS2/XML elements.

3. The definition of software architecture templates for the transformation of XIS2/XML elements to software components.

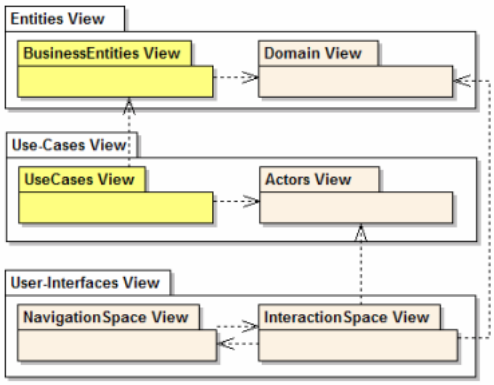


Figure 2.3: XIS Views Organization (extracted from [62])

The XIS2 specifies an application through three main views [59, 62], illustrated in figure 2.3:

- Entities view: basically it's a domain model with a logical representation of "high-level" entities that are defined on top of the domain entities.

- Use Cases View: it's a common use case model and may contain an extension to describe the use cases relation with business entities.
- User Interfaces View: is used to define the interaction spaces of a system and the navigation flow between them.
  - NavigationSpace View: describes the navigation flow between the identified interaction spaces.
  - InteractionSpace View: describes the contents and the overall organization of the different interaction spaces.

2.3 Discussion

This section presents a brief analysis of the MDE approaches mentioned in the previous section. This analysis pretends to give a notion of some factors that are relevant to this type of approaches, such as: the use of platform independent models, the visual aspect of the model language and expressiveness, the destiny platforms and the separation of concerns.

The analysis presented in this section was based in scientific articles [61, 51, 41, 28, 16] about each approach and the usage of each one in simple case studies. The analysis covers eight dimensions:

- **Platform Independent Models** - if the initiative proposes a set of concepts that are sufficient and independent of the underlying platforms.
- **Domain Models** - the ability to model the entities that are relevant to the problem domain in a traditional way, by using classes. The relationships between entities are modeled using simple associations, aggregations and inheritance.
- **Navigation Models** - to describe the navigation flow between the model spaces. This view is useful to support navigability management.
- **Visually Clean** - if the concepts are structured in a form that is not confused and easily comprehended.
- **Separation-Of-Concerns** - if the structure of the concepts are logically organized, the views shows only concepts that are someway related with the view goals.
- **Usability** - if the developers can use the language in an efficienlty way.
- **Effectiveness** - if the model-to-code transformation is effective and produces the right and all the required system artifacts.
- **Specific Platforms** - the family of platforms supported by the approach, for example: generates code for generic applications or only for web applications; and the effectiveness of the model-to-code transformation.

The table 2.1 presents a quantitative comparison between the analyzed approaches, the ranking goes from 1 to 5, being 5 the best ranking.

Each MDE initiative analyzed in this work varies in many aspects and concepts, but each one is effective and efficiently in model design, with a high degree of abstraction and easy to use, as shown in table 2.1.

There are several aspects relative to model-based applications development; some of them are visual aspects, relative to language characteristics, supported models, model views, application organization,

	Acceleo	UWE	WebML	XIS
Platform Independent Models	4	3	4	4
Domain Models	5	5	5	5
Navigation Models	3	4	5	4
Visually Clean	4	3	4	4
Separation-Of-Concerns	3	4	4	4
Usability	3	3	4	4
Effectiveness	5	5	5	5
Specific Platforms	Generic Platforms	Web Platforms	Web Platforms	Generic Platforms

Table 2.1: Comparison between MDE initiatives

and more. Anderson discussed several aspects relative to model-based design [3], namely: models must be visually clean; elements of the model must be sufficiently clear and distinct to avoid confusion or misinterpretations; it must be possible to visually highlight that one element is part of another element.

2.4 Summary

This chapter presented the CMS concept, some of its benefits, three state of the art CMS products and a brief notion of ECM systems. Following, some state of the art approaches of MDE for general and web applications where briefly mentioned along with an introduction of the MDE and MDA concepts. Finally a discussion and quantitative comparison between the referred approaches was presented.

Chapter 3

Domain Specific Language for CMS-based Web Applications

This chapter presents the concepts which provide the context for the development and usage of the domain specific language for CMS-based web applications, which is the subject of this work. In order to define a DSL, this chapter begins by explaining what a DSL is and why one should adopt a domain specific language approach. Then a brief definition of the domain for the language is given, followed by the identification of a platform independent meta-model for CMS-based web applications, with a group of stereotypes extending UML2 class elements. Finally a set of views are identified and defined for an efficiently language usage and separation-of-concerns in the application development process.

The main goal of this chapter is to present an UML-based language for the development of CMS-based web applications, which follows a model driven development approach and provides a way to make CMS-based web applications definition portable to other platforms.

3.1 Domain Specific Languages: An Abstraction Layer Over a Domain

According to Deursen [81], a domain specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. As such, a DSL can be used to generate members of a family of systems in an application domain. A well-designed DSL is based on a thorough understanding of the underlying application domain, giving exactly the expressive power to generate required family members easily.

Potential advantages of DSLs include reduced time-to-market, reduced maintenance costs, and higher portability, reliability, optimizability, and testability [81, 82]. A prerequisite for the design of a DSL is a detailed analysis and structuring of the application domain.

DSLs have limited focus; they aren't like generic programming languages that don't have a specific domain focus. The first step when adopting a DSL approach is to identify an abstraction, then build a DSL to more easily represent the component or system. Thus a common trigger to using a DSL is when one has the awareness that it is possible to simplify the representation of a component or system.

DSLs have the potential to realize, among others, the following benefits [31, 81, 72]:

- *Improving Development Productivity*: the heart of the appeal of a DSL is that it provides means to more clearly communicate the intent of a part or the whole of a system.

- *Communicating with Domain Experts*: A hardest part of software projects, the most common source of project failure, is communicating with the customers and users. By providing a clear yet precise language to deal with domains, a DSL can help improve this communication.

- *Change in Execution Context*: a common driver for using a DSL is when we want to support in different underlying environments.

There are some problems associated with a DSL approach, which must be consider if this work will specify one for the CMS domain. Fundamentally, the reason why one shouldn't use a DSL is because there aren't any benefits in the situation or the cost of building one DSL isn't worthy. Even when DSLs are applicable they do come with problems that are specific to each case.

A common though to remember when discussing these problems is the close relationship between a DSL and the underlying component that it's representing. Many of the costs and benefits of DSLs get confused with the costs and benefits of that component or system. Most of the time a DSL is nothing more than an alternative front-end to a framework.

Many issues exist when adopting a DSL approach [31, 77], but here are stated some of the most relevant to this work:

- *Cost of Building*: a DSL always takes some time and effort to build, essential because of the domain analysis.

- *Language Resistance*: languages are hard to learn and sometimes only bring more complexity to the underlying component. But DSLs tend to be limited and simple which makes them easier to learn, this is reinforced by their closeness to the domain.

- *Hard to Design*: a common concern is that language design is hard and therefore is beyond the skill level of most project teams. As with the language resistance problem, again we need to remember that DSLs are limited in expressiveness, thus the effort to design and understand them is much less than for a general purpose language.

Despise the disadvantages of a DSL, the proposal presented in this work adopts a DSL approach because the benefits have more weight than the disadvantages, mainly, the *Improving Development Productivity* is a highly desired property in a language for the model-driven development of CMS-based web applications. Two disadvantages lies with the language development and not with the usage, so they are not a drawback in our proposal; a third disadvantage is related with the language learning curve, which can be exceeded with a simple and well documented domain specific language.

3.2 CMS-based Web Applications Domain

To define a domain specific language, it's crucial to understand and limit the target domain. This section introduces the domain of the proposal, the CMS-based web applications domain, and its scope in order to define a domain specific language capable to express any application in it and support a bare minimum of features needed for this domain.

The domain analysis is like a systems analysis, except that the goal is to support the development of families of related systems, not just one-of-a-kind production [72]. Several domain analysis methodologies exists, of which ODM (Organization Domain Modeling [67]), FODA (Feature-Oriented Domain Analysis [30]), and DSSA (Domain-Specific Software Architectures [72]) are best known. The most important result of domain analysis is a feature model [13]. A feature model covers the commonalities and variabilities of software family members, as well as the dependencies between the variable features. The feature model

documents feature rationales, stakeholders, constraints (for example features may exclude each other), binding sites, and priorities [30].

Figure 3.1 shows the feature model for CMS-based web applications domain. This model only captures the essential generic elements of a generic CMS and some of them may not exist in a small set of CMS platforms. The feature model will allow the specification of the DSL to be concise and, ideally, to be done directly to the end users of the language who are likely to be domain specialists.

The domain analysis was based on the use of some CMS platforms [26, 15, 83] and through the analysis of scientific literature on the subject [7, 70, 25, 39] and specially the work of João Carmo[11].

In this project, the feature model of the CMS framework is essentially used to produce a DSL. More precisely, the feature model is used to express a generic CMS platform and help in the specification of an UML-based DSL. As already mentioned above, it is important to avoid a situation where a new DSL has to be defined for each new target CMS platform, so the goal is an "unified" DSL for CMS-based web applications.

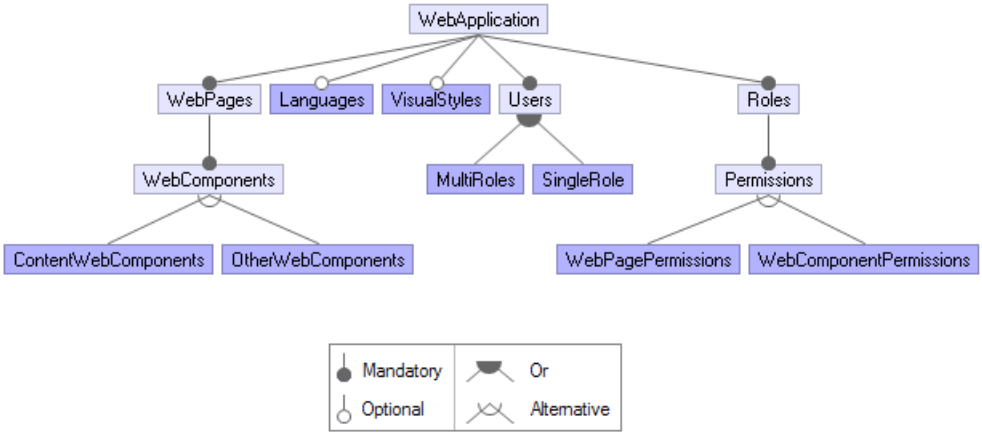


Figure 3.1: CMS-based Web Application Feature Model

In a very simplistic and generic fashion, a CMS-based web application is composed by a variable set of dynamic and/or static web pages. Dynamic web pages can be defined hierarchically, configured and presented in a controlled way by the CMS platform [70]. Each dynamic web page can have a set of modules, spatially arranged in the web page accordingly to some dynamic containers defined in a visual style by the designers. Additionally, it's possible to define and manage different visual themes, which together set out the layout aspects of the entire application, a specific page or just a module/content [11].

Also, CMS platforms allows the management of users together with the management of roles and permissions to provide a generic and flexible mechanism for access control that adapts easily to different requirements and organizational business [11]. Some CMS platforms allows a more focused access management like defining local roles that exist only in the context of a single web page, this is not shown in the feature model because the majority of the CMS platforms do not support this type of access management [11]. Although, this domain analysis cannot ignore this feature because it represent a great advantage for web applications and increases the roles management flexibility.

Besides the feature model, an architectural analysis is required. This work domain is not very architecture-restricted, because each CMS platform can adopt a specific platform, entire distinct from another CMS platform. This way, CMS platforms can be represented with a generic architecture illustrated in figure 3.2.

The generic CMS platform architecture is centered on a "CMS Core" and some type of repository, the most common repository are relational databases and files. Many CMS platforms also provide an

API that offers a set of operations over the platform, like create a new user or a new web page, edit permissions of role on a web page, and all kind of possible operations over the platform. Finally, for a successful CMS platform, extension mechanisms must exist for visual styles, languages (if the CMS platform supports multi-language), web components or modules, and other extensions [7].

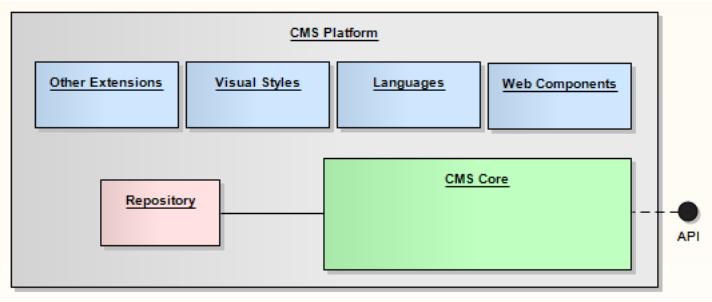


Figure 3.2: Generic CMS Platform Architecture

Usually, the best way, or the only way, to manipulate a CMS platform (for example create web pages or users) is through the API or Repository, shown in figure 3.2. How to access the CMS platform is completely at the discretion of developers and the platform capabilities.

Summarizing, the main concepts of the target domain are: web pages, web components, roles, users, visual styles and possible multi-languages. The goal is to define a language capable to express these elements and the relations between them while not arresting them to any specific platform. Clearly, the technological platform and the repository type of each CMS platform can take several forms and may be at different levels of complexity. This is the biggest challenge in the development of the DSL, to warranty that it's completely transparent from the underlying CMS architecture and technological platform.

For now, the scope of the DSL will be limited to these elements, **not being possible to model new web components, visual styles and languages**. This means that, the modeling of web components, visual styles and languages, resumes to their identification and configuration, presuming that all these elements are already developed and deployed in the target CMS platform.

For example: if a web application will use the visual style VS1, then it's required to map the VS1 (through a platform specific mapping mechanism) to some visual style that already exists in the target platform; or if a web component of type WCT1 will be added to a container, then there is the need to map the web component type to a platform specific web component type. This language restriction will be explained in more detail in Chapter 4 and in the case study of Chapter 6.

3.3 CMS-ML: CMS Modeling Language

The previous section presented an analysis of the CMS-based web applications domain in which the proposal of this work is focused and defines a DSL for the development in this domain. Although the domain and its scope are small and simple, the platform independency is a high goal to achieve, especially in a domain where several technological possibilities exist and new ones are always emerging. This section presents a proposal of a domain specific language for CMS-based web applications.

Stated in the previous sections were some benefits for using a DSL and some characteristics that are desired for the DSL defined in this dissertation. Synthesizing the desired benefits, the set of major goals for the CMS Modeling Language is illustrated in figure 3.3. How the language will achieve these goals will be described throughout this section and in Chapter 5, when the concepts and techniques that enables the goals are presented.

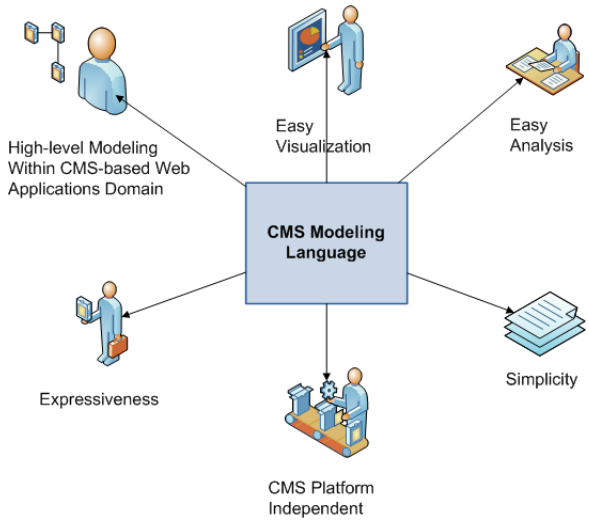


Figure 3.3: CMS Modeling Language Goals

The DSL, called by CMS Modeling Language (CMS-ML), defined in this work has the form of an UML profile. An UML profile provides a generic extension mechanism for customizing UML models for particular domains and platforms. Profiles are defined using stereotypes, tagged values, and constraints that are applied to specific model elements and associations [42].

Often the essence of the domain boundary doesn't lie in the characteristics of the specific language, but in the intent of either the developers or users of this language. If a developer uses the CMS-ML for the development of CMS-based web applications then it stays a DSL, but if somehow he uses the CMS-ML in a general purpose manner or in another domain, then it's no longer a DSL because the language domain boundaries where "transposed".

Calling some concepts a DSL is mostly about helping to describe them to others and knowing whether the techniques in this dissertation are likely to be relevant to build and using the CMS-ML DSL.

The CMS-ML is oriented towards the development of interactive software systems, and its main goal is to allow the modeling of the various aspects of CMS-based web applications, in a way that should be as simple and efficient as possible.

For presentation reasons, the meta-model is presented on two viewpoint, illustrated in figures 3.4 and 3.5: (1) a viewpoint showing all the stereotypes that extends from UML class elements and UML association elements; and (2) a viewpoint showing the enumerations and the stereotypes that extends from UML enumeration literals that are required to set tagged values of some UML class and association elements.

Notice that, all the enumerations literals shown in figure 3.5 are only examples and the reason they are there is to show the stereotype applications on the enumeration literals.

The meta-model identifies the class elements, associations and enumeration literals which when combined provide a web application specification that is simple and enough to produce artifacts in a CMS platform, independently of the platform details. The reader shall see in Chapters 5 and 6 that it's very possible to produce rich platform independent models of CMS-based web applications with the meta-model defined here and generate the corresponding web application in different CMS platforms.

The CMS-ML language defined in the form of a meta-model pretends to be platform independent, simple, powerful, complete and flexible enough to support various CMS technological platforms and to provide the developers with a complete tool to produce CMS-based web applications in a way that is more oriented to the paradigm of Model Driven Engineering/Development (MDE/MDD).

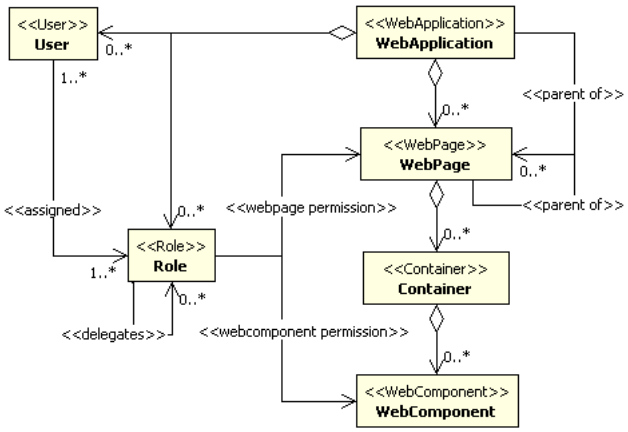


Figure 3.4: CMS-ML Platform Independent Meta-Model: Elements and Associations

The class elements of the meta-model have a set of attributes or tagged values required to completely define each one. For example: to instantiate a *WebComponent* to manage content, the analyst must: (1) create a *WebComponent* element; (2) set the element *type* tagged value (that has the type *WebComponentType*) with the *Content* literal defined in the enumeration *WebComponentTypes*; (3) finally, define the element *content* tagged value with the content of the component.

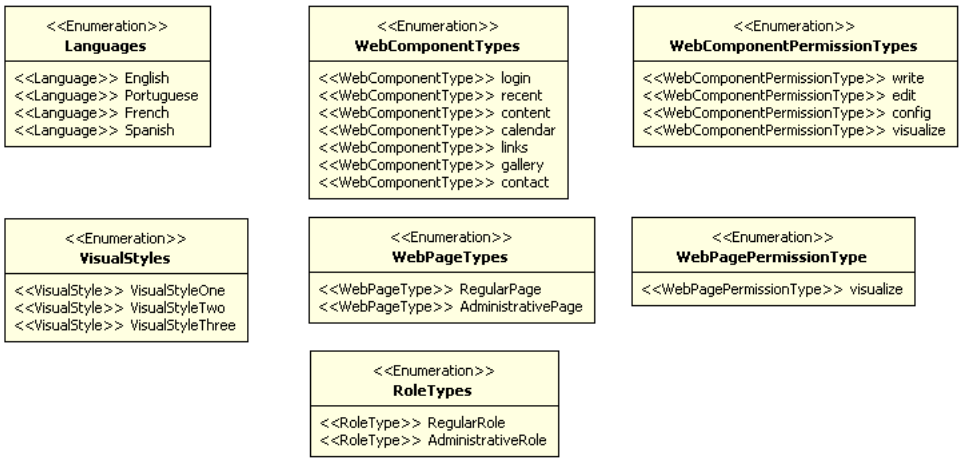


Figure 3.5: CMS-ML Platform Independent Meta-Model: Enumerations and Literals

The profile file, defined in this chapter and used in the case study was specifically developed for the CASE tool adopted in this work, the *ProjectIT-Studio* [23]. Although, the CMS-ML DSL is independent of the CASE tool and it's possible to define a profile file for other CASE tools, such as the Enterprise Architect [69] or the Rational Rose [22].

The process of defining a profile in ProjectIT-Studio UMLModeler involves only the creation of a CMS-ML Profile within a model with all the stereotypes defined in this chapter. In order to improve the reusability of the profile, it is usually a good idea to define a CMS-ML Profile in a separate UML model file, and at a later time apply the CMS-ML Profile to another UML model file.

Because the CMS-ML language is simple and doesn't have many elements, the user can create only one diagram within the CMS-ML Profile with all the required stereotypes. Note that a Profile is only a container of Stereotypes and other Profile Packages, and is not associated with any particular semantics.

The CMS-ML class elements, associations and enumeration literals are defined in the following sub-

sections, their usage is explained in Chapter 5 and demonstrated in the case study of Chapter 6.

3.3.1 CMS-ML Enumeration Literals

As mention before, the CMS-ML language includes a set of enumerations, each one composed by a set of literals that are used to set CMS-ML class elements and associations tagged values of a specific type. This sub-section describes each enumeration literal in detail, why they exist in the form of enumeration literals and why not other UML2 element and where they can be used.

The CMS-ML proposes seven enumerations: *Languages*, *VisualStyles*, *WebPagesTypes*, *WebComponentTypes*, *RoleTypes*, *WebPagePermissionTypes* and *WebComponentPermissionTypes*. Each enumeration groups enumeration literals, all of them with the same stereotype, defined in the CMS-ML metamodel. For example, the *RoleTypes* enumeration will only have enumeration literals with the applied stereotype *RoleType*.

The CMS concepts that are defined as CMS-ML literals are an extension of the UML2 element *EnumerationLiteral*, mainly because no additional detail is required to define them. Using enumeration literals bring an important benefit: the developer can specify a tagged value of a class element or association in the most trivial possible way, only by setting a tagged value with the correct value (already defined in an enumeration).

Another benefit of using enumeration literals is that no additional association or view is required to define new elements. Adding a new enumeration literal element resumes to add a new literal with the required stereotype to the respective enumeration.

Most CMS platforms have support for multi visual styles and languages, so the developer needs to express what language and visual style will be applied to the web application or web page, if the platforms allow that. This is easy done with CMS-ML, first the developer needs to build a list of enumeration literals, with the stereotype *Language* or *VisualStyle*, in the enumerations *Languages* or *VisualStyles*, respectively. Then every time there is the need to set a language or visual style for a web page or web application, the developer only needs to set the corresponding element tagged value (*language* or *visualStyle* tagged value) with an enumeration literal with the right stereotype; sometimes the development tool, like ProjectIT-Studio [65, 66], provides a straightforward way to set these tagged values.

The enumeration literals of CMS-ML offers an easy way to express simple concepts that do not require much detail and are used to recognize special characteristics of an element, which may be required to correctly generate the correspondent artifacts in the CMS platforms. The CMS-ML enumerations and enumeration literals are used in the case studiy presented in Chapter 6, where the reader can find more practical details about the enumerations, literals, associations and class elements.

In a specific CMS platform an element that has the form of a CMS-ML literal, such as *Language*, may require more detail, this can be easily done in the template for the platform, by adding specific detail in the template and not in the platform independent model; we'll see this in more detail in the next chapters.

The following sub-sections details each CMS-ML enumeration literals, which extends from the *EnumerationLiteral* UML MetaClass.

Language

A *Language* represents the idiom to be applied to the web application. A web application can have a set of possible languages, but must have one and only one language associated at a time. These elements are **not modeled with the CMS-ML**, it's only possible to make a reference to a specific language, which must be posteriorly mapped to a platform artifact. Languages are expressed in the form of enumeration literals with the applied stereotype *Language* which belongs to the enumeration *Languages*.

A *Language* literal can be set to two tagged values: *WebApplication.language* and *WebPage.language*.

VisualStyle

The *VisualStyle* defines the visual specification to be applied to the web application. This specification includes the structure of the web pages, color schemes, images, etc. A web application can have several visual styles that may be applied, but only one can be applied at a time, in fact, one must always applied. These elements are **not modeled by the CMS-ML**, it's only possible to make a reference to a specific platform visual style. The visual styles are expressed in the form of enumeration literals with the applied stereotype *VisualStyle* which belongs to the enumeration *VisualStyles*.

A *VisualStyle* literal can be set to two tagged values: *WebApplication.visualStyle* and *WebPage.visualStyle*.

WebPageType

A web page type allows an easy way to classify web pages, for example: specifying administrative web page and front public web pages. The web page types are expressed in the form of enumeration literals with the applied stereotype *WebPageType* which belongs to the enumeration *WebPageTypes*.

A *WebPageType* literal can be set to the *WebPage.type* tagged value.

WebComponentType

Web components can go from calendars to complex shopping catalogs, passing through any type of utility and web feature. Web components can also manage CMS systems content. This way, web components can be of different types, these types are defined in the *WebComponentTypes* enumeration that contains enumeration literals with the applied stereotype *WebComponentType*.

A *WebComponentType* literal can be set to the *WebComponent.type* tagged value.

RoleType

The roles in CMS systems may be of different types, for example: administration roles or public roles. The role types are expressed in the form of enumeration literals with the applied stereotype *RoleType* which belongs to the enumeration *RoleTypes*.

A *RoleType* literal can be set to the *Role.type* tagged value.

WebPagePermissionType

A role can have a set of permissions over a web page, the type of these permissions are expressed in the form of an enumeration literals with stereotype *WebPagePermissionType*, defined in the *WebPagePermissionTypes*. Examples of permissions types are: visualize and configure.

WebComponentPermissionType

As the web pages, the web components can also have an access control. This way, a role can have a set of permissions over a web component, the type of these permissions are expressed in the form of a enumeration literals with stereotype *WebComponentPermissionType*, defined in the *WebComponentPermissionTypes*.

3.3.2 CMS-ML Class Elements

The previous sub-section defined each CMS-ML enumeration literal in detail. This sub-section defines each class element of the domain specific language to better understand their application in the development process and their relationship with the enumeration literals.

In the following sub-sections, a definition for each CMS-ML class element is provided. Some elements have an attribute named *configuration* where the developer can add additional attributes, in the form of key-value pair. The configuration string is described in more detail forward in this section.

WebApplication

The *WebApplication* element represents a CMS-based web application. The model by its own represent the web application, as it contains all the elements and associations, but this entity allows the specification of what language and visual style to use; the homepage and the required web application configuration parameters.

The *WebApplication* is a stereotype for UML class elements. It has a set of tagged values, namely: *VisualStyle visualStyle*, *Language language*, *WebPage homepage* (Rrequired) and *String configuration*.

WebPage

A web page is a logical set of containers that are distributed through the web page. These containers may have a set of web components that implement specific business logic. A web page typically represents a logical section of the site, with a well defined goal and a set of web components that are relevant to that goal. A web page must have one visual style associated at a time, this visual style can be distinct from the one defined in other web pages. In the absence of an association with a visual style, the web page uses the visual associated with the web application. The same applies to the web page language. A web page can also have one parent web page and many child web pages, forming a hierarchical web pages structure.

The *WebPage* is a stereotype for UML class elements. It has a set of tagged values, namely: *WebPageType type*, *VisualStyle visualStyle*, *Language language* and *String configuration*.

Container

A container is a logical space to group web components within the web page. The combination of containers allows the definition of layouts that gives a spatial specification for a web page, this way it's possible to have various layouts for different web pages.

The *Container* is a stereotype for UML class elements with the *int order* (required) tagged value.

WebComponent

A web component is deployed in a container that is part of a web page. A web component has associated business logic to manage a certain type of content. It is responsible for managing and presenting content information with a determined visual style. It's possible to add all types of features to the web application through the inclusion of modules.

When modeling a *WebComponent* element it's very important to keep in mind that we are not developing a new web component, instead we are instantiating and configuring an already developed web component that is deployed in the platform, these elements are **not modeled by the CMS-ML**.

The *WebComponent* is a stereotype for UML class elements. It has a set of tagged values, namely: *WebComponentType type* (required), *String content* and *String configuration*.

Role

A role has a set of permissions. It may be associated with users in order to give them permissions to some web site elements; can be applied to a set of web pages and web components; may be restricted to a web page context (local roles); and may have a delegation relationship with other roles, this way, a role can delegate its permissions to another role.

The *Role* is a stereotype for UML class elements. It has a set of tagged values, namely: *WebPage webPage*, *RoleType type* and *String configuration*.

User

A user represents an identity in the web site. In order for an user to access some of the web site areas or to gain access to certain operations, he needs a role. A user must be assigned to at least one role.

The *User* is a stereotype for UML class elements. It has a set of tagged values, namely: *String name*, *String email* and *String configuration*.

Configuration String Attribute

The configuration string is a mechanism that allows the developer to add new attributes to the language elements. It has the goal to provide an extension to the elements attributes while not making the element platform specific.

For example, an album web component may require two inputs, the number horizontal and vertical images in a matrix. Because the web component element doesn't have tagged values to specify the dimensions of a matrix, these parameters can be specified in the configuration string in the form of a key-value pair:

```
numberHoriz=5;numberVerti=6
```

The configuration string tagged value is subsequently treated and processed by the templates for the platform.

3.3.3 CMS-ML Associations

The elements described in the previous sub-section must have relationships among each others in order to define a coherent model able to provide sufficient detail to generate the corresponding CMS-based web application, without sacrificing its expressiveness. CMS-ML has a set of associations' stereotypes that can be applied according to the definitions of the following sub-sections.

assigned

The *assigned* is a directed association and means that the user in the association source is assigned to a specific role in the association target. A user can have many *assigned* associations with roles. This way, the *assigned* is applied to UML directed associations between two UML class elements, the association source with a *User* class and the association target with a *Role* class.

parent of

The *parent of* is a directed association between web pages or between the web application and web page. It means that the association source, web page or web application, is hierarchal superior to the association target web pages.

The *parent of* is applied to UML directed associations between two UML class elements, the association source with a *WebApplication* or *WebPage* class and the association target with a *WebPage* class.

webpage permission

The *webpage permission* is a directed association and means that a Role has a specific permission over a web page. This association has always a defined tagged value that represents the access type, which is a literal, defined in section 3.3.1, from enumeration *WebPagePermissionTypes*.

The *webpage permission* is applied to UML directed associations between two UML class elements, the association source with a *Role* class and the association target with a *WebPage* class.

webcomponent permission

The *webcomponent permission* is a directed association and means that a Role has a specific permission over a web component. This association has always a defined tagged value that represents the access type, which is a literal, defined in section 3.3.1, from enumeration *WebComponentPermissionTypes*. A role can only have permissions over a web component if it has permissions over the web page that contains the web component; otherwise, the permissions over the web component are discarded. This association can originate some inconsistence if not used correctly; it's the responsibility of the developer to use it in accordance with the expected.

The *webcomponent permission* is applied to UML directed associations between two UML class elements, the association source with a *Role* class and the association target with a *WebComponent* class.

delegates

The *delegates* is a directed association between Roles. When a role delegates its permissions to another role, the second role inherits all the permission from the first role. This association can originate some inconsistence if not used correctly; it's the responsibility of the developer to use it in accordance with the expected.

3.3.4 CMS-ML Model Views for the Development of CMS-based Web Applications

The goal of this section is to describe a set of views to be used in application development in order to give several viewpoints of the application, hide the model complexity, allow the developers to focus in a small set of the application and structure the application elements to effectively use CMS-ML. The views defined in this section are detailed in Chapter 5 and applied in the case study presented in Chapter 6.

The purpose of views is to enable developers to understand very complex systems, and to organize the elements of the application. Some complex system specifications are so extensive that no single individual can fully comprehend all aspects of the specifications. Furthermore, when developing even the simplest application, the developer faces many challenges related to visualization and organization of the model. The concept of view, therefore, is to provide separate viewpoints into the specification of a given application. These viewpoints allow a separated specification of a particular set of aspects and/or elements of the application.

One great lack in the CMS-based web applications development approaches, such as the web formulary-based development, is that the stakeholders don't have the ability to see the web application according to specific viewpoints. The conceptual web application model illustrated in figure 3.6, shows the desired concepts around a web application and the stakeholders.

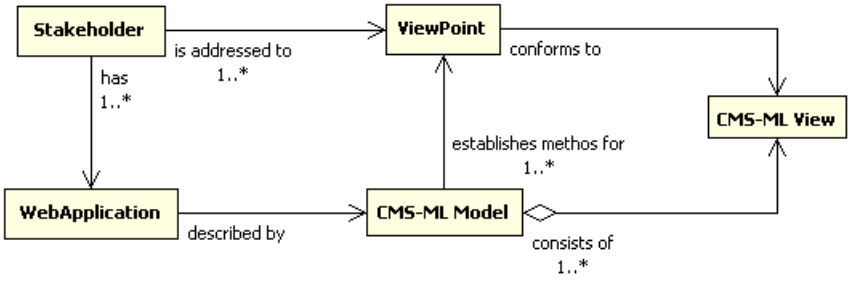


Figure 3.6: Conceptual WebApplication Model

By describing a web application through a CMS-ML model, with several well organized views, the stakeholders can obtain the viewpoints most suited for their needs. This is the fundamental reason that leads to the definition of the CMS-ML model views of this section.

The CMS-ML model views can be categorized into four main views: *WebApplication View*, *WebPages View*, *Roles View* and *Users View*.

Each one of the views represents how part of the web application can be modeled and presents a different perspective over CMS-ML class elements, associations and enumerations. This will enable the developers to understand where exactly the CMS-ML elements fit in and their applicability. These different views can be seen with a structured form in figure 3.7.

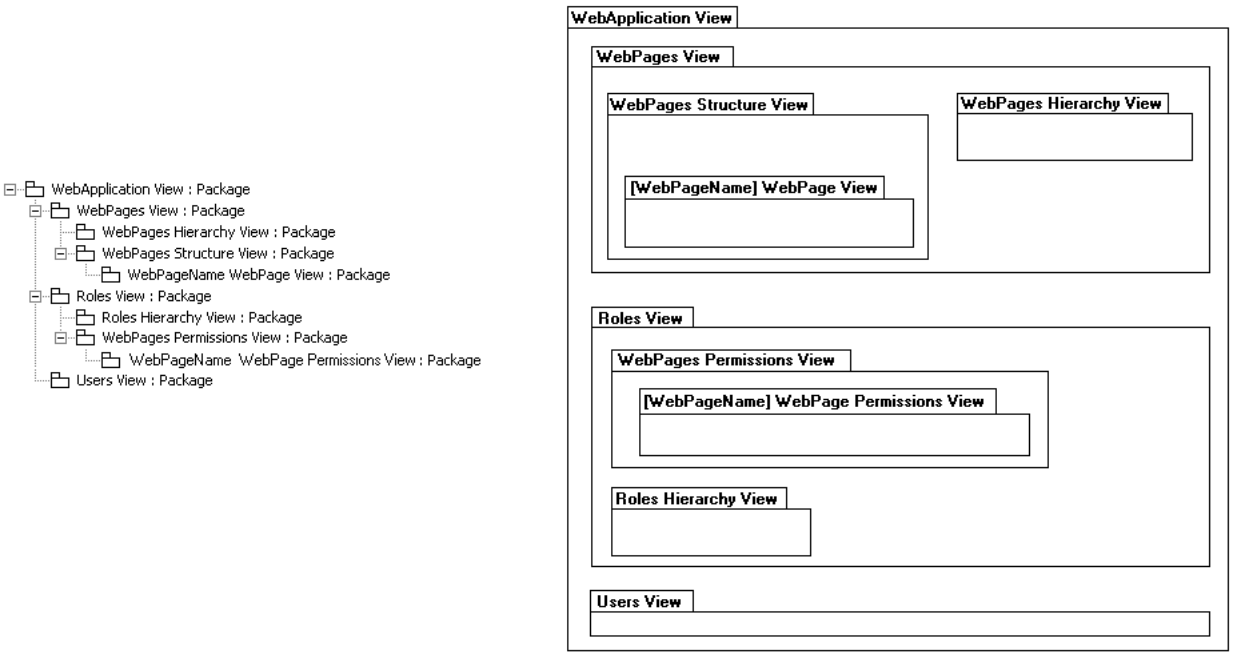


Figure 3.7: CMS-ML Model Views

Figure 3.8 gives an overview of the CMS-ML views and meta-model. It shows all the views overlapped with the CMS-ML class elements, enumerations and associations. From this detailed macro view the reader can understand in which view each CMS-ML meta-model element is defined and where to define each view. Note that the associations *parent of* and *delegates* must be defined in the *WebPages Hierarchy View* and *Roles Hierarchy View*, respectively.

The *[WebPageName] WebPage View* and the *[WebPageName] WebPage Permissions View* are views

specific to a *WebPage*, so there will be one of these views for each *WebPage*. Note that these two views can be automatically generated by the supporting tool, when a new *WebPage* element is created, section 7.2 will detailed this automated task in more detail.

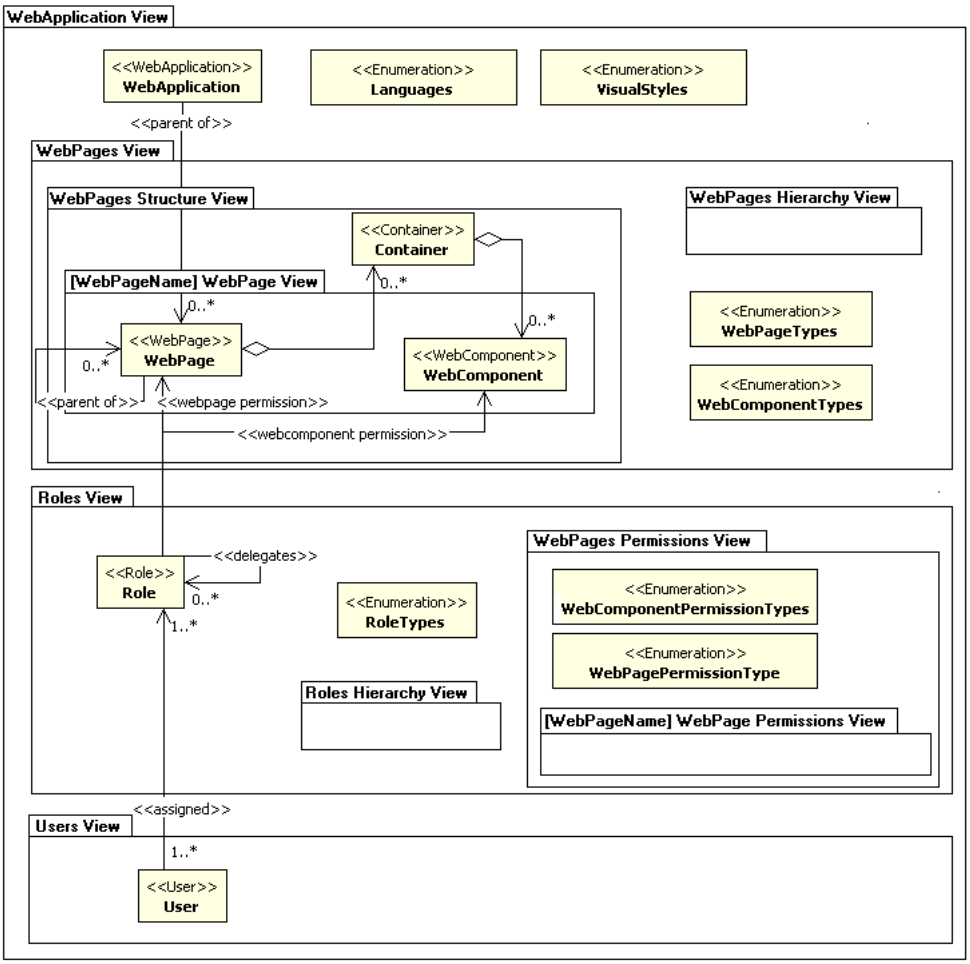


Figure 3.8: CMS-ML Views and Meta-Model

The goal of CMS-ML views is to give simple views that focus in simple concepts in order to give a clear understanding of the web application model parts.

The following sub-sections describes each CMS-Ml view in detail.

WebApplication View

The *WebApplication View* is the main model view, it contains three specific macro views: the *WebPages View*, the *Roles View* and the *Users View*; and contains two enumerations: the *Languages* and *VisualStyles* enumerations (section 3.3.1); along with a class element with the stereotype *WebApplication* (see section 3.3.2 for more detail about the stereotype).

Figure 3.9 shows a typical *WebApplication View*, the enumeration literals shown in the figure are only examples.

The *Languages* and the *VisualStyles* enumerations are defined in this view because they represent concepts at the web application level. These concepts are global to all the web application and are used by the *WebApplication* and the *WebPage* class elements.

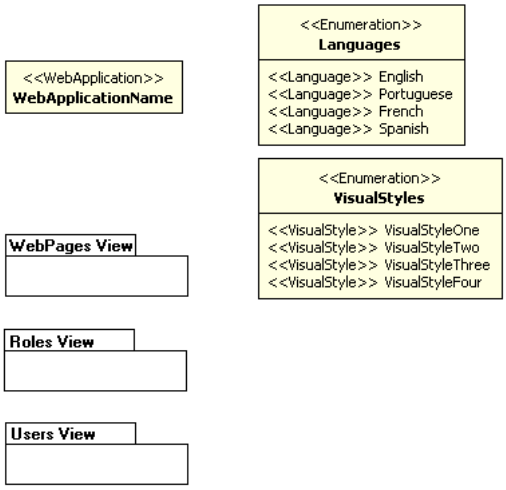


Figure 3.9: WebApplication View Example

WebPages View

The *WebPages View* contains two enumerations: the *WebPageTypes* and the *WebComponentTypes*; along with two sub views related to web pages: the *WebPages Structure View* and the *WebPages Hierarchy View*. Figure 3.10 shows the *WebPages View* with the two sub-views and the two enumerations, the enumeration literals of the figure are only examples.

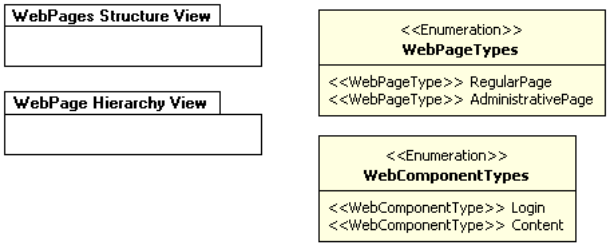


Figure 3.10: WebPages View Example

Like the two enumerations of the *WebApplication View*, the *WebPageTypes* and the *WebComponent-Types* enumerations are defined in this view because they are confined to the web page scope. The *WebPageTypes* contains the literals that represent the type of a web page, used to classify web pages. It's evident that these concepts are related to the web page concept so the *WebPageTypes* enumeration is defined in this view.

On the other hand, the *WebComponentTypes* enumeration is also related to the web page concept, because the web components are defined in the context of a web page, they are a structural part of a web page. The web components are defined in the *WebPage Structural Views*, which is a sub-view of the *WebPages View*, this is the reason to define the *WebComponentTypes* enumeration in the web pages macro view.

WebPages View::WebPages Structure View

The *WebPages Structure View* shows a high level view over the *WebPages* structural definition. As figure 3.11 shows, the *WebPages Structure View* aggregates a set of other views, a structural view for

each *WebPage*. This way, we have a view that focus only in a *WebPage* structure, with the *Containers* and its *WebComponents*.

This view also shows the *Containers* of the web application. The *Containers* are defined in this view mainly because they are part of the *WebPage* structure, so all the *Containers* are defined in the *WebPages Structure View* and then placed inside each *WebPage*, defined in the correspondent view.

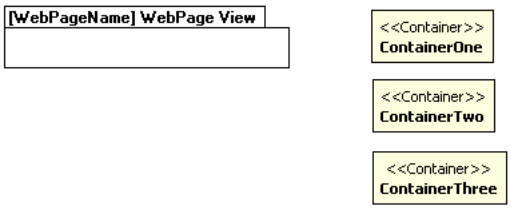


Figure 3.11: WebPages Structure View Example

Figure 3.11 shows a view named by *[WebPageName] WebPage View*, which represent a view for a specific *WebPage*. The *[WebPageName]* must be replaced by the name of the *WebPage* and a view of this type must be created for each *WebPage*.

Each *[WebPageName] WebPage View* will define a *WebPage* structure, with the *Containers* defined in the *WebPages Structure View*, and the *WebComponents* instantiated in the respective *WebPage*. The *WebComponents* used by the *WebPage* are all defined in the *WebPage* structural view.

The *[WebPageName] WebPage View* will be more detailed in Chapter 5 and Chapter 6.

WebPages View::WebPages Hierarchy View

The *WebPages Hierarchy View* shows the hierarchical relationships between *WebPages* elements, previously defined in the correspondent structure view.

A *WebPage* hierarchy relationship in CMS-ML is defined by a directed association with the stereotype *parent of* (see section 3.3.3). Thus, this view only contains class elements with the stereotypes *WebPage* or *WebApplication*, and associations with the stereotype *parent of*.

Section 3.3.3 defined that the *parent of* association is possible between elements with stereotype *WebPage*, the element at the end of the association will have a hierarchical level immediately below the element at the association source. The association source element is the direct parent of the element at the association target.

It's also possible to have a *parent of* association between *WebApplication* and *WebPage*, in this case, the association begin is always a *WebApplication* element and all the target elements are *WebPage*. These *WebPages* are at the top level of the web pages hierarchy.

Figure 3.12 shows an example of a simple web pages hierarchy view.

Roles View

The *Roles View* shows a high level view over the *Roles* definition. Figure 3.13 shows that the *Roles View* aggregates two other views: the *WebPages Permissions View* and the *Roles Hierarchy View*; also, in this view, the *RoleTypes* enumeration is defined with the *RoleType* literals. The *RoleTypes* contains the literals that represent the type of a role, used to classify roles. These concepts are related to the role concept so they are defined in this view.

This view also shows the roles defined in the system, i.e., all UML class elements with the *Role* stereotype.

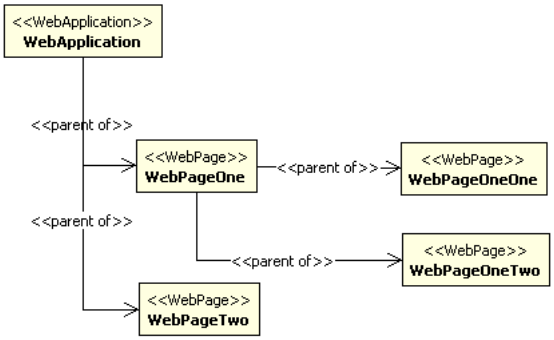


Figure 3.12: WebPages Hierarchy View Example

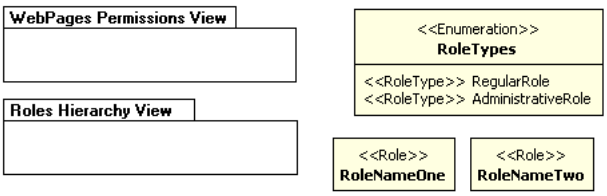


Figure 3.13: Roles View Example

Roles View::WebPages Permissions View

The *WebPages Permissions View* goal is to give the roles permissions over each web page defined in the *WebPages Structure View*. To do so, this view contains a view for each web page, named *[WebPageName] WebPage Permissions View* where the *[WebPageName]* is the name of the web page related to the view.

This view also contains two enumerations: *WebPagePermissionTypes* and the *WebComponentPermissionTypes*. The *WebPagePermissionTypes* and the *WebComponentPermissionTypes* contains a set of enumeration literals of type *WebPagePermissionType* and *WebComponentnPermissionType*, respectivelly.

Figure 3.14 shows an example of the *WebPages Permissions View*.

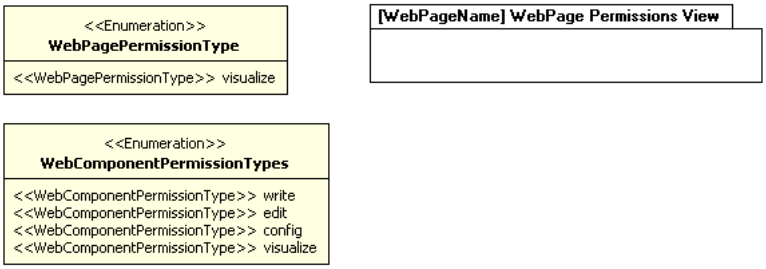


Figure 3.14: WebPages Permissions View Example

Each *[WebPageName] WebPage Permissions View* contains all the elements defined in the correspondent structure in the *[WebPageName] WebPage View*, i.e., the respective *WebPage* element and its *WebComponents*. Figure 3.15 shows an example of a *[WebPageName] WebPage Permissions View*. Note that the associations between *Roles*, *WebPages* and *WebComponents* have different stereotypes, depending to the association target element, and each association has the *type* tagged value set to a specific permission type (defined in the *WebPagePermissionTypes* or *WebComponentPermissionTypes*).

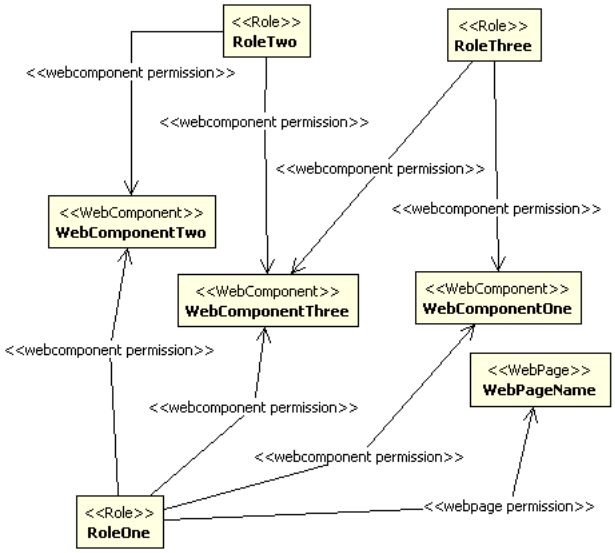


Figure 3.15: WebPageName WebPage Permissions View Example

The role permissions are exemplified in the case study of chapter 6.

Roles View::Roles Hierarchy View

Roles Hierarchy View gives a viewpoint over the delegation associations between roles. This delegation is specified using the CMS-ML *delegates* association (details in section 3.3.3) between *Role* class elements. Figure 3.16 shows an example of a roles hierarchy view.

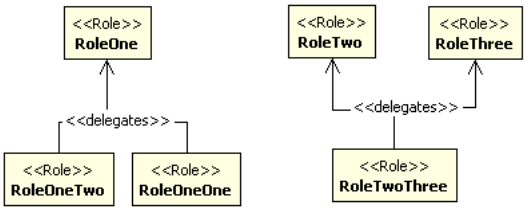


Figure 3.16: Roles Hierarchy View Example

Users View

The *Users View* shows the definition of all the *Users* of the web application and their assignment to *Roles*, through the *assigned* association defined in section 3.3.3. Figure 3.17 shows an example of *Users View*.

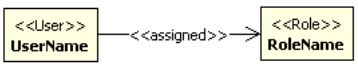


Figure 3.17: Users View Example

3.4 Summary

This chapter presented the definition of a language for the development of CMS-based web applications, which is part of the proposal that is the main contribution of this dissertation. This chapter started with an introduction of the concept DSL showing why it is a good approach for this work proposal. Then an analysis to the target domain was presented. Based in the domain analysis, a platform independent Meta-Model for CMS-based web applications were defined along with all the language elements: enumerations and enumeration literals, class elements and associations. Finally, a proposal for model structure was presented with a set of model views, in order to improve the web application development using the CMS-ML approach.

Chapter 4

Transformation Processes and Technical Details

In the previous chapter a domain specific language for the development of CMS-based web applications was presented. Although, this language has a great high level of abstraction, so it's necessary to define mechanisms to transform this language to a more "machine friendly" representation.

This chapter begins with a brief introduction of the CASE tool in which the transformation process for CMS-ML was develop in the context of this work, in order to understand the technicalities associated with the tool and consequently, with the transformation. The platform independency is an important CMS-ML characteristic, so this chapter explains what is a platform independent model and a platform specific model to better understand the transformation. A detailed architectural view of the transformation process is given, divided by two sections, according to the transformation step or level. Then an explanation of the three transformation directions is presented, which is an important characteristic of the transformation process. Finally, the chapter ends with an important section that presents a detailed description of the transformation process adopted in the CMS-ML approach.

4.1 ProjectIT-Studio CASE Tool Overview

Due to the context of this work, a CASE tool is required to support the development and maintenance of CMS-based web applications, using the CMS-ML language defined in the previous chapter. A CASE tool is a special software that aims to support activities of software engineering, related with development methodologies [36, 48]. The adoption of a CASE tool presupposes a predisposition to the application of rules and principles to all the application development process.

ProjectIT-Studio [50] is a CASE tool focused in the realization of high productivity activities associated with the management and specification of requirements and tests, model design, automated code generation and software development.

ProjectIT-Studio is part of the research initiative ProjectIT [23], that aims to analyze, integrate and support best practices of management and development of information systems projects.

In a very high level fashion, ProjectIT-Studio is an orchestration of plug-ins on the platform Eclipse.NET and is composed by three main component (figure 4.1): Requirements, UMLModeler and MDDGenerator [50]. The *ProjectIT-Studio Requirements* component is a tool and a textual domain specific language for the specification of requirements documents [50]. The requirements component is not relevant for this dissertation work, so the ProjectIT-Studio was chosen because of the other two components (i.e., UMLModeler and MDDGenerator), its usability and the XIS2 approach, presented in section 2.2.4, that

also transform models-to-code with this tool.

The *ProjectIT-Studio UMLModeler* is a complete tool that allows the designers to create visual models through UML2 language or profiles derived from it [55]. On the other hand, the *ProjectIT-Studio MDDGenerator* is the component responsible for the definition and management of transformation processes model-to-code [52]. This transformation mechanism allows the manipulation of UML2 models, developed with the UMLModeler component, and generate source code or documentation, according to templates [21].

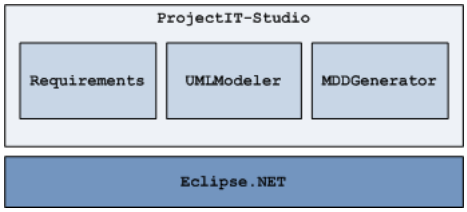


Figure 4.1: Overview of ProjectIT-Studio (extracted from [52])

The transformation of model-to-code based in the template mechanism offers advantages when compared to other approaches, in particular: promotes iterative and incremental development because it facilitates the continuous refinement and modification of templates; it's independent of source language (UML2, C#) or target platform, and simplifies process of generation of any textual artifacts, including documentation [21].

An overview of the *ProjectIT-Studio MDDGenerator*, shown in figure 4.2, is the most important ProjectIT-Studio component for this dissertation, so this chapter presents a more detailed introduction about it. The *MDDGenerator* receives one or more templates and the model, then it produces a textual artifact (for example: code or documentation), to do so, it uses a template engine, as shown in figure 4.2.

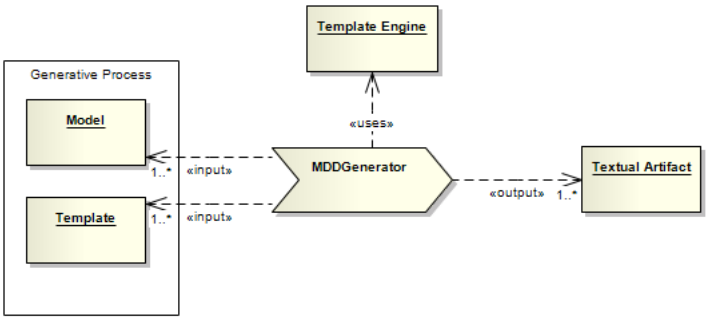


Figure 4.2: Overview of *ProjectIT-Studio MDDGenerator* (adapted from [64] page 275)

In order to use the CMS-ML language with the ProjectIT-Studio tool, a change in the tool was required, specifically, a change in the MDDGenerator so that it can support the CMS-ML Meta-Model. This change was done in the context of this work and will be referred in the section 4.5.

More than introducing the *ProjectIT-Studio* tool, this section gave the motivation for the usage of this tool in this dissertation, which is mainly due to the *UMLModeler* and the *MDDGenerator* potential when associated in the development process in a MDE/MDD way.

4.2 Platform Independent Model and Platform Specific Model

This section presents the two fundamental concepts for model transformation, the PIM and PSM, and what's the relationship between them.

The **Platform-independent model** (PIM) is a model that's independent of implementation technology and is at a high level of abstraction. It focuses on the operation of a system while hiding the details necessary for a particular platform and describes a complete specification that does not change from one platform to another [40], but does not show its details. A PIM is similar in concept to logical models from structured development [20] or essential models [12].

In the other hand **Platform-specific model** (PSM) is a model created by transforming a PIM tailored to specify a system in terms of implementation constructs available in one specific technology. It combines the specifications in the PIM with the details that specify how that system uses a particular type of platform [40]. A PSM may provide more or less detail, depending of its purpose. Although, a PSM may be an implementation, if it provides sufficient information to construct a system, or it may be a more specific and limited PIM that will be used for further refinement to a PSM that can be directly implemented [65]. For systems built using several technologies usually there is the need for one PSM for each technology.

These two models have a well defined relation between them, this relation is crucial as it will allow the transformation of one model to another. The figure 4.3 shows the meta-model of the relation between PIM and PSM.

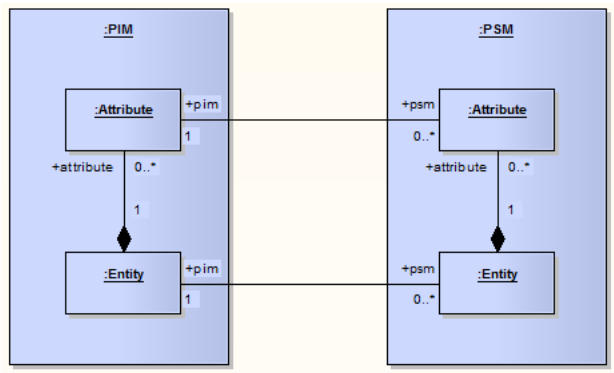


Figure 4.3: Relation between PIM and PSM

PIM is an important concept because it's supposed to survive the constant changes in software technologies. When a new technology emerges, the PIM remains stable and with reusable entities that are transformed to a new PSM. PIM is a crucial concept in the context of this work because it forms the background for the CMS-ML Models.

4.3 Transformation Processes

The previous section introduced the PIM and the PSM concepts, crucial ones in the MDA approach. The main goal of this section is to introduce the PIM and PSM transformations according to the MDA approach to better understand and discuss the transformation process adopted in this dissertation, which shall be presented in the following section. The transformation process adopted in this dissertation doesn't transform the PIM into a PSM (section 4.5 details the adopted transformation process), although it's interesting to contemplate this approach in the theory, for analysis reason and for possible future work.

So what is a transformation? Broadly, a transformation is a process that receives a source element as input and generates, through a transformation process, a target element as output.

Figure 4.4 shows the MDA approach for a generic model transformation [40]. Although, this model is very generic and may be applied in various areas, this work is interested in a more specific area, the web applications and one of its subareas, the CMS-based web applications. Figure 4.5 shows a comparison of three possible application areas for the MDA transformation proposal. Figure 4.5 shows the PIM and PSM levels: a CMS PIM is more specific than a web application PIM that is more specific than a generic application PIM, this specificity ladder is represented in the figure; the same level concept applies to the PSM. On the other side, are the platform artifacts, all platform specific elements like source code and documentation will be called by *Artifact*. In figure 4.5 we can also observe that a CMS artifact is contained on a CMS platform, a CMS platform is a sub set of the Web applications platform, which is a sub set of a generic application platform.

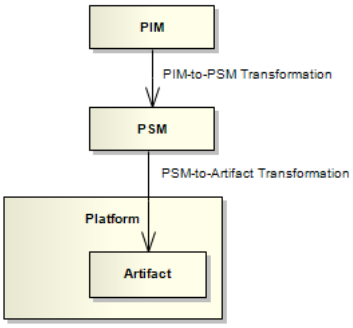


Figure 4.4: Conceptual view of PIM to PSM transformation

All the arrows in figure 4.4 and 4.5 represents a transformation, we can observe that two transformation processes are required in order to transform a model (PIM) into platform artifact. Let's call the first transformation process of *PIM-to-PSM Transformation* and the second *PSM-to-Artifact Transformation*. As the figures suggests, the first transformation is a more high level transformation and the second is a more specific transformation, which requires more platform knowledge and may be very complex, depending of various factors, such as platform complexity and the developers expertise. The definition of these transformations concepts is crucial in order to develop a transformation for the language defined in chapter 3, so they're detailed in the following sections.

4.3.1 *PIM-to-PSM Transformation*

As briefly mentioned in the previous section, the *PIM-to-PSM Transformation* is a high level transformation, specifically, this transformation receives a source model as input and generates a target model as output. A PIM usually has the source role and the PSM has the target role. PIMs may be transformed to PSMs that include information specific to the current state of the art implementation technologies. Later, PSMs can be used to generate platform artifacts.

This transformation involves more complexity then the *PSM-to-Artifact Transformation*, presented in the previous section, because the source model is platform independent and the target, is platform specific. This particularity gives complexity to the process, as the mapping between the two models may not be trivial for various reasons, such as the lack of an element in the target platform or the mapping to a platform isn't straightforward. This is more clearly understand when we keep in mind the concepts mentioned in the previous sections, that the source model, the PIM, is very generic and can have elements that do not exist in the target model, the PSM. The relation between model elements is described in

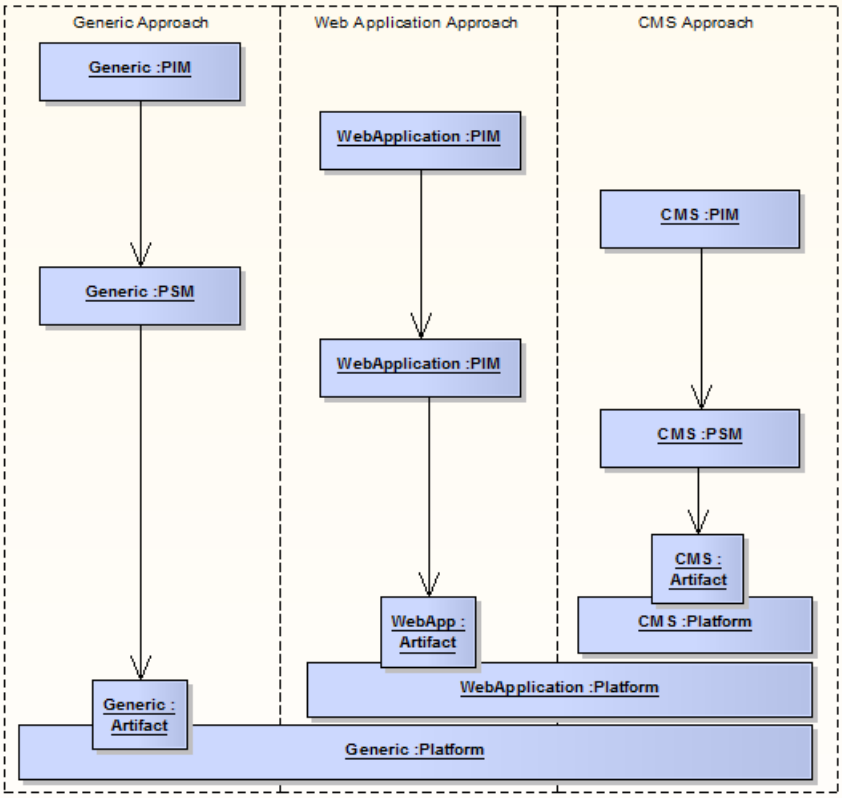


Figure 4.5: MDA approach, Web Application Approach and CMS Web Application Approach

figure 4.3.

The most common mechanism used to specify how this transformation will proceed is called mapping. The mapping is a platform specific specification for the transformation of a PIM into a PSM. Mappings should be easy to understand and write, in order to easily support current and emerging technologies.

The most commonly adopted mapping is the so called model instance mapping [40]. The model instance mapping identifies model elements in the PIM which should be transformed in a particular way to specific PSM elements. This mapping allows the addition of information in the PIM to be used by the transformation, this information pieces are called marks. PIM elements are marked, the marks in the PIM elements points to a specification in the mapping. All the marked PIM elements are transformed, according to a mapping specification, to the corresponding PSM elements, which can not be straight and involve several manipulations to adapt a PIM element to one or more PSM elements.

Marks are applied to one or more PIM elements that indicate their roles in the mapping, the result is used to develop a PSM that may have a different set of elements, as illustrated in figure 4.3.

The *PIM-to-PSM Transformation* is usually a complex process, so the most reasoning strategy to attack this process is "divide to conquer", i.e., use the mapping concept and divide this transformation in two steps. Figure 4.6 shows the two parts of the transformation. The input to the transformation process is the PIM and the output is the PSM. The first part receives the PIM (which is the input of the transformation process) and the marks to that PIM, specified in the mapping, and the result is a marked PIM. The second part receives the marked PIM and the mapping, and produces a PSM (which is the output of the transformation process).

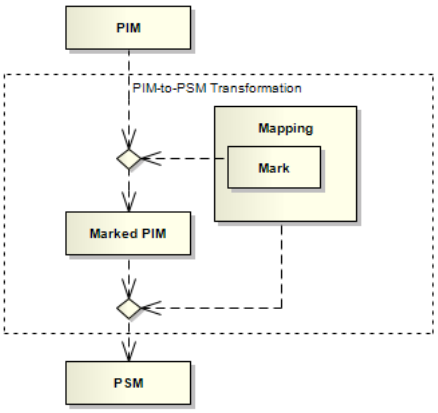


Figure 4.6: The PIM-to-PSM Transformation Process

4.3.2 *PSM-to-Code Transformation*

As mentioned before, there's a second transformation process that receives a PSM and produces specific platform artifacts. This transformation is described in this section.

The common strategic approach to this transformation is called template-based [13]. A template consists of the target text containing splices of meta-code to access information from the source and to perform code selection [78]. Templates lend to iterative development as they can be easily updated with new information.

To generate platform artifacts, the process identifies parts of the PSM and maps them, according to a platform specific template, to one or more platform artifacts. Any PSM element may have a form of representation in the platform, but that is not mandatory and only depends on the templates definition.

The figure 4.7 illustrates the *PSM-to-Artifact Transformation* in detail. The platform artifacts are generated through the combination of a template and a PSM. Note that, all the elements in this transformation process (PSM, Template and Artifacts) are platform specific, so it's necessary to define a template based on a PSM and platform artifacts.

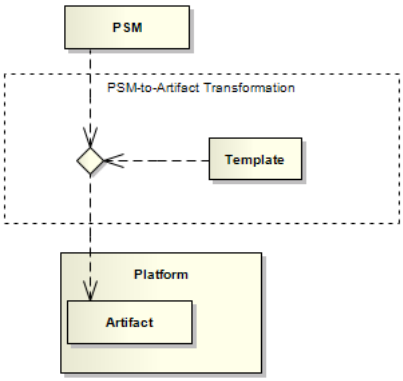


Figure 4.7: The *PSM-to-Artifact Transformation* Process

4.4 Forward, Reverse and Round-Trip Transformation

The previous sections defined the transformation processes that followed only one direction, a top-down transformation process, from PIM-to-code, passing through a PIM-to-PSM and PSM-to-Code trans-

formation. It's interesting to introduce some functionalities related with the code and documentation generation.

The direction is a crucial concept in model transformation because it's a property that represents the versatility of the language and the transformation process capabilities.

Several authors [19, 58, 64] refers to two or three transformation directions, but there's no standard for the directions names. We adopted the names defined in [64]: *Forward*, *Reverse* and *Round-Trip* transformation, illustrated in figure 4.8.

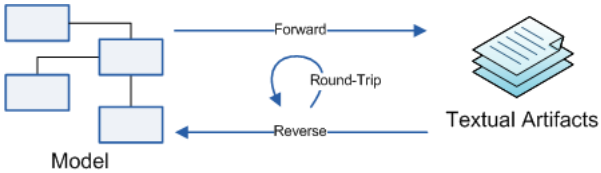


Figure 4.8: Forward, Reverse and Round-Trip Transformation (adapted from [64] page 158)

Forward transformation is when one model has the role of source and another model has the role of target, and the transformation process has only one direction [21]. Suppose that there's a PIM, the transformation process receives this PIM and produces a PSM, as already mentioned. This transformation process is unidirectional; each model only has one role at a time. Information may be lost in the process and it's not possible to do the inverse transformation, i.e., to generate the PIM through the PSM. The same principle applies to PSM-to-Code.

Reverse transformation shares the same principles of *Forward* transformation except that the transformation has the oposite direction. This means that, in this type of transformation we can produce a model (PIM, PSM or both) from existing platform textual artifacts, like code or documentation.

Round-Trip transformation is when both models can assume the role of source and target in different instants and the transformation process is possible in both directions [21]. In this type of transformation it's normal to lose information in the process, when the transformation has the source model as a more specific model than the target model. Suppose that there is a PIM, the transformation process receives this PIM and produces a PSM or Artifacts, like the unidirectional process, but in this type of transformation it is possible to generate the PIM, through the PSM or Artifacts. Although, the final PIM may be incomplete due to the lack of information lost in the process. All depends of the mappings, templates and the PIM expressiveness.

The Round-Trip transformation enables the maintenance of web applications using models, as the process allows the PSM-to-Code and the Code-to-PSM transformations, this way the developers can model a web application then generate the web application code and posteriorly, automatically build a model from an already existing web application.

4.5 CMS-ML Model Transformation Process

The previous sections introduced, in a briefly and simple way, the transformation process according to the MDA approach. However, the approach used in this work doesn't follow the MDA "by the letter"; instead it adopts some aspects of the MDA approach that are most suited to this dissertation. This section gives a detailed description of how the transformation is applied to CMS-ML models using the ProjectIT-Studio CASE tool.

Why not use the MDA approach as described in the previous section? In MDA, a PIM is transformed into a PSM; the result of this step is another model that offers platform specific details. Besides this benefit, in this work, no other benefit was identified and this one doesn't bring advantages enough to

justify the development of such process, on the contrary, the PIM to PSM transformation process will only bring unjustified and unnecessary complexity to a development process that aims to be as simple as possible. Suppressing this transformation step also suppresses the need for a mapping mechanism between PIM and PSM, redirecting all the developers' effort to the templates definition when new CMS platforms are adopted. We shall see how the transformation is easily done only by the use of templates.

The model transformation approach adopted in this work is the same of the XIS/XIS2 on ProjectIT-Studio [48, 60, 36, 66]. Figure 4.9 illustrates overview of how a CMS-ML model is transformed to platform artifacts.

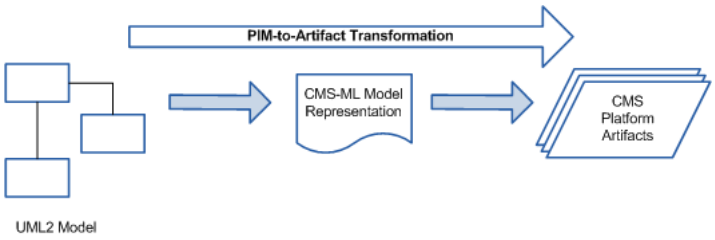


Figure 4.9: Overview of the Transformation Process of CMS-ML Models

Instead of having two transformations, *PIM-to-PSM* and *PSM-to-Artifact*, we have only one *Forward* transformation, the *PIM-to-Artifact Transformation*. From the figure 4.9, we can see that the *PIM-to-Artifact Transformation* contains 2 steps, but in fact the first step requires no additional development for new CMS platforms, it only exists for development and testability reasons. This way, the transformation process receives as input a *CMS-ML Model Representation*, which is build from the model, and not a UML2 Model with the CMS-ML profile.

The following sections explain with more detail both steps, but before that, a description of the required project structure for the transformation process is given.

CMS-ML Project Structure

The project structure is very simple and can be easily built automatically, but for now the tool does not supports this feature, so the analyst must perform the simple task of creating it.

The CMS-ML project structure is basically the same as XIS/XIS2, only for standardization reasons and there's no need to change it, we want to support both approaches with the ProjectIT-Studio. Figure 4.10 shows an example of a project structure in ProjectIT-Studio.

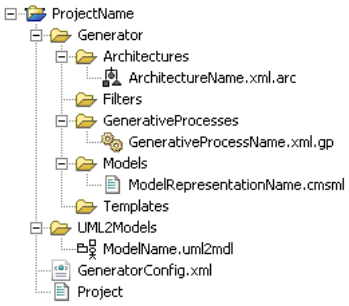


Figure 4.10: CMS-ML Project Structure in ProjectIT-Studio

The *UML2Models* folder is specially to aggregate all the models of the project, usually there is only one UML2 model per project, but if more than one exists, they should all be in this folder.

The *Generator* contains four folders that are associated with the generation process of model transformation. Each folder contains files of the correspondent types: architectures (".arc"); generative processes (".gp"); templates (".tpl"); and model representations (".cmsml"); all these concepts are introduced later in this chapter, section 4.5.2.

The first step when we create a new project is to create the folders tree. After we need to add a special file, required by the MDDGenerator, which configures the generator engine to use the folders of the project structure. Figure 4.11 shows an example of the *GeneratorConfig.xml* file for the project structure of figure 4.10.

```
<GeneratorConfig>
   <Architectures baseDirectory="Generator\Architectures"/>
   <Models baseDirectory="Generator\Models"/>
   <GenerativeProcesses baseDirectory="Generator\GenerativeProcesses"/>
   <Filters baseDirectory="Generator\Filters"/>
   <Templates baseDirectory="Generator\Templates"/>
</GeneratorConfig>
```

Figure 4.11: Generator XML configuration file for the ProjectIT-Studio MDDGenerator

By making a file like the one in figure 4.11, we are configuring the generator to use the correct folders for architectures, models, generative processes, filters and templates (see figure 4.10).

The architecture and generative processes files are trivially created, the developer only needs to create an empty architecture file and a generative process file in the architecture and generative processes folder, respectively. The UML2 model file should be created in the UML2Models, using the "ProjectIT-Studio Wizards" that automatically creates an empty UML2 model file.

4.5.1 *UML2 Model to CMS-ML Model Representation*

This section describes the first part of the CMS-ML transformation process, illustrated in figure 4.12. The input is an UML2 model with the profile defined in section 3.3, this model is in fact a CMS-ML model but it's represented in the form of UML2 elements. This model is converted to a *CMSMLMetaModel*, which is a set of inter-related memory objects. Finally, the objects are serialized into a CMS-ML representation in the form of a XML file (with extension ".cmsml").

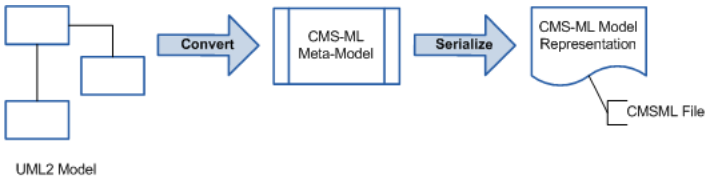


Figure 4.12: The first step of the *CMS-ML Model Transformation* Process, the *UML2 Model to CMS-ML Model Representation*

This way, when one wants to develop a CMS-based web application using the CMS-ML language, the first step is to create an UML2 Model using the profile defined in chapter 3. The resulting UML2 model has a specific representation that is not related with the CMS domain, it consists of UML elements with stereotypes, defined in the CMS-ML profile.

An UML2 model in ProjectIT-Studio (built with the UMLModeler component, referred in section 4.1) has an internal representation in memory. This representation is a typical objects model with classes for each UML2 element, like *Association*, *Class* and *Attribute*. Obviously, this UML2 objects model representation is appropriate when we want to work with UML2 elements; that's not the case in this work. The goal is that, in the templates developments, the programmers only uses CMS-ML elements

(i.e., *WebApplication*, *WebPages*, *WebComponents* and all the ones defined in section 3.3) and not UML2 elements (i.e., classes, associations, attributes). So there's the need to convert it to a form that is more close to the CMS-ML language, in order to only use concepts specific to the CMS domain on the development of the templates.

To better understand this necessity, let's follow the example of figure 4.13. The example is a viewpoint of the CMS-ML model that illustrates a simple *WebPage*, which is an UML2 class element with the stereotype *WebPage*, *Containers* and *WebComponents*.

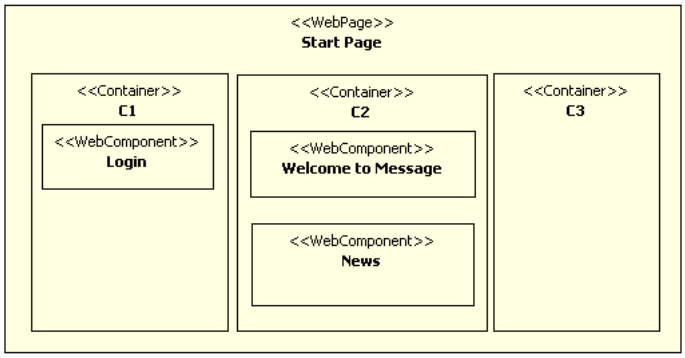


Figure 4.13: Simple example of a *WebPage*

Converting the *WebPage* UML2 class element to a CMS-ML memory object is almost done trivially and involves some simple validations (like checking the stereotype and the correspondent tagged values), what's not trivial is determining the containments, for example determining that the *Container* "C1" is part of the *WebPage* "Start Page".

All the elements from all the model views are converted to the CMS-ML Meta-Model, but the biggest challenge in this conversion step was the *Roles* hierarchy, namely: to propagate the permissions up in the roles hierarchy tree and making sure that no duplicated permissions exist nor contradictory permissions.

```
<WebApplication identifier="3215f9d3-2ab8-477c-a78b-db3dcb4cb733" name="TestWebApplication" configuration=""
                language="English" visualStyle="VisualStyleOne" homepage="a7526a9e-696d-4229-b762-9b208c172eec">
   <WebPages>
      <WebPage identifier="a7526a9e-696d-4229-b762-9b208c172eec" name="Start Page" configuration=""
               type="Front-End" language="Spanish" visualStyle="VisualStyleTwo" parent="">
         <Containers>
            <Container identifier="b4fec3e3-7b05-4f99-baab-22aca9b64806" name="C1" configuration="" order="1">
               <WebComponents>
                  <WebComponent identifier="1bbf4e00-ef8b-45c9-befa-8aad09616e49" name="Login" configuration=""
                                type="login" content="" order="1" />
               </WebComponents>
            </Container>
            <Container identifier="43d0b770-1ba9-4bee-b317-8afdbc4c3634" name="C2" configuration="" order="2">
               <WebComponents>
                  <WebComponent identifier="94bb7a7b-6b5c-468b-94c0-d02fd263fd11" name="News"
                                configuration="listNumber=10;format=detailed" type="recent" content="" order="2" />
                  <WebComponent identifier="fae0231f-86f4-46be-9194-8cf3c72286d9"
                                name="Welcome to Message" configuration="" type="content"
                                content="Welcome to this page. Feel free to browser" order="1" />
               </WebComponents>
            </Container>
            <Container identifier="597dce73-aecb-44a6-b197-2ca096334665" name="C3" configuration="" order="3" />
         </Containers>
   </WebPage>
   </WebPages>
   <Roles />
   <Users />
</WebApplication>
```

Figure 4.14: The CMS-ML representation for the *WebPage* example

Figure 4.14 shows the CMS-ML representation (".cmsml" XML file) for a *WebApplication* that only

contains the *WebPage* of the example illustrated in figure 4.13. The CMS-ML Model representation has only concepts related to the CMS Domain and all the XML elements and attributes in the file correspond to the ones identified in section 3.

Summarizing, the first step in the *CMS-ML Model Transformation* Process resumes to transforming an UML2 Model with the CMS-ML profile to a ".cmsml" file that has a representation independent from the UML2 specification and is more close to the CMS Domain.

CMSML Meta-Model Module

The *CMSMLMetaModel* module is the implementation of the meta-model of the CMS-ML modeling language, i.e., a model representation in the form of a network of objects, which is used in the definition of model-to-model or model-to-code transformation. The structure of the *CMSMLMetaModel* is consistent, as supposed, with the definition of the CMS-ML language defined in chapter 3.

This module was developed in the context of this work and has two main components that enable: (1) convert a model with the UML meta-model to a correspondent model with the *CMSMLMetaModel*; and (2) serialize and deserialize the *CMSMLMetaModel* from and to a XML format. The first component is necessary because the models created in UML with the CMS-ML profile are stored in the UML meta-model to then use it in the transformation process. The second component is necessary to configure the generative processes.

4.5.2 *CMS-ML Model Representation* to *CMS Platform Artifacts*

The previous section described the first step in the *CMS-ML Model Transformation Process*, the *UML2 Model to CMS-ML Model Representation* transformation. This section presents the second step of the process, which receives the result of the first step, the CMS-ML Representation in the form of a ".cmsml" file, and produces the platform artifacts.

Figure 4.15 illustrates the *CMS-ML Model Representation to CMS Platform Artifacts* in detail.

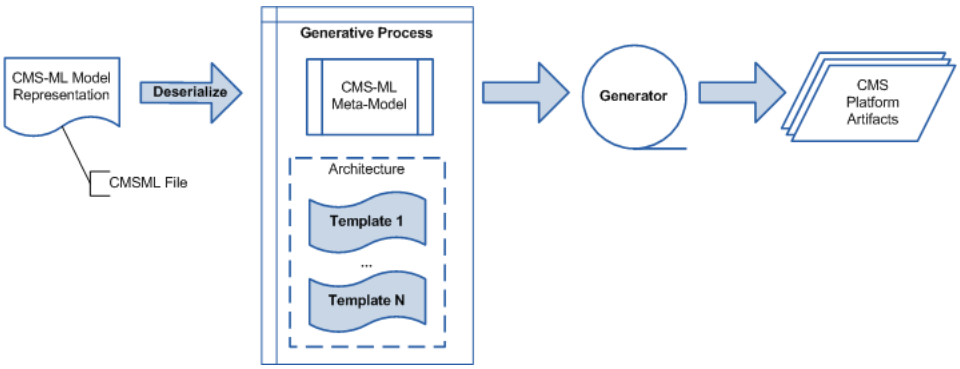


Figure 4.15: The second step of the *CMS-ML Model Transformation* Process, the *CMS-ML Model Representation to CMS Platform Artifacts*

The second step of the transformation process begins with the deserialization of the ".cmsml" file, which resulted from the first step, and produce an internal representation in memory of *CMSMLMetaModel* objects, exactly the same that resulted from the UML2 Model conversion in the first step.

In order to describe this transformation step, three new concepts must be introduced: architecture, template and generative process. Next, these concepts are defined.

Template

A **template** is a crucial element in the model transformation. A template is a formal technical definition of a transformation model-to-artifact, developed in a specific language similar to ASP (Active Server Pages) [57]. The language as a set of elements to define templates (see appendix A of Rui Silva work [66] for more detail) that are used with code blocks of C#.

Figure 4.16 shows an example of a template. This example is only informative and it's not in the scope of this work to analyse and discuss the templates definition, for a more detailed discussion about the templates and the templates engine, refer to Rui's Silva work [66].

```
<%@ Template Type="JoomlaCMS" Description="" %>
<%@ Assembly Name="CMSMLMetamodel" %>
<%@ Import NameSpace="CMSMLMetamodel" %>
<%@ Argument Name="webApplication" Type="WebApplication" %>
<%@ Output File="JoomlaDeployment.sql" %>
<% bool first; %>
<!-------------------------------------------------------->

USE `joomla_db_schema`;

<%
// Create WebPages
foreach (WebPage webPage in webApplication.WebPages.Values)
{
%>
   INSERT INTO `jos_sections` (`title`)
   VALUES ("<%= webPage.Name %>");
<%
}
%>
```

Figure 4.16: Template example for the creation of all *WebPages* defined in the *WebApplication* model.

As already mentioned, templates must be developed for each new technological platform. So we must keep in mind that in generative programming approaches, instead of writing the code, we'll be writing the templates that produce the code or documentation. However, as flexible and powerful as this is, we don't need to start building templates for the same platform each time there is the need to development a new application on top of it, instead we use a template already developed for that platform. In other words, templates for a specific platform are built one time only and reused many times (partially and sometimes totally). This way, templates are a focus of great effort when adopting a new CMS platform.

Architecture

In the context of CMS-ML model transformation process, an **architecture** is a coherent and interconnected group of templates. The process of creating an architecture is to group templates that meet the requirements. In this process, we also define the platform for the generated artifacts, because the templates are representations of artifacts for specific platforms (eg. Java, C#) and can by categorized according to its purpose (e.g., database, business logic, documentation). This way, we can determine certain aspects of the target platform.

ProjectIT-Studio has a graphical interface for the specification of architectures for generative processes, figure 4.17 illustrates an example of an architecture for a CMS-ML project. In this example, we have six templates aggregated by three categories: DrupalCMS, JoomlaCMS and WebComfortCMS. Each category has two templates: one template generates documentation for the CMS-ML model and the other generates the correspondent CMS platform artifacts to be deployed. In this example, we are only selecting for the architecture the DrupalCMS and the JoomlaCMS templates, excluding the WebComfortCMS templates; this means that the selected templates will be available for the generative process.

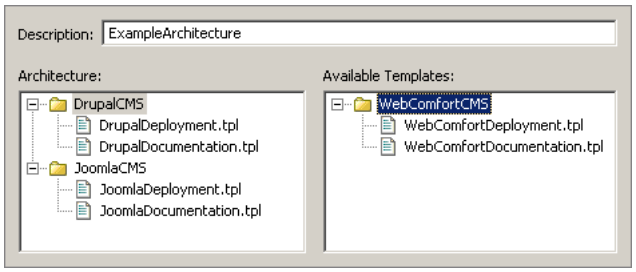


Figure 4.17: Architecture example with six templates

Generative Process

The **generative process** is a simple component and basically it's an input configuration for the generator (ProjectIT-Studio MDDGenerator), which associates a model to the architecture.

The developer can configure the generator through the selection of an architecture and a model. The generative process is responsible for the model instantiation and execution of the templates of the architecture. To do so, it receives as input a model defined in the CMS-ML language and an architecture where we desire to generate the application.

4.6 Summary

This chapter presented the tool and the mechanisms that support the CMS-ML proposal. The chapter begun with a brief introduction to the supporting tool, the ProjectIT-Studio, in order to better understand the transformation mechanisms, which are related to the tool. After, a reference to the MDA approach was given, with two crucial concepts about model transformation, the PIM and PSM, and the transformation processes required to transform a PIM-to-Code. Finally, the chapter ended with the definition of the transformation process adopted for the CMS-ML model transformation using the ProjectIT-Studio, which is composed by two steps: the *UML2 Model to CMS-ML Model Representation* followed by the *CMS-ML Model Representation* to *CMS Platform Artifacts*.

Chapter 5

Methodology for the Development of CMS-based Web Applications Using the CMS-ML Language

In chapter 3, a formal definition of a DSL for the development of CMS-based web applications was given. Then chapter 4 presented the CASE tool used in the context of this work and described the model transformation process, required to transform a CMS-ML model into platform artifacts. At this point, all the background is defined in order to use the CMS-ML for the development of complex CMS-based web applications. Although, how do the developers achieve an UML2 model with the CMS-ML profile capable to express a web application, and how to do the transformation with ProjectIT-Studio? These two questions will be answered in this chapter, which presents a simple methodology, guidelines and inter-related processes for the development of CMS-based web applications using the CMS-ML language supported by the ProjectIT-Studio CASE tool.

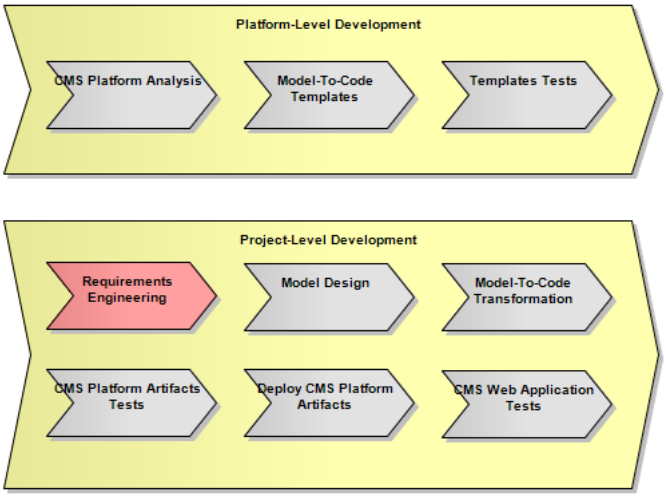


Figure 5.1: Development Processes Overview

The approach proposed in this chapter comprises two processes, illustrated in figure 5.1: (1) **Platform-Level Development**: which comprises activities that focus on the development of mechanisms that enable the model transformation for new CMS platforms; and (2) **Project-Level Development**: contains

activities that aims to produce a CMS web application and other artifacts (for example documentation).

There's an isolated activity, the **CMS-ML Profile**, not shown in figure 5.1, where the CMS-ML Profile is defined by the architect, based on chapter 3, in a form that the analysts can use it to develop models. This activity is not included in the *Platform-Level Development* nor in the *Project-Level Development* processes because it isn't executed for each new platform nor for new projects. The CMS-ML profile is defined once and re-used many times on both *-Level Development* processes. Both development processes cooperate to achieve a quality web application.

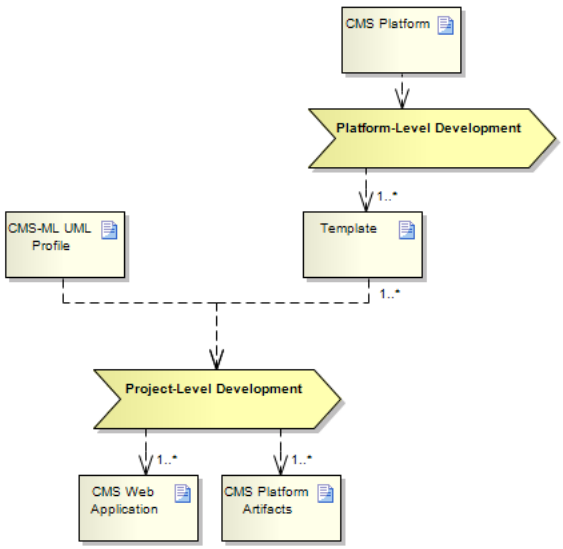


Figure 5.2: Artifacts Flow Overview

Project-Level Development process receives a set of templates, produced by the *Platform-Level Development* process and the CMS-ML Profile and produces CMS Platform Artifacts and CMS-based web applications. Figure 5.2 illustrates the development processes overview where we can see the artifacts interchange.

Each of the processes and its activities will be detailed in the following sections.

This chapter does not intend to be a detailed manual to the CMS-ML language nor to the ProjectIT-Studio. As any language, the CMS-ML must provide a well defined syntax and semantic, which is given in chapter 3 and complemented with some guidelines given in this chapter. The ProjectIT-Studio already has a very good set of works, all referred during this dissertation, so this chapter doesn't presents a detailed description of how to do some actions with the tool; the reader is highly encouraged to consult the ProjectIT-Studio manuals whenever needed.

5.1 CMS-ML Approach Development Actors

Before describing each process, there is the need to define which actors are involved in the development processes, in order to better define each process and their responsibilities.

The approach proposed in this work - the CMS-ML DSL and the transformation mechanisms that produces CMS platform artifacts - has a set of tasks and, at this point, no reference to the intervenients in the process has been given, which is highly required for a well defined and successful approach. This section identifies the development process actors that are responsible for some development tasks associated with the CMS-ML approach. Five primary actors exist:

- **Architect**: the architect is responsible for conceptual and high level tasks, namely the definition of the CMS-ML UML profile. Notice that the CMS-ML profile defined in this work, for the *ProjectIT-Studio Studio*, can be adopted, however, the approach and tools are independent of the adopted profile, which means that a new CMS-ML profile must be developed for new tools other then *ProjectIT-Studio*.

- **Analyst**: the analyst is responsible for the requirements acquisition, through the application of requirements techniques like reunions or JAD. These requirements are formally defined in the most appropriate way for the project or as the organization demands. After the requirements validation, the analyst is responsible for defining the model that reflects the requirements, for this he must use the CMS-ML UML profile, defined by the architect. He's also responsible for the analysis of new CMS platform in order to produce a CMS platform specification.

- **Programmer**: the programmer has two major tasks. The first is to give a more technical form to the CMS platform specification designed by the analyst, in other words, to implement the transformation templates for a new CMS platform. He's also responsible for producing the CMS platform artifacts, to do so, he receives the model defined by the analyst and applies the transformations to the model (model-to-model and/or model-to-code), based in the transformation templates previously implemented. Also this role is required to implement transformation mechanisms not supported by the profile or for developing new features not supported by the target CMS platform.

- **Tester**: the tester has all the tasks related with quality assurance, namely: testing the new CMS platform templates, the CMS platform artifacts and the final CMS-based web application.

- **Integrator**: the main task of the integrator is to plan and execute the deployment or installation of the CMS platform artifacts, which resulted from the transformation process, producing a requirements-aligned web application.

Although the five actors were always referred in the singular, the reader should keep in mind that some tasks cannot be carried by only one singular individual, but by a team. It highly depends on each project characteristics.

5.2 Platform-Level Development

The approach proposed in this dissertation aims to be platform independent, through the use of model transformation templates. This means that templates are platform specific and new templates are required when the developers intend to build a web application in new CMS platforms.

The *Platform-Level Development* process has a small set of activities with the goal to develop templates for a new CMS platform, as illustrated in figure 5.3.

This process is very simple in terms of activities and artifacts, it consists of an analysis of a particular CMS platform and later the development and testing of a set of templates that should include all aspects of the platform in analysis.

This section describes the *Platform-Level Development* process, along with its intervenients and activities details.

5.2.1 CMS Platform Analysis

When analysts design a web application to respond to a set of requirements, they want to use all the capabilities of the target CMS platforms and want to have the final product as a high quality product, regardless of the platform.

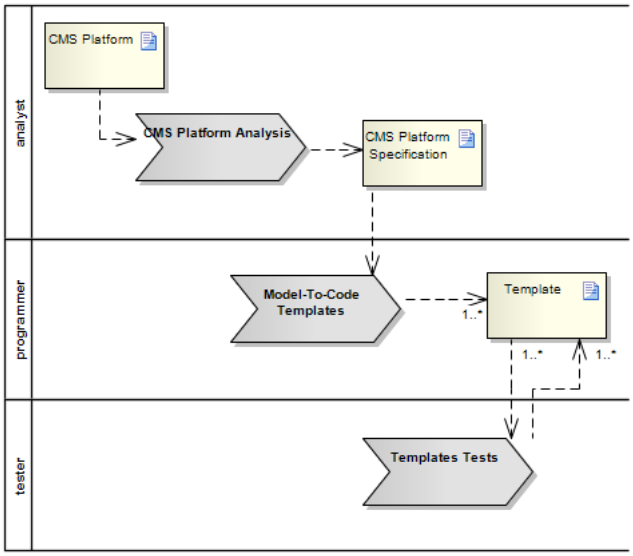


Figure 5.3: Project-Level Development Process Overview

Platform analysis activity focuses on producing one or more artifacts that fully describes all features of the CMS platform, needed to produce a full-featured high quality web application on that CMS platform. Analysts can use several techniques to specify the platform, from simple textual descriptions to special notations, the important is to express all features and the "how" to conceive each feature through the use of CMS-ML concepts; for example: does the target CMS platform has web pages, can we represent it with a *WebPage* CMS-ML element and how to transform the CMS-ML concept to a platform artifact?

Ideally, the analysis of the CMS platform should be made in order to meet the language CMS-ML, to align CMS-ML concepts with platform specific concepts. This way the analyst can specify precisely the mapping between the CMS-ML concepts and platform-specific concepts that will be posteriorly implemented by the programmers.

Each CMS platform has a specific domain which is a subset of the CMS domain, as specified in section 3.2. Sometimes these subsets leave the whole area of CMS, in these cases the CMS-ML language may not be sufficient to specify few of the concepts that are not included in the field of generic CMS; figure 5.4 illustrates a particular CMS platforms domain scenario.

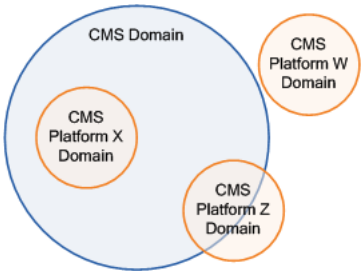


Figure 5.4: CMS Platforms Domains Intersections

From figure 5.4 we can see that three possible cases exists: (1) CMS Platform X Domain - the platform only has standard concepts, which are all supported by the CMS-ML language; (2) CMS Platform Z Domain - the platform has many standard concepts but also has specific concepts, not supported by the CMS-ML; and (3) CMS Platform W Domain - in this case the CMS-ML language is unable to model web applications, because the target platform doesn't comprises any of the CMS-ML concepts.

Cases where a CMS platform contains concepts that are not contemplated by the CMS-ML language must be clearly identified and defined in order to achieve a solution to these cases. Such cases can be solved in three ways: (1) patch the templates for the platform with the specific concepts, "hard-coded", for each project; (2) if these concepts are important and relevant in the field of CMS, extending the CMS language with these specific concepts will be the most correct solution; or (3) use an existing CMS-ML element that best suits the new concept and set a key-value pair in the string configuration that represents the new concept; example: if we want to support plug-ins, we can use the *WebComponent* element with the pair "special=plug-in" in the configuration string and the templates would treated this *WebComponent* accordingly.

Another extremely important part of the analysis is the identification and definition of the "out-of-the-box" web components of the CMS platform. This is because the templates have to be able to instantiate and configure each web component, and each one is a different case. The analysis must also consider how new web components are developed for the CMS platform, because new web components will certainly be required for new projects.

5.2.2 Model-To-Code Templates and Templates Tests

The *Model-To-Code Templates* activity is performed by the programmers and aims to develop a set of templates for a specific CMS platform. To do so, the programmers receive a CMS platform specification, conceived by the analysts, that has all the information required to produce the templates.

A template [13] is a copy of the artifact (e.g. HTML page, source file, etc.) with built-in actions that a template engine evaluates when processing the template, similar to the ASP [75] (Active Server Pages) (refer to section 4.5.2 for more detail about the templates).

By themselves, the templates are sufficient to define these transformations, because they contain all the instructions to transform a model to some type of textual artifact. It's not in the scope of this work to present an extensive description of the templates and the templates engine of ProjectIT-Studio, although the chapter 4 of the Rui Silva work [66] describes this topic in detail.

Unlike the templates development, testing templates is a task of extreme difficulty. Templates are not a piece of executable software; they are an input to a process that merges them with a model to produce artifacts. It's crucial to test the templates to reduce the errors in the *Project-Level Development* process.

The tests are performed by the testers and consists of submitting the platform templates to several test-case models and then validate the transformation output.

5.3 Project-Level Development

This section describes the *Project-Level Development* process, along with the its intervenients and activities details.

The *Project-Level Development* is a process with a structure imposed on the development of CMS-based web application products for clients. It includes a set of inter-related activities performed by different actors.

This process does not intend to give a development life cycle, the only purpose is to give a set of activities and artifacts with the goal to produce CMS-based web applications.

Figure 5.5 shows the overview of the *Project-Level Development* process with all its activities, actors and the artifacts inputted and outputted for each activity.

Before starting modeling the web application, the analyst must gather all the stakeholders' requirements. The requirements engineering activity is by its own a subject of great work, so it's not discussed

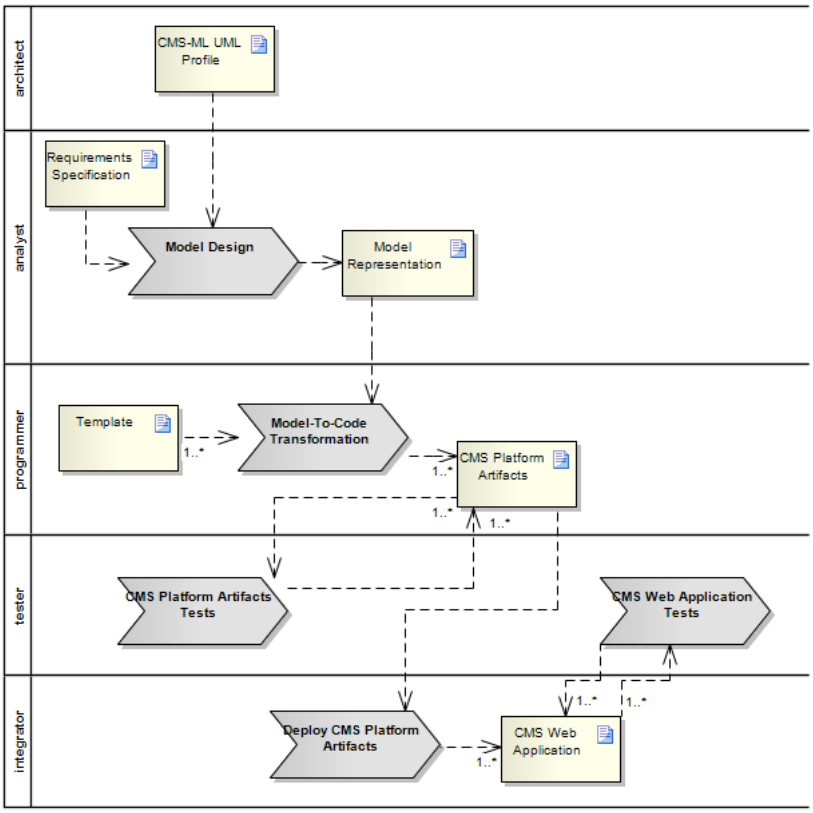


Figure 5.5: Project-Level Development Process Overview

in this dissertation. Many works exists in the requirements engineering field, I'll point to the work of David Ferreira [18], which is a good approach on requirements engineering.

In this work, we'll assume that requirements engineering is performed according to another approach and the analysts already have all the requirements formally defined and validated.

5.3.1 Model Design

The *Model Design* is the analyst main activity within the *Project-Level Development* process. In this activity, the analyst takes the requirements specification and defines a solution in the form an UML2 model with the CMS-ML Profile and the CASE tool.

To transform requirements into a working system, analysts must satisfy both customers and the system builders on the programmers' team, through a conceptual model. The customers understand what the system is to do. At the same time, the programmers must understand how the system is to work. For this reason, design is really a two-part iterative process. CMS-ML DSL allows the analyst to produce a conceptual design of the system that tells the customer exactly what the system will do and allows the programmers to understand the actual components needed to solve the customers' problem.

This activity is one of the most crucial parts in this work, because that is where CMS-ML language is used. Thus, this section will be longer in order to describe the main tasks undertaken in this activity.

The first task of this activity is to create the project and configuration for the MDDGenerator, as defined in section 4.5, and import the CMS-ML Profile. Then the analyst can proceed with the web application modeling. To better understand the modeling task, we shall give a "running" example (the simple example named by *SimExe*), which is very simple and with the single purpose to guide the model design activity description.

WebApplication View

A good start point is to develop the *WebApplication View*, it depends if the analyst is comfortable with this start.

To develop the *WebApplication View* the analyst must first create a package with the name *WebApplication View* and the correspondent diagram inside this package. In this diagram, the analyst can create: (1) the *Languages* and *VisualStyles* enumerations with the literals required to satisfy the clients requirements; (2) an UML class and apply the stereotype *WebApplication*, then set its tagged values according to the requirements; (3) finally, create three packages, for the three views, and the correspondent diagrams: the *WebPages View*, the *Roles View* and the *Users View*. Figure 5.6 shows the *WebApplication View* for the *SimExe*.

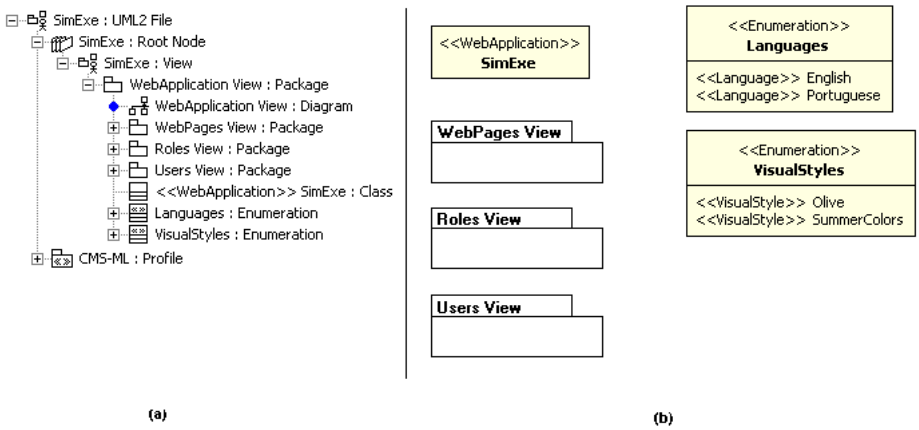


Figure 5.6: *SimExe*: (a) current project structure; (b) *WebApplication View*

All the enumeration literals (the ones of figure 5.6 and the ones defined in the following enumerations) are only an example, the analyst should create all the literals necessary to satisfy all the client requirements.

It's crucial to assure that all the literals have the right stereotype applications, because, for now, the ProjectIT-Studio doesn't proceeds with this type of validation, ProjectIT-Studio only allows to set a tagged value with a literal of the correct type.

The use of meaningless names on the enumeration literals helps maintaining the model platform independently, but it's recommended to use meaningful literals names that suggest some feature associated with the literal, never arresting them to specific platform concepts.

WebPages View

After the *WebApplication View*, the most logical and best path is to define the *WebPages View* and leave the *Roles View* and the *Users View* to the end.

The *WebPages View* is quite simple and consists of two other views: the *WebPages Structure View* and the *WebPages Hierarchy View*; and two enumerations: the *WebPageTypes* and the *WebComponentTypes*. This way, the analyst needs to create two packages with the views names and the correspondent diagrams and the two enumerations. Figure 5.7 shows an example of the *WebPages View*.

From the two sub-views of the *WebPages View*, we start by defining the *WebPages Structure View*, since it is not possible to define web pages hierarchies without first defining the web pages.

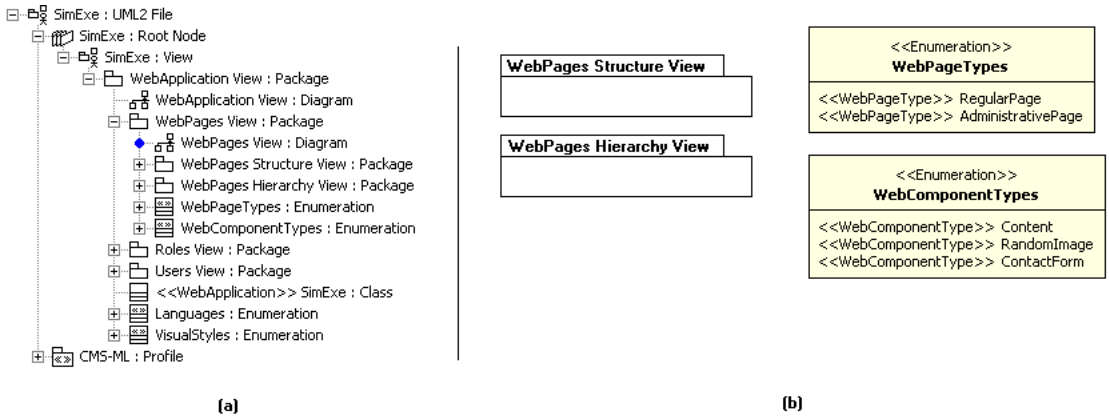


Figure 5.7: *SimExe*: (a) current project structure; (b) *WebPages View*

WebPages Structure View

The *WebPages Structure View* is where the analyst can define the containers to be used in the web pages definition. In the example in figure 5.8, two containers are defined: *SimExe_C1* and the *SimExe_C2*, both with the tagged value *order* set (see section 3.3.2 for more detail about the *Container* element).

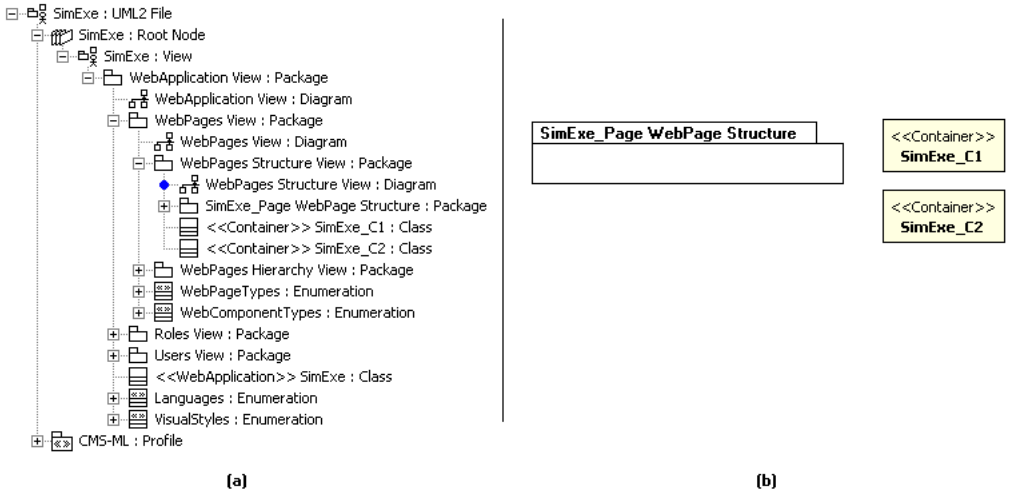


Figure 5.8: *SimExe*: (a) current project structure; (b) *WebPages Structure View*

Figure 5.8 show a package named *SimExe_Page WebPage Structure*, the analyst must create a package like this whenever a new page definition is required. So, to define a new web page, the analyst must create a new package inside the *WebPages Structure View*, it's advised to name the package according to the rule: *[WebPageName] WebPage Structure*, where the *[WebPageName]* must be replaced by the name of the web page. Notice that, this is not mandatory but highly recommended to maintain a coherent set of inter-related views.

Let's take for example the *SimExe_Page*. Inside the *SimExe_Page WebPage Structure*, the analyst must create an UML class element with the stereotype *WebPage* and, when required, define its tagged values (see figure 5.9). Note that all the containers defined in the *WebPages Structure View* are contained in the web page, in the correct order (defined with the *order* tagged value). The converter will always define an order according to the containers *order* tagged value, despise the displacement of the containers in the diagram.

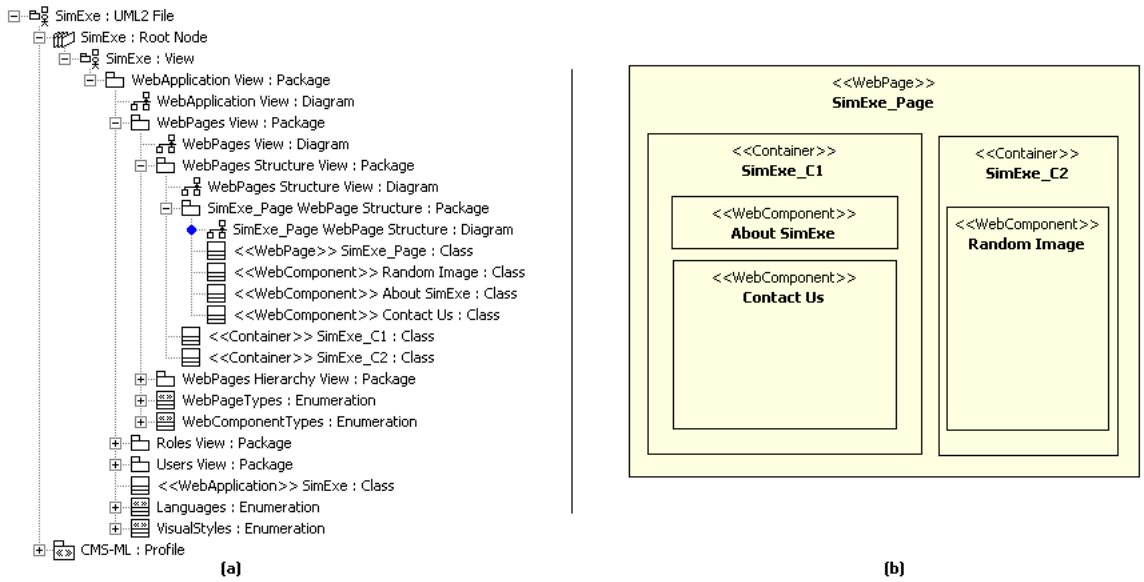


Figure 5.9: *SimExe*: (a) current project structure; (b) *SimExe_Page WebPage Structure*

The visual displacement of the elements in this view, exemplyfied in figure 5.9 (b), is crucial to define the containment implicit relationship. All the elements are completely contained in other elements, for example: both containers are completly contained in the web page. For an element to be contained inside another, its visual representation must have all vertices inside the visual representation of the element which must contain it. Example: the *Contact Us WebComponent* is contained inside the *SimExe_C1 Container*, which is contained inside the *SimExe_Page WebPage*; this is trivially understood as: the *Contact Us WebComponent* is instantiated in the *SimExe_C1 Container* of page *SimExe_Page WebPage*.

Figure 5.9 also shows the web components that are part of the web page. The web components must always have a defined type with one of the literals defined in the *WebComponentTypes*. In the example of figure 5.9, each web component *type* tagged value are: *Content*, *RandomImage* and *ContactForm*; respectively. The programmer will later specify, in the templates, the mapping of these web component types to platform components that **already exist in the target platform**.

Another displacement property is the *WebComponents* dispositions. A *WebComponent* has a vertical order inside a *Container*, for example: *WebComponents About SimExe* and *Contact Us* have order 1 and 2, respectively, inside the *SimExe_C1 Container* of *SimExe_Page WebPage*, this is automatically determined by the model converter, referred in section 4.5.1.

WebPages Hierarchy View

When all the *WebPages* elements are defined, the analyst can then proceed with the specification of the hierarchical relationships between web pages, through the *parent of* association (see section 3.3.3 for more detail). To do so, the analyst must "drag" all the *WebPages* elements and the *WebApplication* element into the *WebPages Hierarchy View* diagram. Then create directed associations between them with the stereotype *parent of*.

Figure 5.10 shows the web pages hierarchy for *SimExe*. The hierarchy is very simple, only one web page exits, the *SimExe_Page*, that has as parent the web application, this means that the *SimExe_Page* is at the highest level in the hierarchy.

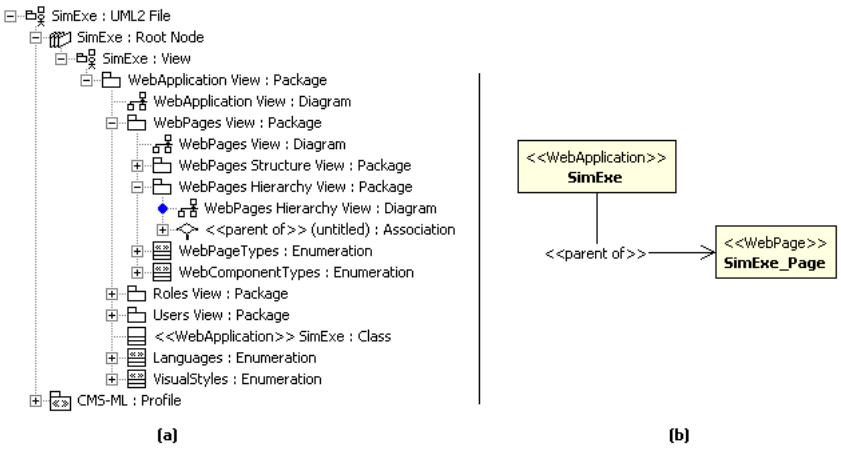


Figure 5.10: *SimExe*: (a) current project structure; (b) *WebPages Hierarchy View*

Roles View

Modeling the access control is much more complex than modeling the structural elements of the web application. All the roles and their permissions over the web application must be clearly identified by the analyst before begin modeling them.

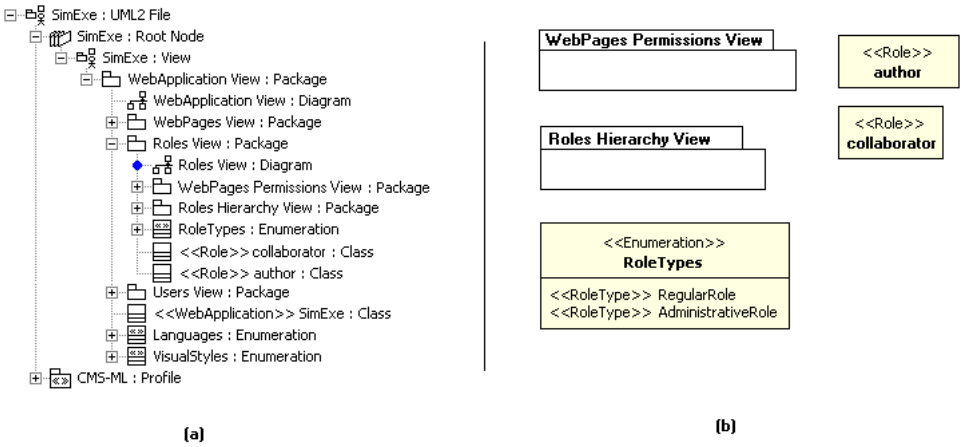


Figure 5.11: *SimExe*: (a) current project structure; (b) *Roles View*

Inside the *Roles View*, the analyst must create: (1) the *RoleTypes* enumeration with all the literals required to classify the roles; (2) an UML class element with the *Role* stereotype for each system role; and (3) two package views: the *Roles Hierarchy View* and the *WebPages Permissions View*. Figure 5.11 shows a *Roles View* example.

There are two roles in the *SimExe* web application, the "author" and the "collaborator", to define them, the analyst must create an UML class element with the stereotype *Role* then define all its tagged values (see section 3.3.2).

WebPages Permissions View

The *WebPages Permissions View* focus on the roles permissions over web pages and their web components. This way, the *WebPages Permissions View* contains two enumerations: the *WebPagePermissionType* and the *WebComponentPermissionTypes* with all the permission types for web pages and web

components, respectively.

It also contains a view for each web page that focuses on the permissions over a specific web page. The view has the name: *[WebPageName] WebPage Permissions View*, where the *[WebPageName]* is the name of a web page. Figure 5.12 illustrates the *WebPages Permssions View* for the *SimExe* example.

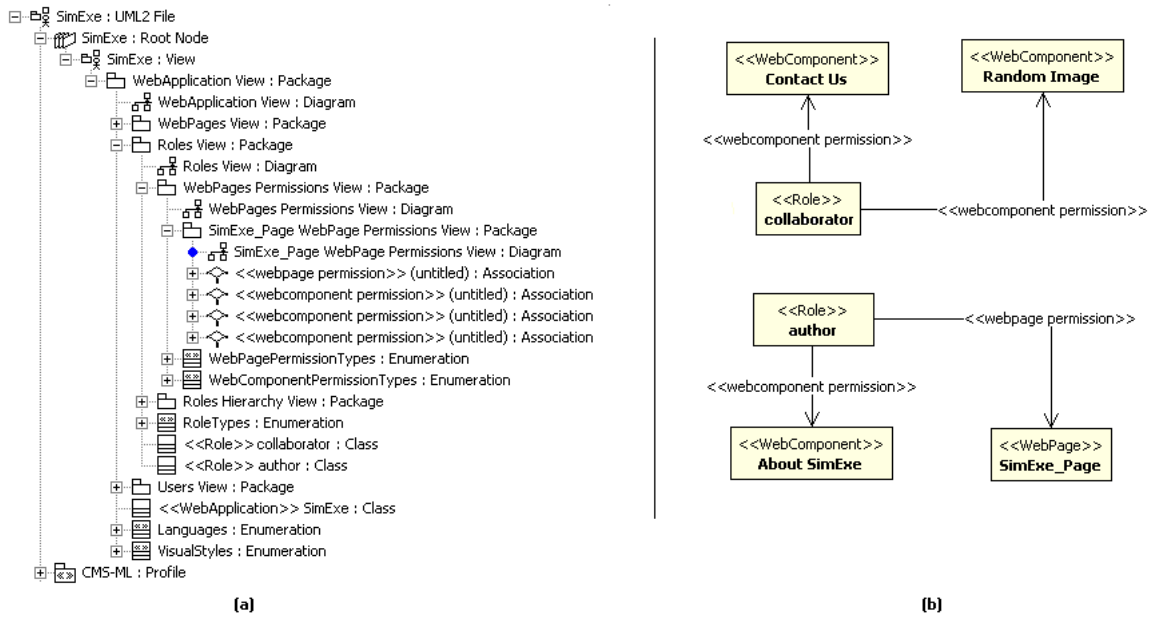


Figure 5.12: *SimExe*: (a) current project structure; (b) *WebPages Permissions View*

To build these web pages permissions views, the analyst needs to: (1) create the *[WebPageName] WebPage Permissions View* package and the correspondent diagrams for each web page, inside the *WebPages Permissions View*; (2) "drag" the roles, the correspondent web page and its web components elements to the *[WebPageName] WebPage Permissions View* diagram; (3) define a directed association between the role and the web page or web component, apply the *webpage permission* or *webcomponent permission* stereotype, accordingly to the target element, and define the correct permission type tagged value of the association (see section 3.3.3).

The biggest difficulty of the access definition lies with the permission type. The types vary dramatically from platform to platform, but the analyst has to abstract from the platform and produce a platform independent model. The problem lies on the templates that will transform the model concepts into platform artifacts. Each platform has a template, regardless of the templates that exist for other platforms, this may cause permissions inconsistency; for example: permission on a template of a platform X may not have the same meaning on the template of a platform Y. So there must be some mechanism to maintain consistency among templates. This issue is left to the developers' criterion, as it is extremely difficult to assure all the templates, for each platform, respect the same definition. In fact, consistency maintenance must be applied to all concepts of CMS-ML.

For now, this mechanism depends only on the analyst specification and the programmer should respect and implement this specification in the same way for each template.

The consistency problem is aggravated in the modeling of access permissions, mainly because it's where most of the CMS platforms greatly differ. So the model might have a role that in platform X has the permission to write on a page and the same role in the platform Y does not have the same permission over the same web page, the reasons for this ma be of various types, for example because the platform Y does not support permissions over web pages.

Roles Hierarchy View

The *Roles Hierarchy View* defines the hierarchical relationships between roles. To define these relationships, the analyst only needs to drag the roles, previously defined, to the *Roles Hierarchy View* diagram and then create a directed association with the stereotype *delegates* between two roles.

Figure 5.13 shows an example of a roles hierarchy. The "author" role will have all the permissions of the role "collaborator", plus some permissions defined for the "author" role.

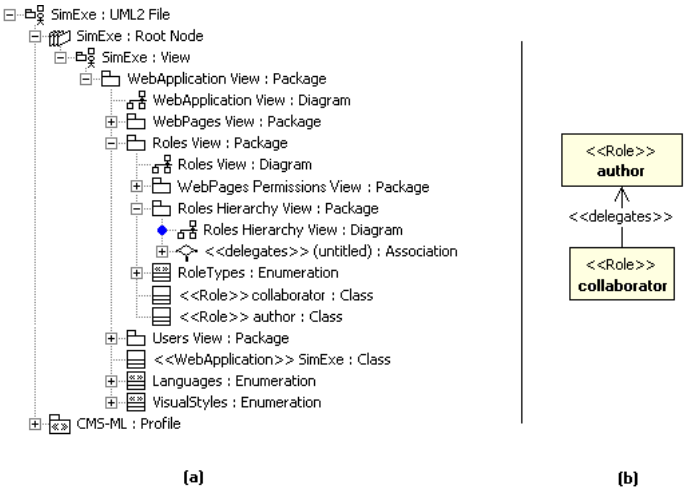


Figure 5.13: *SimExe*: (a) current project structure; (b) *Roles Hierarchy View*

Users View

Finally, the last view is the *Users View*. This view is very simple and boils down to create UML class elements, apply the *User* stereotype to the elements, define the users tagged values and make a directed association, with the stereotype *assigned*, from a *User* to a *Role*, as figure 5.14 illustrates.

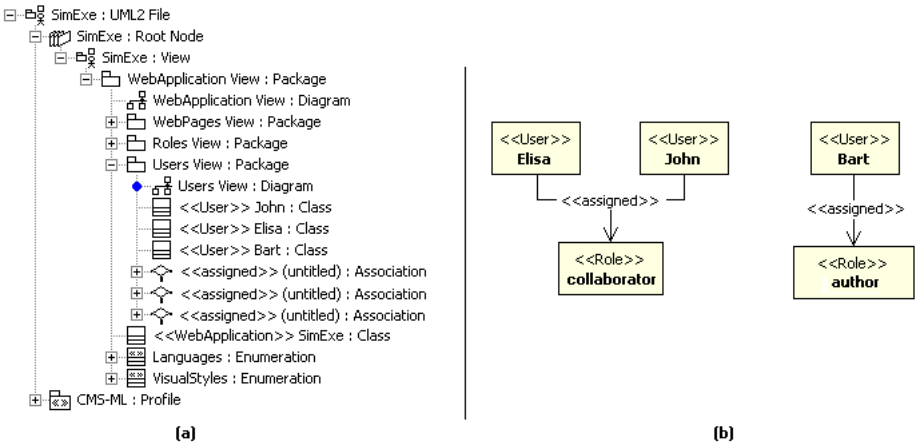


Figure 5.14: *SimExe*: (a) current project structure; (b) *Users View*

5.3.2 Model-To-Code Transformation and CMS Platform Artifacts Tests

The *Model-To-Code Transformation* is performed by the programmers and mainly includes tasks as developing web components to support new features and, if required, minor changes to the templates and

apply the transformation process. This activity receives the templates and a model representation and produces a set of platform artifacts (ex: code and documentation).

Almost all new projects require new components to include new features in the web application. This way, the programmers need to develop new web components for CMS platforms in the context of distinct projects, and change the templates so that the transformation process can instantiate and configure the new web components.

For example, in a project there might be the necessity for a web component to monitor the energy consumption of a server. This web component is highly specific and probably the template doesn't have the required information to instantiate it in a web page, nor does the CMS platform have this feature out-of-the-box. So the programmers must develop this component and deploy it in the CMS platform, then change the templates to support the new module. The model-to-code transformations define the required mechanisms to instantiate and configure the concepts, defined in the model that the current templates for the platform doesn't support.

Templates are defined in the *Platform-Level Development* and are intended to be reutilized with a specific platform, so they are as most generic as possible for a specific CMS platform. Although, no project is equal, this way the templates surely will not be enough to completely support all the requirements. The model-to-code transformation process can be viewed as an overlay over the templates, adding them support for more platform specific features and project details, necessary to satisfy the requirements.

Like the web components, languages specifications and visual styles may be required for each new project; so the programmers or designers must develop these elements and change the templates to include them.

The *CMS Platform Artifacts Tests* submits the CMS platform artifacts, produced by the *Model-To-Code Transformation*, to a set of quality assurance tests and may produce the some changes in the CMS platform artifacts.

5.3.3 Deploy CMS Platform Artifacts and CMS Web Application Tests

Deploying CMS artifacts in a CMS platform can take several forms and is directed related with the platform access way chosen by the analysts, in the *CMS Platform Analysis* activity.

Usually deploying the articles resumes to: (1) run the artifacts code that will use the platform API to create the model elements in the platform; or (2) execute artifacts SQL scripts that will access directly the platform database and create the elements.

After a successfully deployment, the resulting web application needs to be inputted to a quality assurance activity, the *CMS Web Application Tests*. This activity is the last activity of the *Project-Level Development* process, so it must assure that the web application is fully aligned with the clients' requirements and with no errors nor failures.

5.4 Summary

This chapter presented the two processes: (1) one for the development of new platform specific templates, and therefore, that aims to extend the portability of the CMS-ML for other CMS platforms; and (2) a second for the development of CMS-based web applications, using the CMS-ML language and the templates.

This chapter, started with an introduction of the development processes of the approach. Then the actors for the development using the CMS-ML approach where presented along with the two developments process: the *Platform-Level Development* and the *Project-Level Development* process.

A detailed description of each process activities where given, along with simple examples.

Chapter 6

Evaluation and Case Study

In the last three chapters a concrete proposal of a DSL for CMS-based web applications was given, along with a proposal for the development approach using the CMS-ML DSL. The purpose of this chapter is to complement the thesis validation that was given along the work, by describing one application of the proposal made in this dissertation, a fictional CMS-based web application: the *MyWebHome*.

The work described in this dissertation and its application to the case study are, actually, quite intertwined, as they have occurred in parallel, influencing each other. The web application was deployed in two distinct CMS platforms: *Joomla!* and *Drupal*; to test and evaluate the CMS-ML language benefits, features and capabilities.

The case study was fully developed using the CASE tool ProjectIT-Studio (section 4.1) with the extension to support the CMS-ML language. It may be seen as a reference case study, developed for testing and evaluation of this thesis work, namely: (1) the CMS-ML language; (2) the ProjectIT-Studio *CMSMLMetaModel* module; and (3) the templates for the CMS Platforms (i.e., Joomla! and Drupal).

This chapter is divided according to: (1) brief description of the web application requirements; (2) the overview of the correspondent CMS-ML model for the web application; (3) the result and conclusions for the case study.

6.1 CMS-ML Evaluation: *MyWebHome* Case Study

MyWebHome is a CMS-based web application designed to manage various contents and features of a personal web site. It is purposely limited regarding to the structure and sectioning, the type of content of each page and the modules used.

This section begins with the requirements description, and then the requirements are transformed into an UML model with the CMS-ML Profile. The model that represents the web application is discussed along with the difficulties and the case study conclusions.

6.1.1 Requirements

The *MyWebHome* web application must present the following requirements:

- A home page, with a welcome message and the most recent created contents;
- A projects page for the ongoing, finished and future projects. This main page should list all the projects, a photography of the projects team and a link for the projects. Each project should have a more detailed page;

- An academic page where all the academic relevant work is described, along with a list of related links. This page should provide list other related pages for the LEIC and MEIC courses. Each sub-page should detail the correspondent course;
- A page for various contents related to personal interests. This page should show a picture of the authors and a list of interesting links. It's also desired a list of movies, art, music and leisure activities. Additionally, this web page should provide a link to a special interest: photography;
- A photography page with a catalog to manage images;
- A page for a description "about me" and a form to send email to the site administrator;

The web application must also provide mechanisms for access control and content manipulation, which are presented in the following requirements:

- The web application must provide a login module for registered users. A registered user can login and logout with the module;
- There must be an administrator user to control the entire site;
- Anonymous users can only read the entire web application, except the administration panel;
- A collaborator user can read the entire web application and write in the projects page and in each specific project page;
- Initially, there must the following users: author fracar, collaborators raumes and joacar;

6.1.2 CMS-ML Model

The *MyWebHome* model was designed in order to support the requirements presented in the previous section. The model already presents the CMS-ML UML Profile's stereotypes applied to the UML elements. This section shows the model macro view for the *MyWebHome* web application, using the approach proposed in this dissertation. A more detailed model description is presented in appendix A, which illustrates all the elements, associations, views and tagged values; along with the results for each platform.

The model macro view it's split by figures 6.1 and 6.3. All the views defined are in accordance with section 3.3.4, where they are specified. An explanation for each of the views defined in this case study is given, in order to show how the proposal behaves with a practical case.

The figure 6.1 presents the macro view of the ***WebApplication View*** with two enumerations: the *Languages* and the *VisualStyles*. The *Languages* enumeration contains the languages that can be applied to the web application or to a specific web page, this case study will only use the English language, this is defined as a tagged value in the *WebApplication* element. The *VisualStyles* enumeration contains three visual styles, each style with a non-descriptive name. When a visual style is used, it will be mapped to a platform specific visual style.

The *WebApplication View* also contains the *WebApplication* class element, with the name of the web application: *MyWebHome*. This element is trivially created and requires a set of tagged values, which are detailed in appendix A.

The ***WebPages View***, illustrated in figure 6.1, contains two enumerations: the *WebPageTypes* and the *WebComponentTypes*. The *WebPageTypes* enumeration contains two types: regular web page and administrative web page; both can be applied to web pages. The analyst can also add more web page types, if the requirements demands. The *WebComponentTypes* enumeration contains all the web component types required to specify the web component, note that each *WebComponentType* is closely

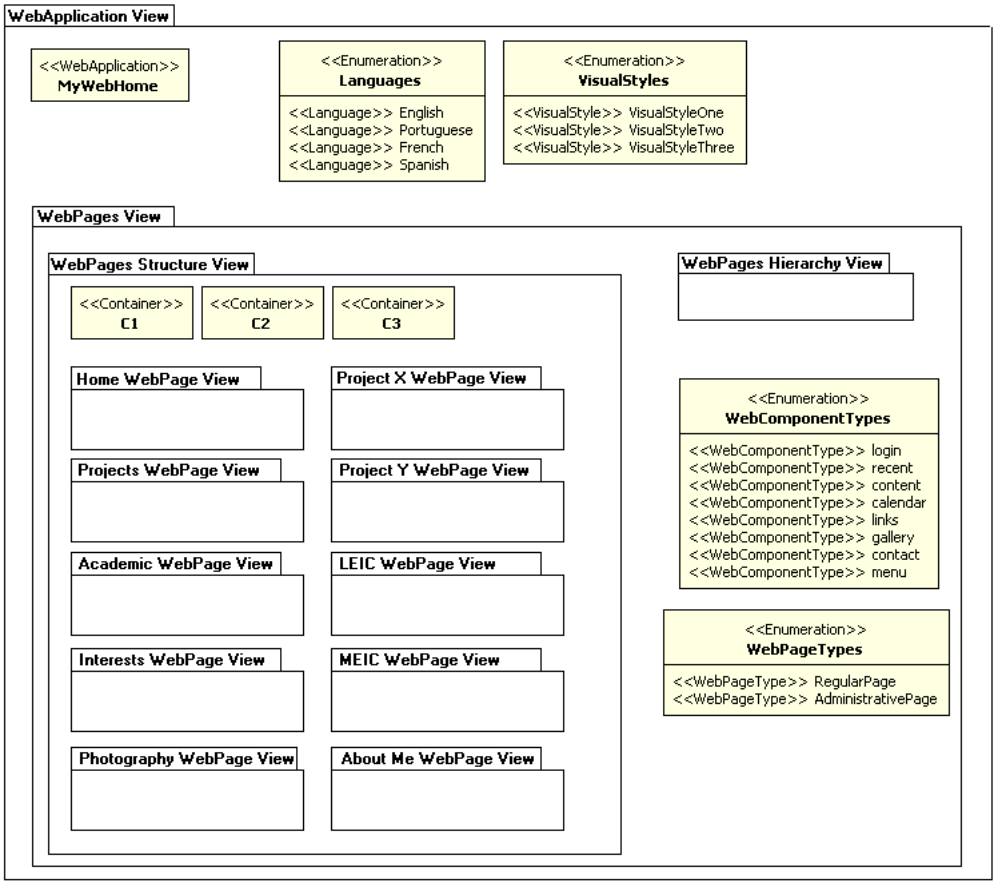


Figure 6.1: *MyWebHome* Model Macro View

related with the web component functionality. The web component types are specified according to the desired application features and mapped to some platform feature.

The *WebPages View* also contains two view: the *WebPages Hierarchy View* and the *WebPages Structure View*.

The ***WebPages Hierarchy View*** which contains the hierarchical relationships between the web pages defined in the structure view.

The ***WebPages Structure View*** contains three *Containers*: C1, C2 and C3; these contains are closely related to the visual style used by the web application. In this case study, it's assumed that the used visual styles support the three containers dispositions.

The *WebPages Structure View* includes several other views, one view per *WebPage* element, as defined in section 3.3.4. The *WebPage Structure View* is one of the most important views because it's where the analyst defines the page composition and features. For instance, the *Projects WebPage* illustrated in figure 6.2 is composed by three logical spaces, the containers: the left container is empty; the middle container has three *WebComponents* with type *content* (not shown here for space reasons but detailed in appendix A) and with a certain content text; and the right container has a project team image which is a content *WebComponent* and has another *WebComponent* with type *links* that presents a list of hypertext links (defined in the content tagged value). Note that all the *WebComponent* types associated with these *WebComponents* are part of the *WebComponentTypes* enumeration and therefore are of type *WebComponentType*, otherwise the association to the tagged values was not possible.

Besides the *WebPages View*, the *WebApplication View* contains two other main views: the *Roles View* and the *Users View*; illustrated in figure 6.3.

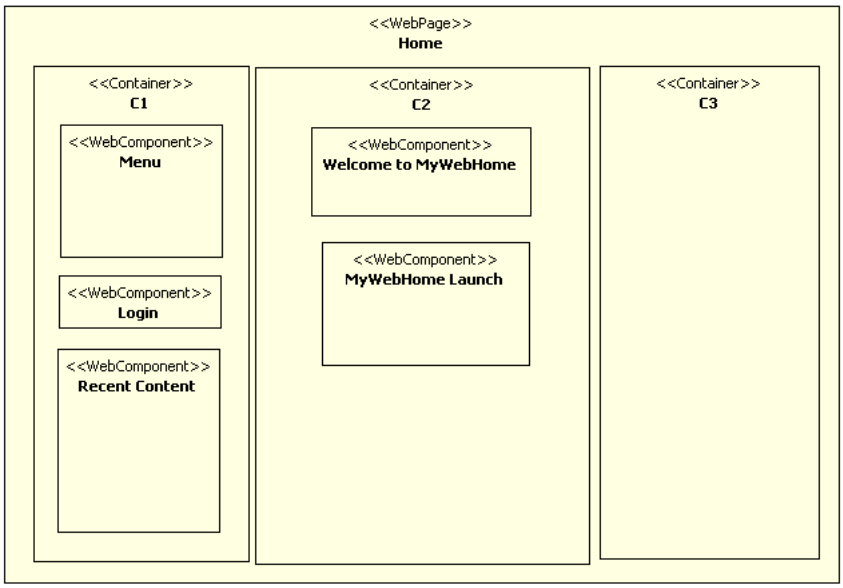


Figure 6.2: *MyWebHome Projects WebPage Structure*

The ***Roles View*** contains the *RoleTypes* enumeration with two types: system role and local role; both can be applied to roles. This view also aggregates all the class elements with the stereotype *Role*: the author role and the collaborator role; both trivially defined. The ***Roles Hierarchy View*** also belongs to the *Roles View* and contains the delegation relationships between the roles.

The ***WebPages Permissions View*** contains two enumerations: the *WebPagePermissionTypes* and the *WebComponentPermissionTypes*. Both enumerations contains a set of permissions for web pages or web components, desired for this web application. Besides the two permissions enumerations, this view also contains permissions view for each web page, that are not illustrated in figure 6.3 for size reasons. The web pages permissions view is further detailed in appendix A.

Finally, the *WebApplication View* contains the ***Users View*** where all the application users are defined, i.e., UML class elements with the stereotype *User*. This view also shows the assignment relationships among users and roles.

For further details and additional information about the model for the *MyWebHome* web application, the reader should consult the appendix A.

6.1.3 Conclusions

The *MyWebHome* requirements and model were developed independently of the underlying platforms, as expected. This way, no greater difficulties was presented in these two phases. The model is designed only with conceptual elements and in a great level of abstraction.

The biggest challenges in the modeling process was in some of the concepts associated with a specific CMS platform, that can be very specific and sometimes not directly supported by the CMS-ML. Also, validate and test the resulting web application on a specific platform, can present some difficulties, namely in the validation of the model alignment with the web application.

The two underlying CMS platforms used in this case study had no effect on modeling task, which shows that the modeling task, using the CMS-ML, is platform independent.

A fact which is evidenced in the modeling task with the CMS-ML language is that this task focuses on the requirements and web application design, irrespective of the platforms, as intended.

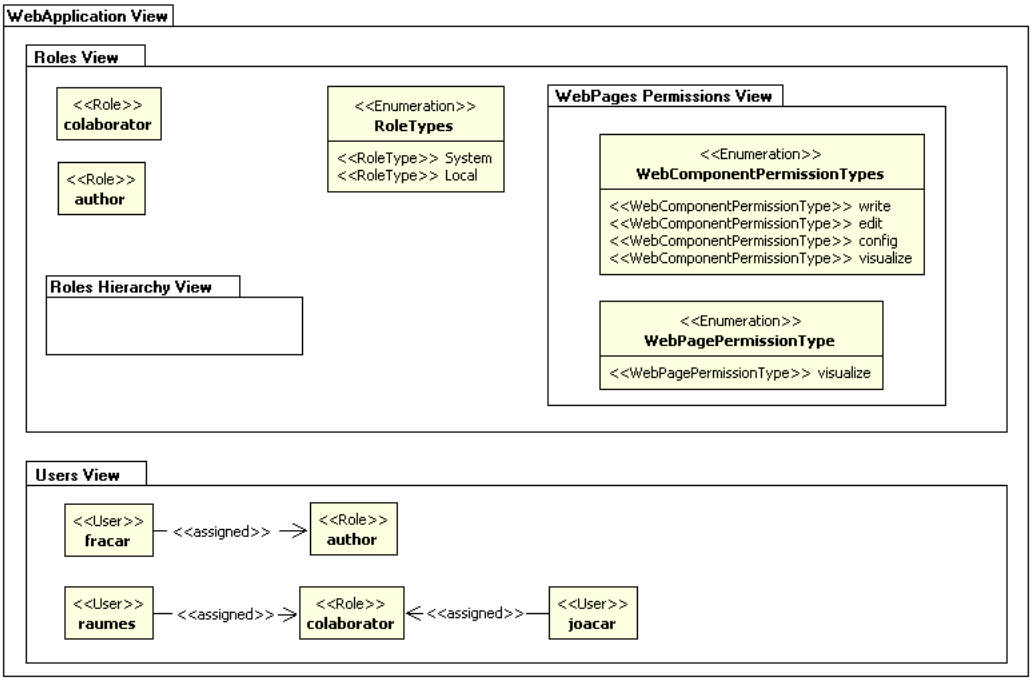


Figure 6.3: *MyWebHome* Model Macro View

From the *MyWebHome* case study, the reader can confirm that the language usage is very straightforward, all the elements are unambiguous and the views enable a good separation of concerns and facilitate the development.

Summarizing, **modeling** with CMS-ML can bring some challenges; most of the effort and time focuses in the requirements and how to satisfy them, not in how to represent them in the model, nor in the model design.

6.2 Transformation and Tool Support Evaluation

The biggest challenge in the proposed approach, and therefore, where more difficulties arise, is to obtain detailed knowledge of the target platform: (1) how to map the CMS-ML concepts to specific platform artifacts; (2) how to access the target platform (API versus database access); and (3) do the platform has all the components/modules required for the web application.

The *MyWebHome* model was transformed into two distinct CMS platforms: *Joomla!* and *Drupal*.

Transforming the model to a web application in ***Joomla!*** presented the following characteristics: (1) the best way to access Joomla platform was through the database, this means that the model transformation process produces SQL script files; (2) the biggest difficulty was to find Joomla modules to satisfy the requirements, most of them were open source modules available in internet sites; (3) the permissions mechanisms of Joomla are very simple and not so flexible as modeled by the analyst, this means that the templates transformed the complex permissions of *MyWebHome* to a more simple and Joomla oriented permissions definition; and (4) there weren't Joomla concepts that weren't supported by the CMS-ML, some of them required the use of the configuration string tagged value to define configuration parameters for modules.

Drupal CMS platform involved much more difficulties than Joomla, mainly because the Drupal has specific concepts, not standard in the most CMSs platforms, and due to the authors' platform knowledge.

The model transformation for ***Drupal*** presented the following characteristics: (1) to contrast with

the Joomla access, in Drupal the templates used the platform API and also to exemplify that it's possible to use more than one way to access the target platform; (2) finding the right modules to satisfy the requirements were also a problem; (3) the permissions mechanisms in Drupal are more flexible than the Joomla! permissions, but not flexible enough to enable a direct mapping between the model and the platform; this way, the template transformed the complex permissions of *MyWebHome* to a more simple and Drupal oriented permissions definition; and (4) some Drupal concepts aren't supported by the CMS-ML, like taxonomies, modeling the taxonomy concept is possible by using a special type of web component to represent the concept or by extending the language with new elements.

The developments done in ProjectIT-Studio tool in order to support the CMS-ML approach was very simple and straightforward, reviling the high extensibility of the tool for new features. Although the CASE tool can be improved to: (1) automatically generate all the CMS-ML views, when a new project is created; and (2) automatically create a new *[WebPageName] WebPage Structure View* and *[WebPageName] WebPage Permissions View* package when a new *WebPage* is created.

Some modeling improvements features in ProjectIT-Studio can bring high model design performance gains, by reducing repetitive tasks, such as views creations for each project. This and other desired features are identified in section 7.2.

Summarizing, modeling and then transform the model to a platform can bring some challenges, most of the effort and time focuses in the templates development, not in the model design. The templates development can be a very complex task, depending of on the platform complexity and on the developers' platform knowledge. Resuming chapter 5, templates are developed in the *Platform-Level Developement* process and reused many times in the *Project-Level Development* process, which attenuates the templates development difficulty.

6.3 Summary

This chapter presented the application of the proposal made in this dissertation to a fictional simple web application - the *MyWebHome* web application. The result obtained with this application show the effectiveness of the proposal made in this dissertation, the complexity related to the development of a CMS-based web applications and the adequacy of the proposal for real-world usage.

Chapter 7

Conclusions

This concluding chapter presents the main contributions of this dissertation and discusses some of the new research avenues that this work opens.

7.1 Main Contributions

The primary goal of the work presented in this dissertation was to simplify the development and maintenance of CMS-based web applications and make them more portable. Moreover, that this should be accomplished in a developer friendly way - that is, in such a way that it is not only easily comprehensible by an average developer, but also that it is practical to use in current web application development.

The proposal achieves this goal, because throughout this dissertation, an analyzes and identification of some of the problems and benefits with the existing approaches and technologies is presented, in chapter 2. Then a proposal of a solution to those problems is given, and the solution is validated through its application to a case study. So, the achievement of this goal is the primary contribution of this dissertation.

More specifically, the proposal introduced a graphical language for CMS-based web applications modeling, in a platform independent form. Using this language, developers can develop web applications without knowing specific platforms or frameworks. Also a proposal of a development methodology for CMS-based applications was described, it's composed of a set of inter-related activities that exchange artifacts with the goal to produce a requirements-aligned CMS-based web application, using a model driven approach.

7.2 Future Work

Even though the work described in this dissertation is self-contained, in the sense that it may be readily used without further developments, it does not constitute, quite naturally, the ultimate solution for any of the problems that it addresses. Rather on the contrary, the decision to make proposals that are simple to learn and to implement makes these proposals particularly susceptible to being extensible in many various ways.

In fact, we believe that it should be a goal of any research work to open up new research directions. Therefore, we hope that others, as we intend ourselves, will follow-up on the work described in this dissertation.

In particular, we envision that the following topics, presented in no particular order, will be pursued further in the future:

- Perhaps the most important point for future work is the web components modeling. Actually the CMS-ML allows the development of CMS-based web applications, with web components that already are implemented and deployed in the target CMS platform. It's highly desirable to extend the language in order to enable the modeling of web components. To do so, new views must be created, a view for a new web component, where it will be defined. Also new language elements must be defined, such as: text box, button, label, link, and more. This point is by its own a subject of great work and of great interest.

- Another very useful feature is the tool capability to monitor the modeling and generate new packages for new elements. During the web application modeling, some repetitive tasks exist, that affects the analysts' productivity and can become very exhaustive. This way, the tool can be extended in order to automatically generate new elements that are mandatory in a certain cases. A crucial case is when a new *WebPage* element is defined, posteriorly, a new view must be created for the permissions of this web page, this task can be automated.

- The model serialization functionalities should also be improved. Although there is nothing wrong with the current strategy (save a model to an XML file, and use the correct deserializer to load a model from an XML file), it forces a model serialization and posterior deserializaion. This is very useful for testing purposes, but introduces an unnecessary step in the process, which doesn't append advantages to the final web application. By the contrary, serialize and deserialize the model every time the developer needs to test a minor change in the model, can be a very exhaustive task.

- Implement in the future is the tool support for the validation of the CMS-ML models based on the language specification. The strategy currently planned for the implementation of this feature requires a continuous validation of the current model according to the CMS-ML specification. Obviously, this feature should have the lowest priority possible, so that the continuous validation of the model does not affect the performance of the ProjectIT-Studio's user-interface. The validation includes: (1) CMS-ML profile validation; (2) association member ends are according to the language specification; and (3) CMS-ML elements tagged values correctness; among others.

- The specification of "model-to-model" transformation, specifically "pim-to-psm" transformations. Although currently, the work proposes a "pim-to-artifacts" transformation and is effective in this task, it would be desirable to define a mechanism that provides a set of "model-to-model" transformation primitives, which could then be used in the specification of "model-to-model" transformations at a high-level, with any level of complexity. The inclusion of "pim-to-psm" transformation should not exclude the "pim-to-artifact" transformation; it's desirable to provide the developer with the option to choose the model transformation, if "pim-to-psm" then "psm-to-artifact", or only "pim-to-artifact".

- The maintenance of CMS-based web applications is another highly desired and interesting characteristic for the proposal future. In order to allow the maintenance of web applications, using the CMS-ML approach, it's required to extend the CMS-ML transformation process in order for it to become a round-trip transformation process, currently it's only a forward transformation process. This future work point will allow the development of CMS-based web applications with the CMS-ML, as proposed in this dissertation, and also to build the CMS-ML model through an already existing CMS-based web application - automatically build the model through a CMS-based web application, i.e., "artifacts-to-PIM" transformation.

These topics are, by no means, an exhaustive list of all the research work that may follow from what is described in this dissertation. Rather, it is a list of some of the topics on which we had already some

thoughts, and with which this dissertation will finish.

Appendix A

The *MyWebHome* Case Study

This appendix presents: (1) a detailed illustration of the model views that represent the *MyWebHome* web application; and (2) the result of the model transformation on two distinct platforms: (1) Joomla and (2) Drupal.

This appendix appears as an extension of Chapter 6 and should be used after reading that chapter.

A.1 MyWebHome CMS-ML Model

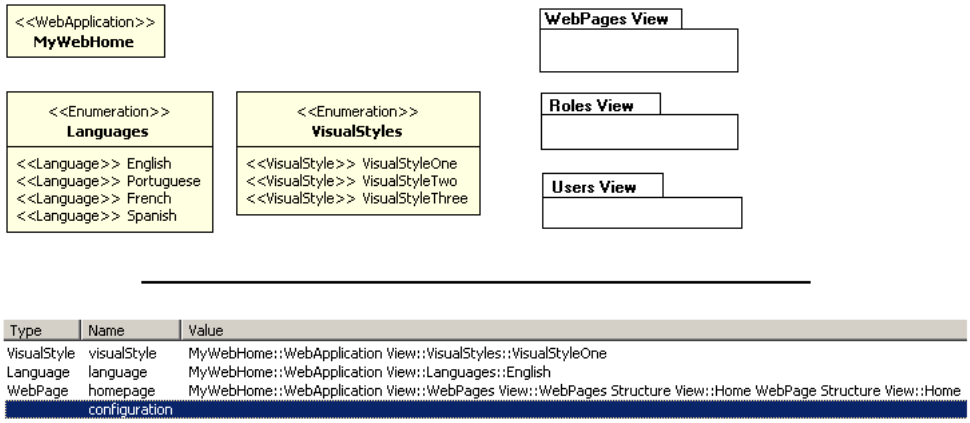


Figure A.1: (top) *WebApplication View*; (bottom) *MyWebHome WebApplication* tagged values

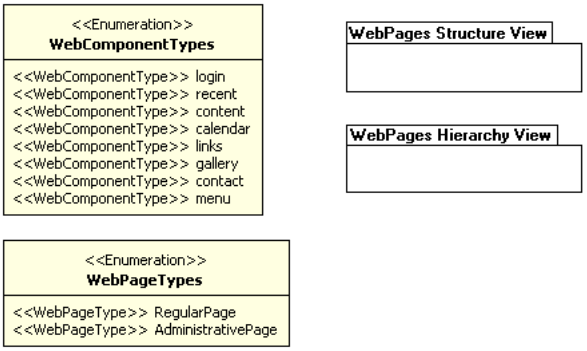


Figure A.2: *WebPages View*

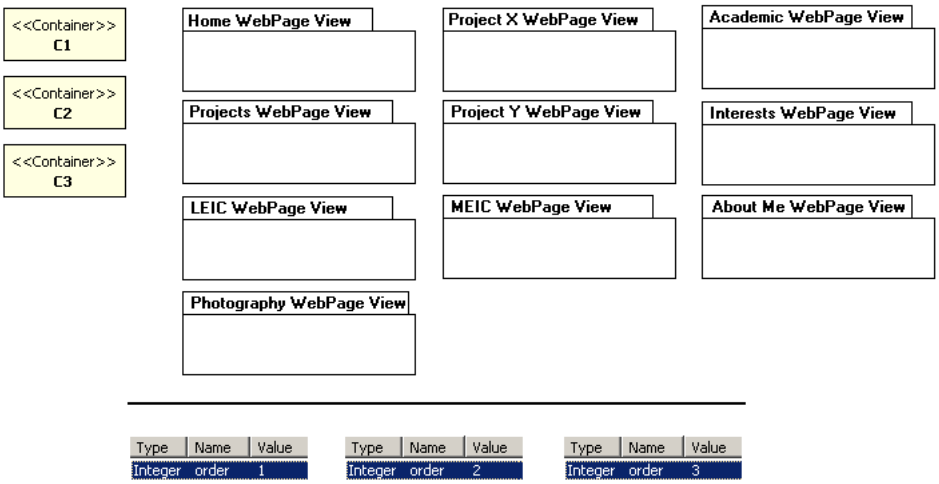


Figure A.3: (top) *WebPages Structure View*; (bottom left) *C1 Container* tagged values; (bottom center) *C2 Container* tagged values; (bottom right) *C3 Container* tagged values

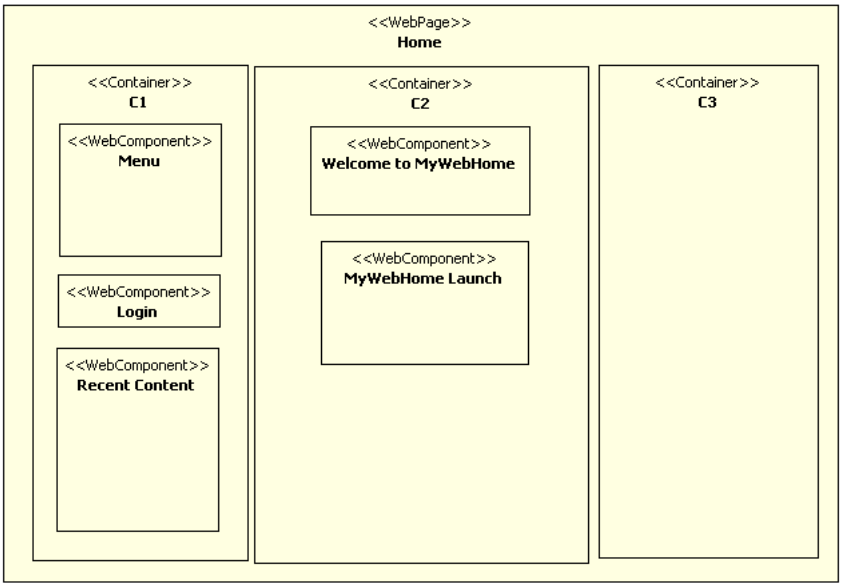


Figure A.4: *Home WebPage View*

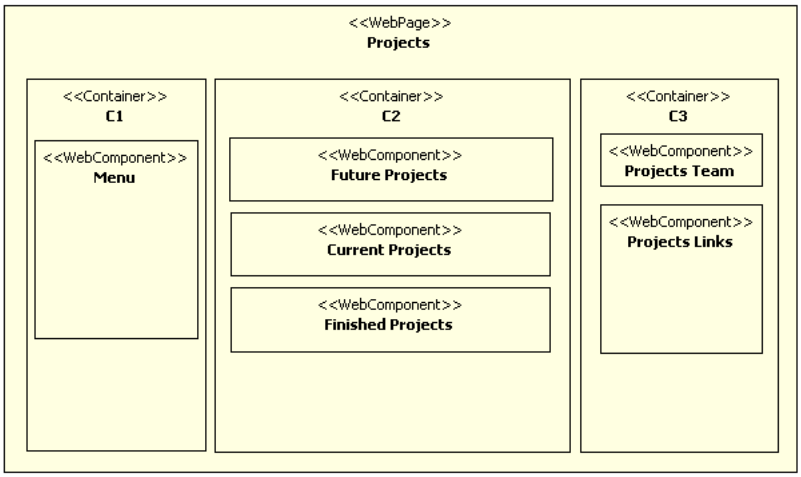


Figure A.5: *Projects WebPage View*

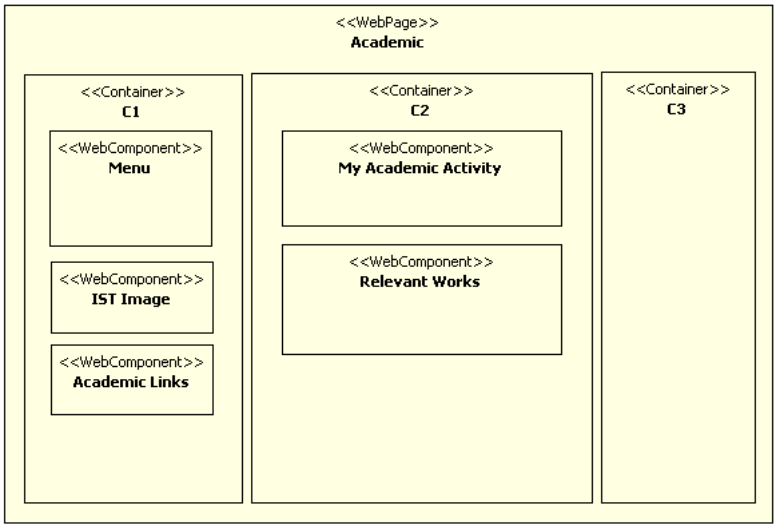


Figure A.6: *Academic WebPage View*

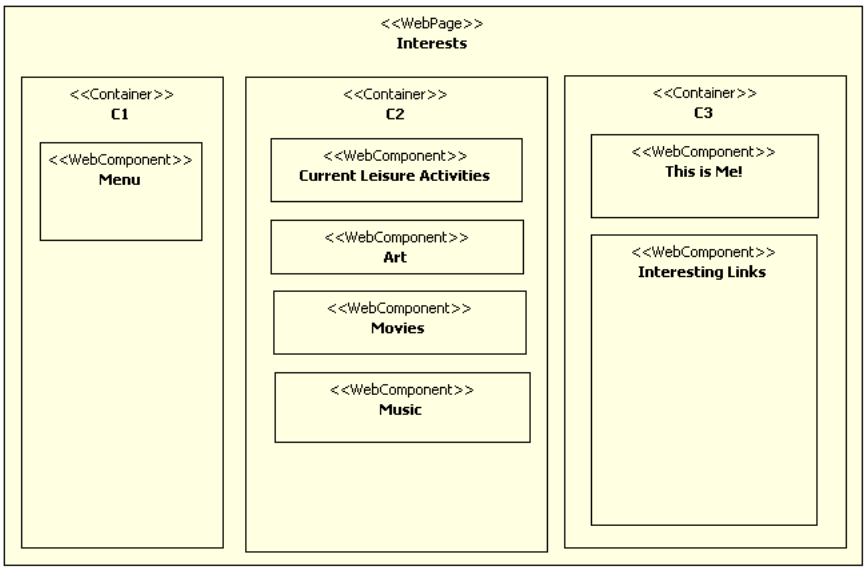


Figure A.7: *Interests WebPage View*

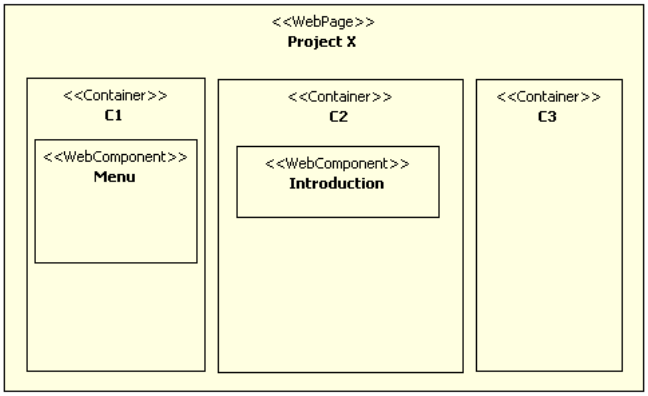


Figure A.8: *Project X WebPage View*

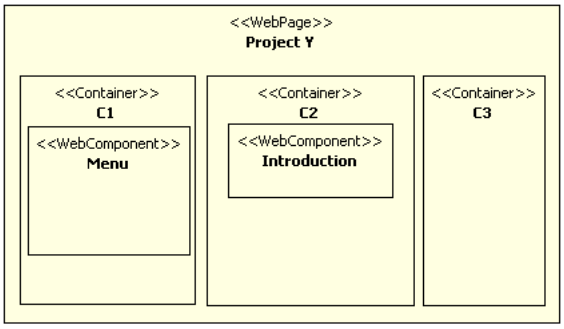


Figure A.9: *Project Y WebPage View*

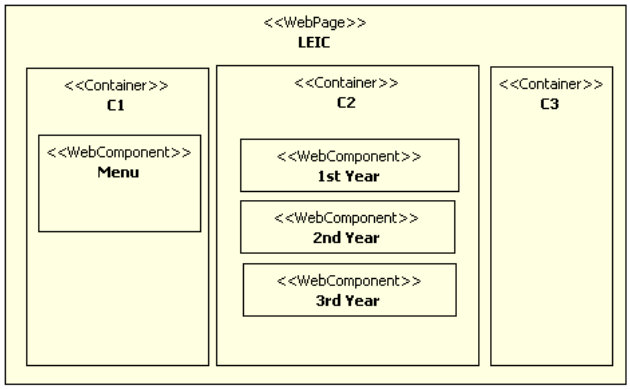


Figure A.10: *LEIC WebPage View*

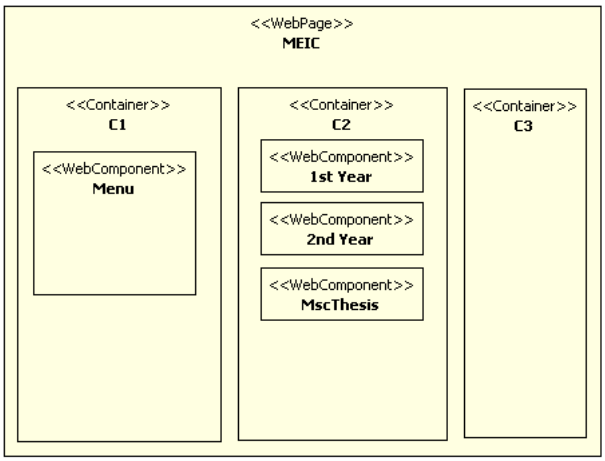


Figure A.11: *MEIC WebPage View*

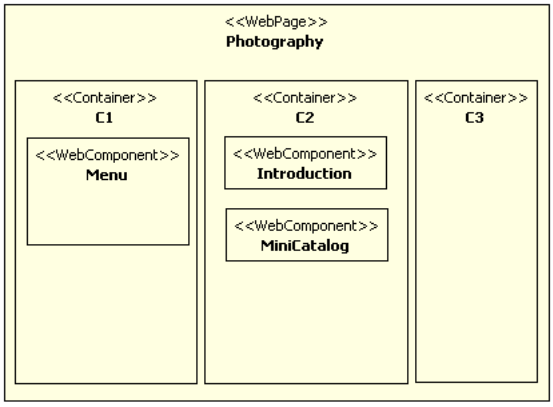


Figure A.12: *Photography WebPage View*

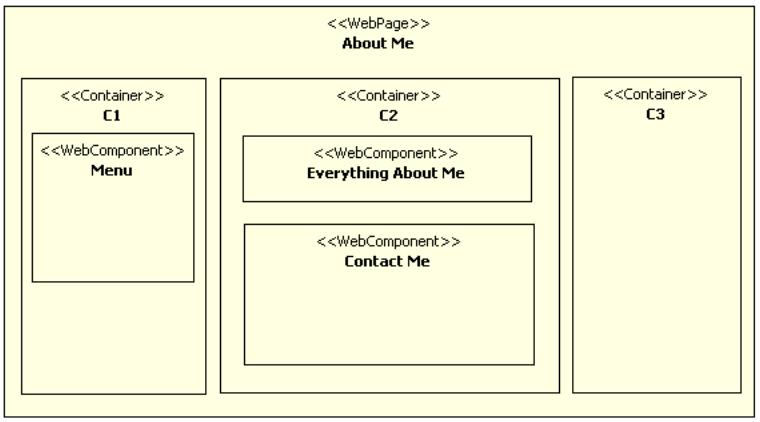


Figure A.13: *About Me WebPage View*

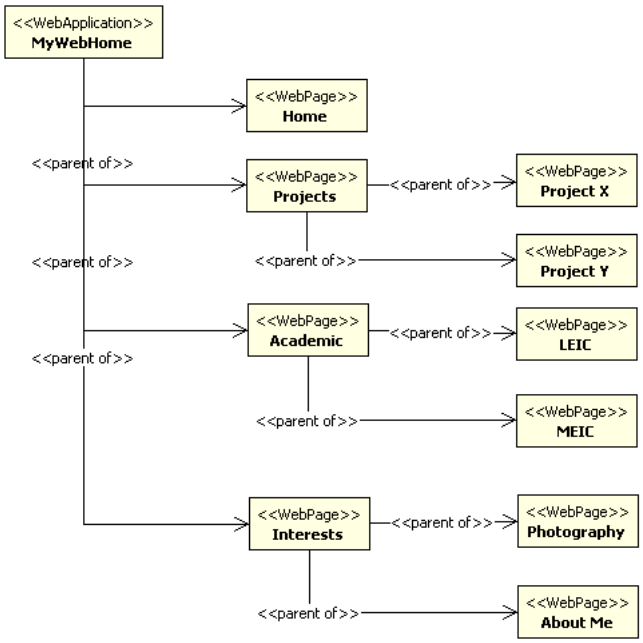


Figure A.14: *WebPages Hierarchy View*

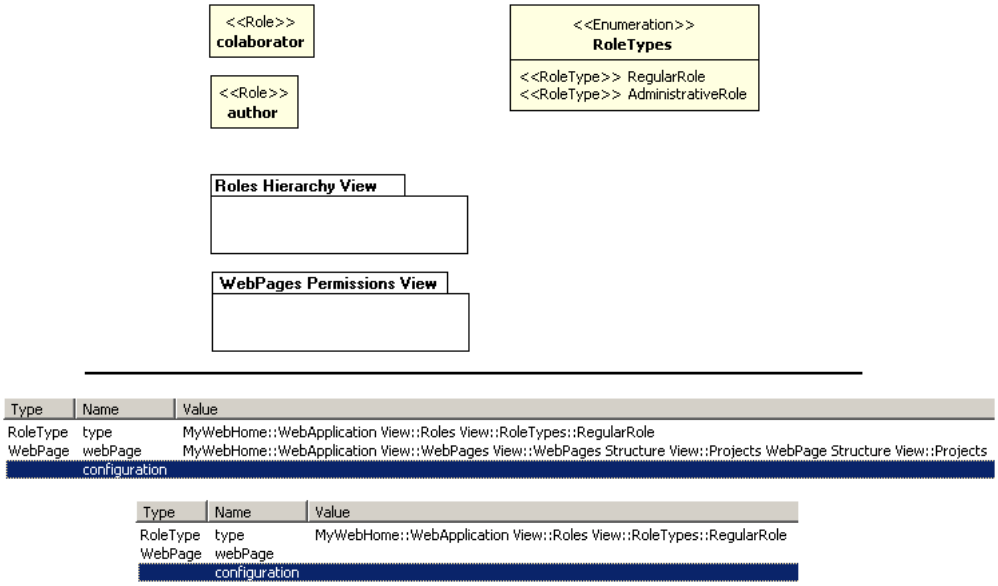


Figure A.15: (top) *Roles View*; (bottom top) *collaborator Role* tagged values, collaborator role is local to the *Projects* web page; (bottom bottom) *author Role* tagged values

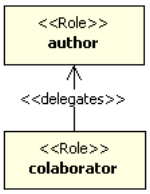


Figure A.16: *Roles Hierarchy View*

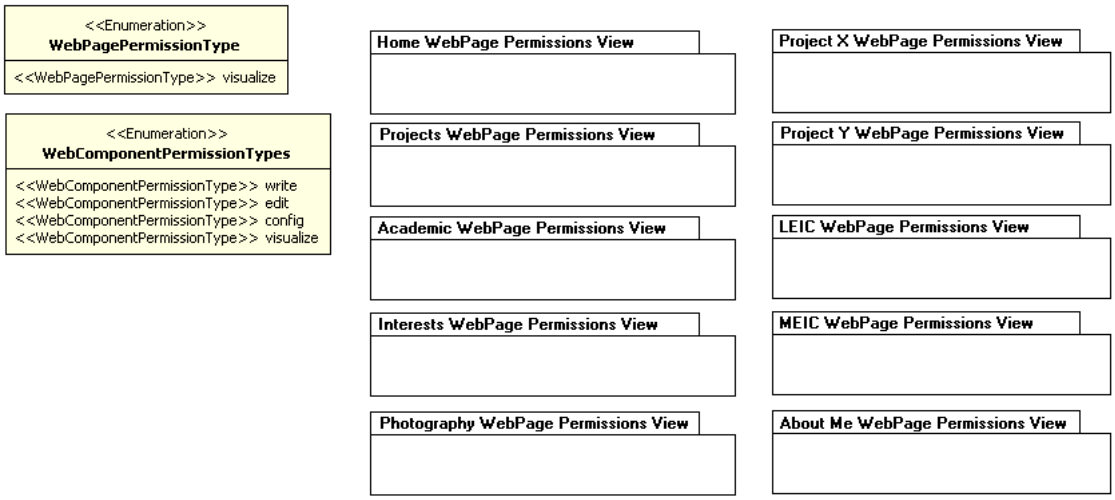
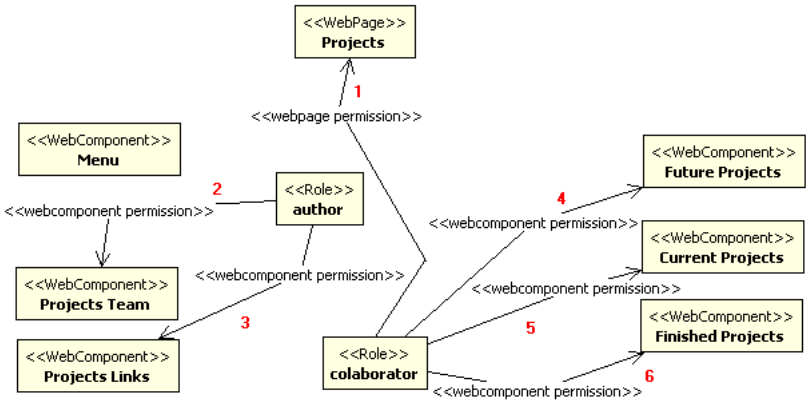


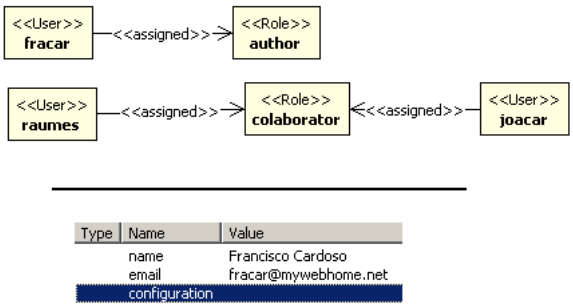
Figure A.17: *WebPages Permissions View*



Type	Name	Value
WebPagePermissionType	type	MyWebHome::WebApplication View::Roles View::WebPages Permissions View::WebPagePermissionType::visualize

Type	Name	Value
WebComponentPermissionType	type	MyWebHome::WebApplication View::Roles View::WebPages Permissions View::WebComponentPermissionTypes::write

Figure A.18: (top) *Projects WebPage Permissions View*; (bottom top) webpage permission type for association 1; (bottom bottom) webcomponent permission for associations 2 to 6



Type	Name	Value
	name	Francisco Cardoso
	email	fracar@mywebhome.net
	configuration	

Figure A.19: (top) *Users View*; (bottom) *fracar User* tagged values

A.2 *MyWebHome* on Joomla

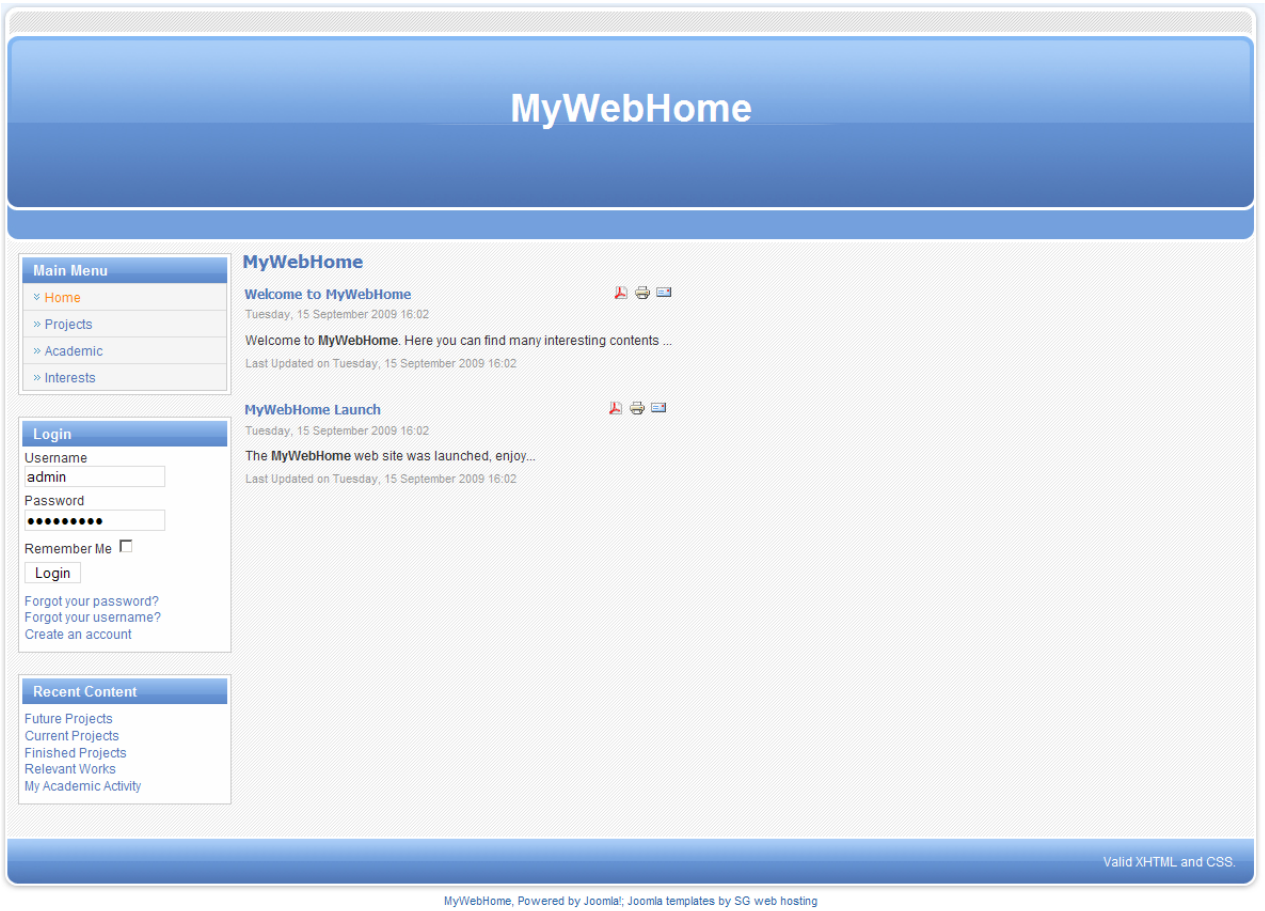


Figure A.20: *Home* web page on joomla - figure A.4

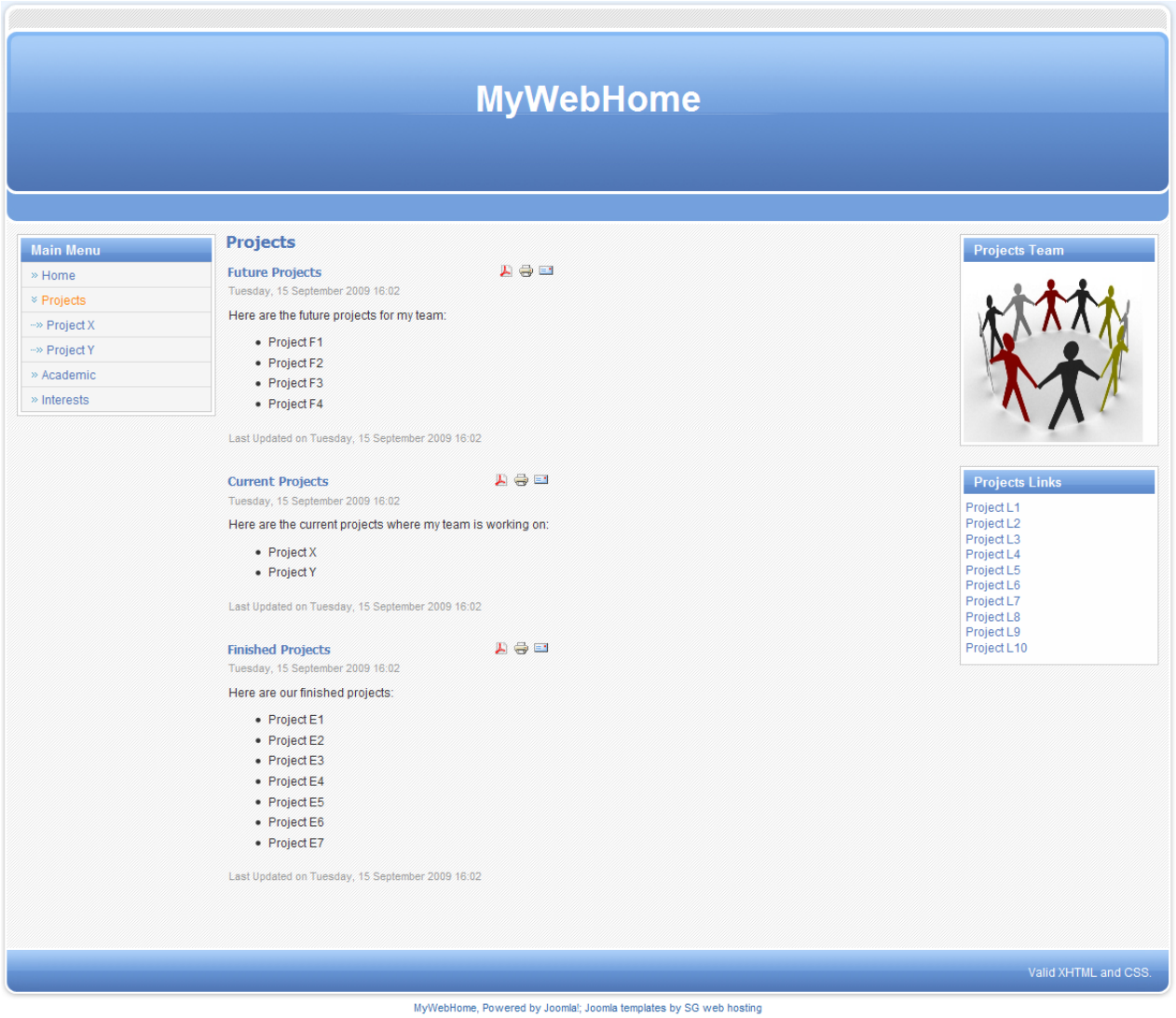


Figure A.21: *Projects* web page on joomla - figure A.5

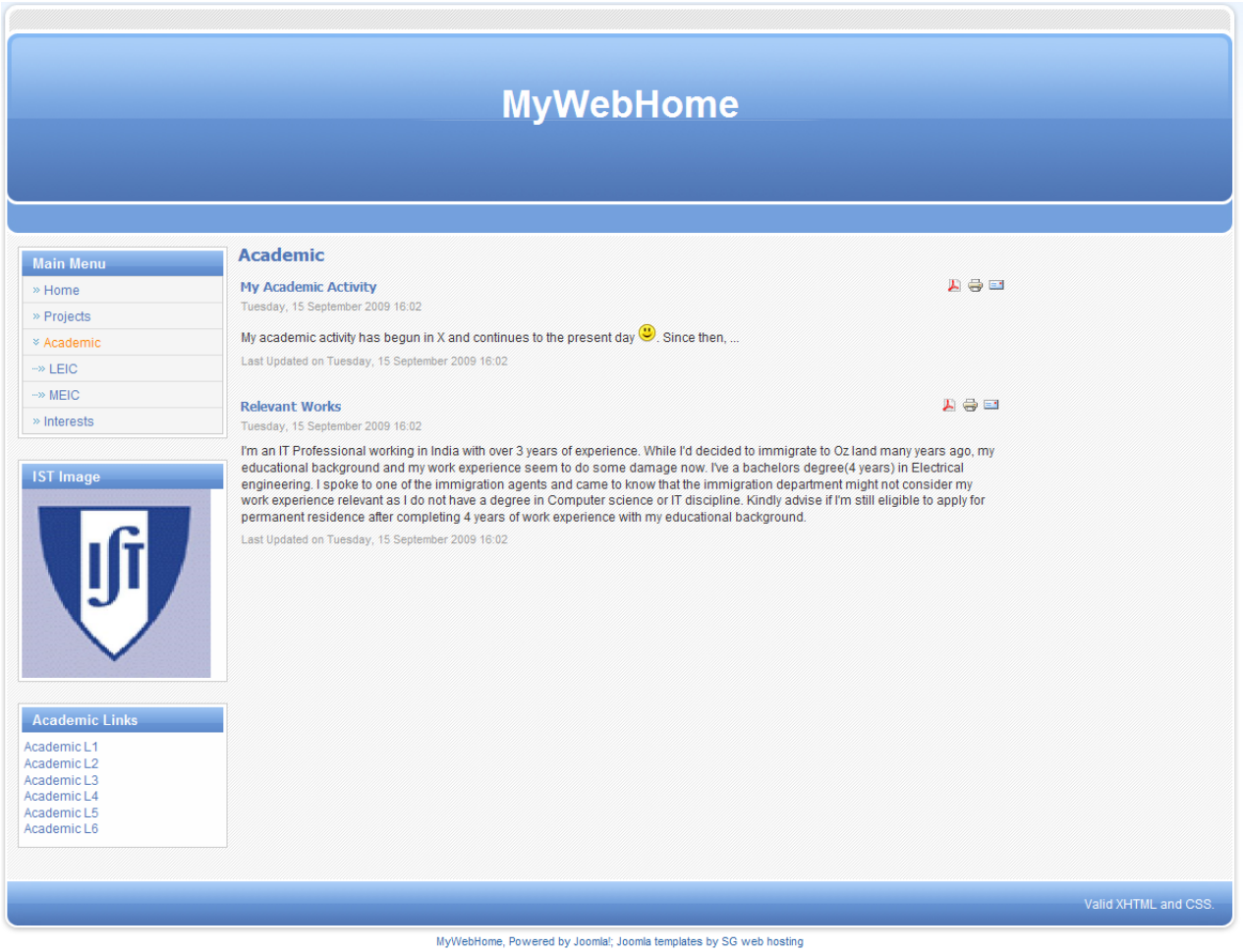


Figure A.22: *Academic* web page on joomla - figure A.6

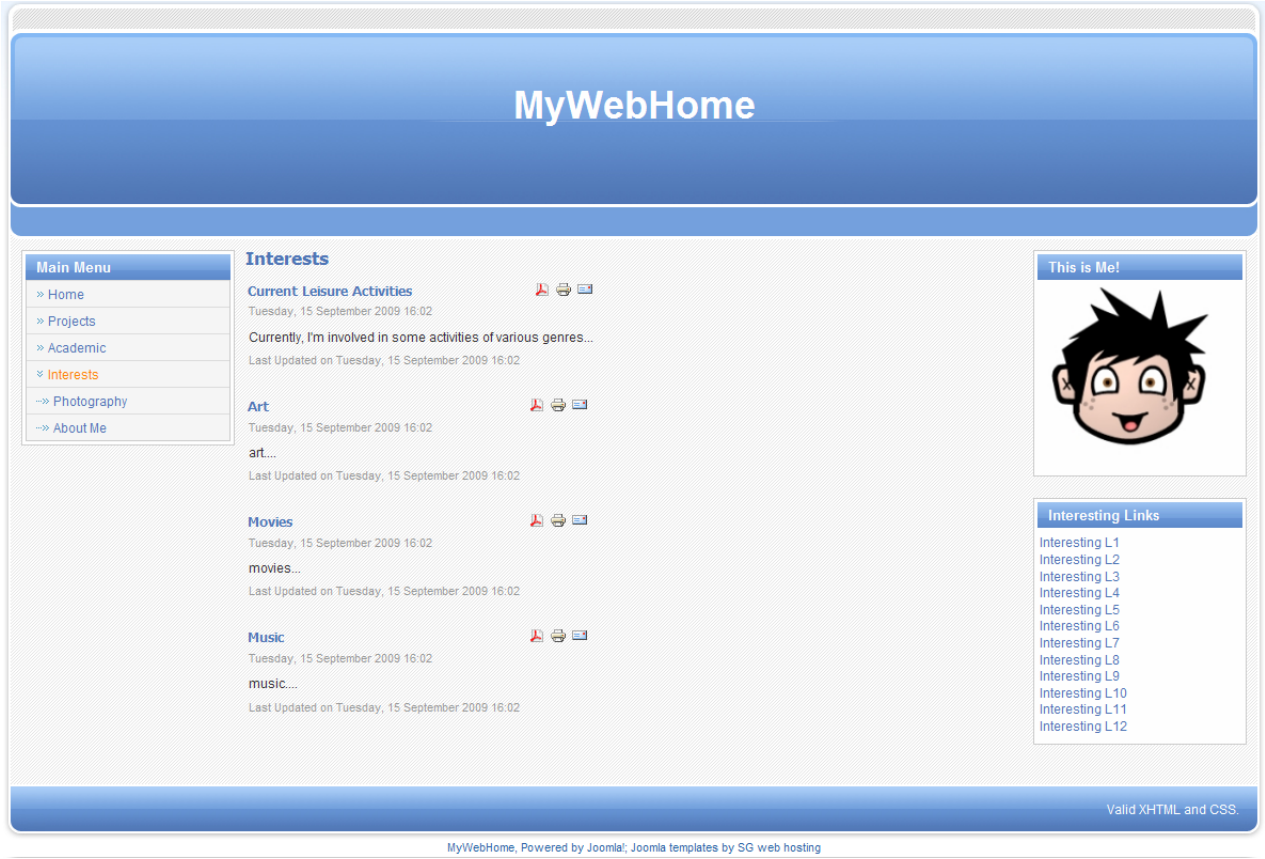


Figure A.23: *Interests* web page on joomla - figure A.7

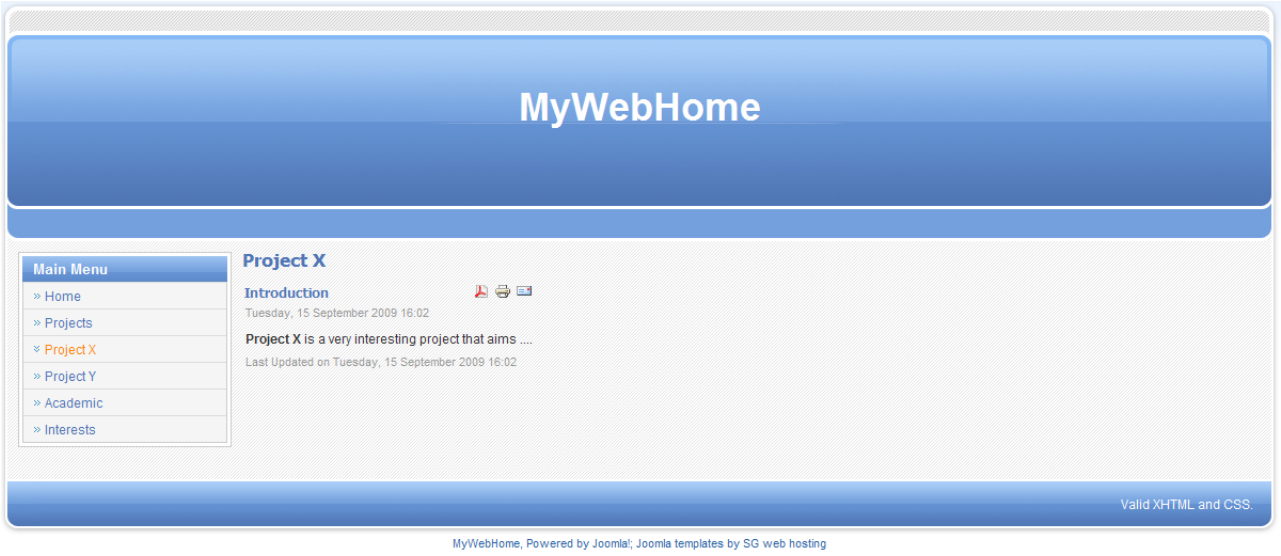


Figure A.24: *Project X* web page on joomla - figure A.8

Figure A.25: *Project Y* web page on joomla - figure A.9

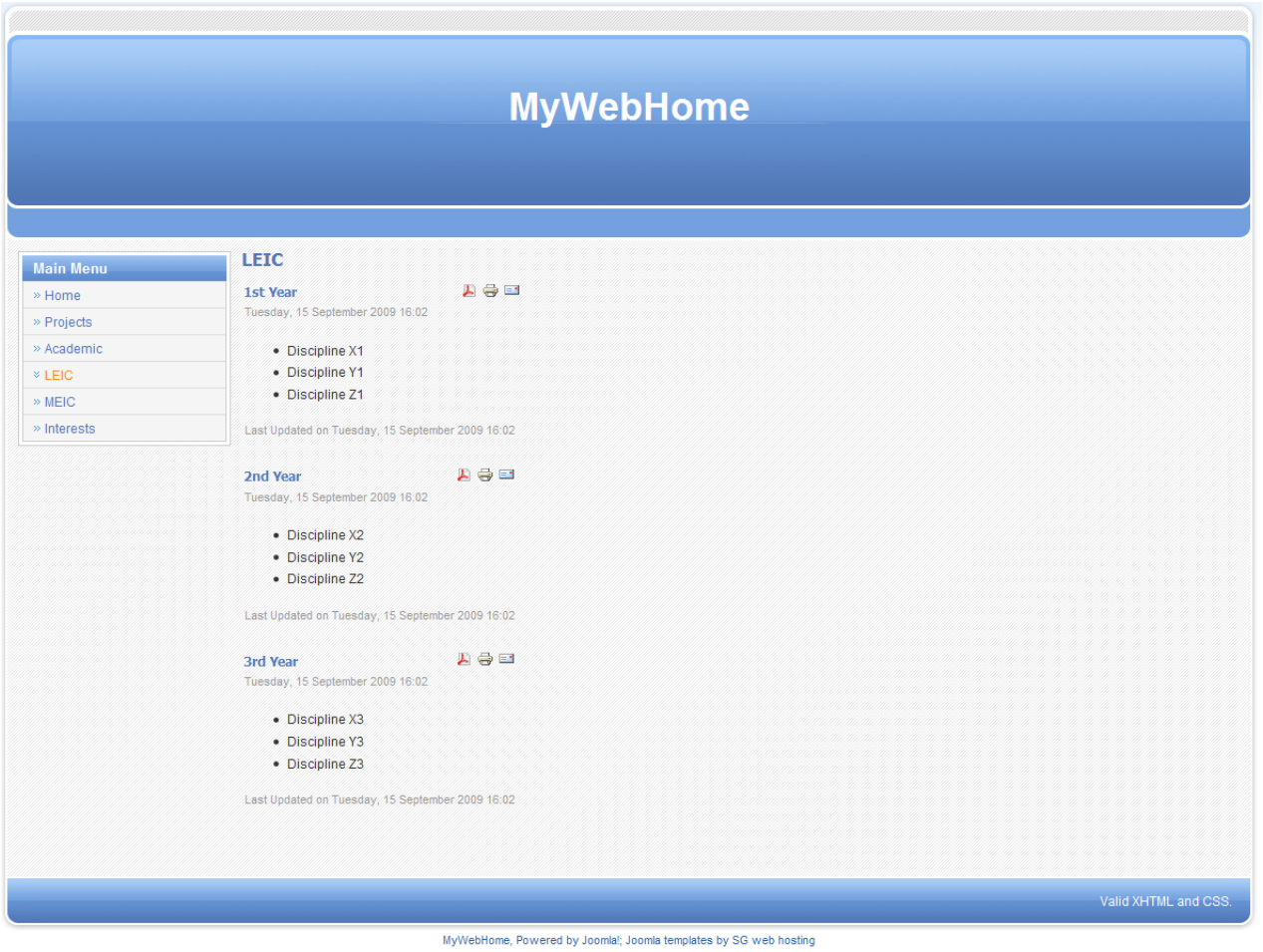


Figure A.26: *LEIC* web page on joomla - figure A.10

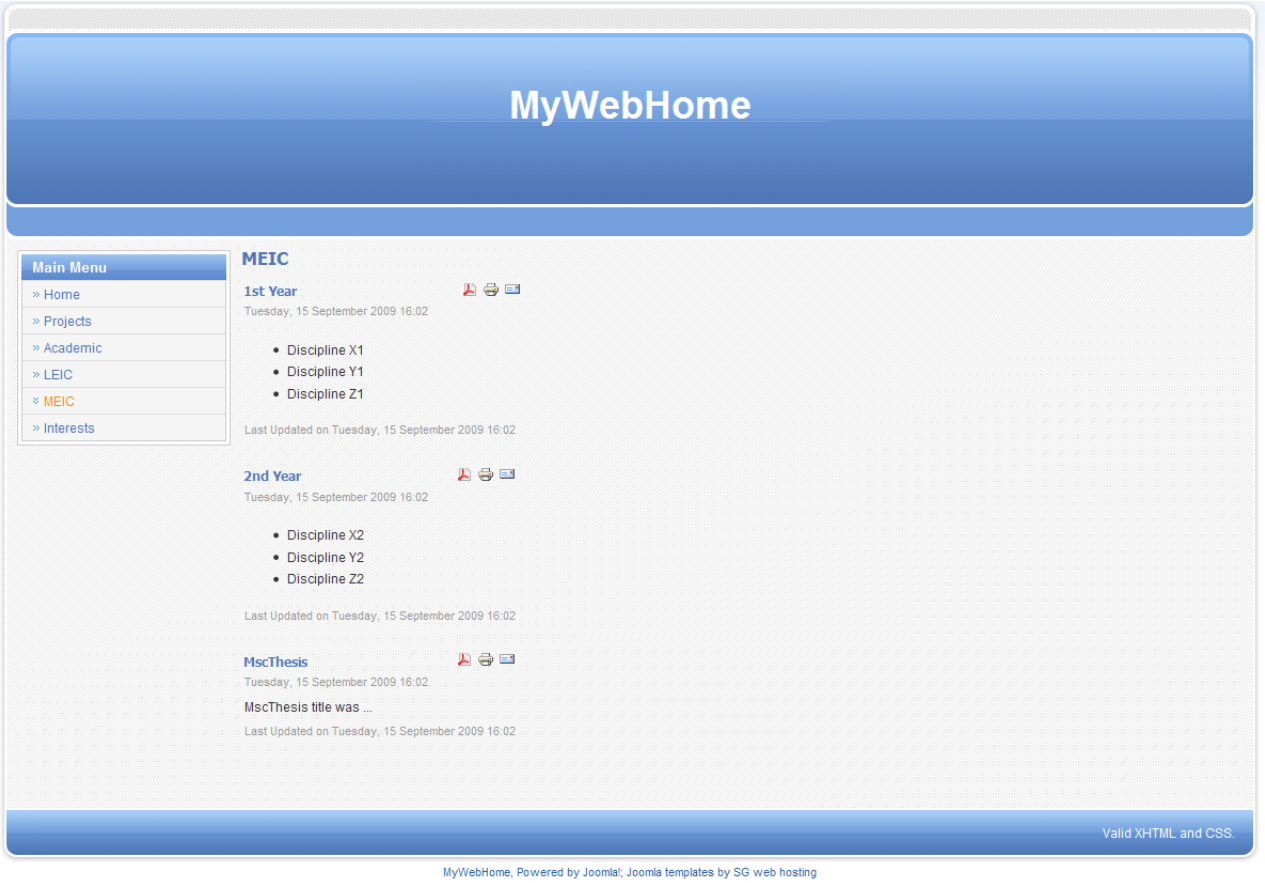


Figure A.27: *MEIC* web page on joomla - figure A.11

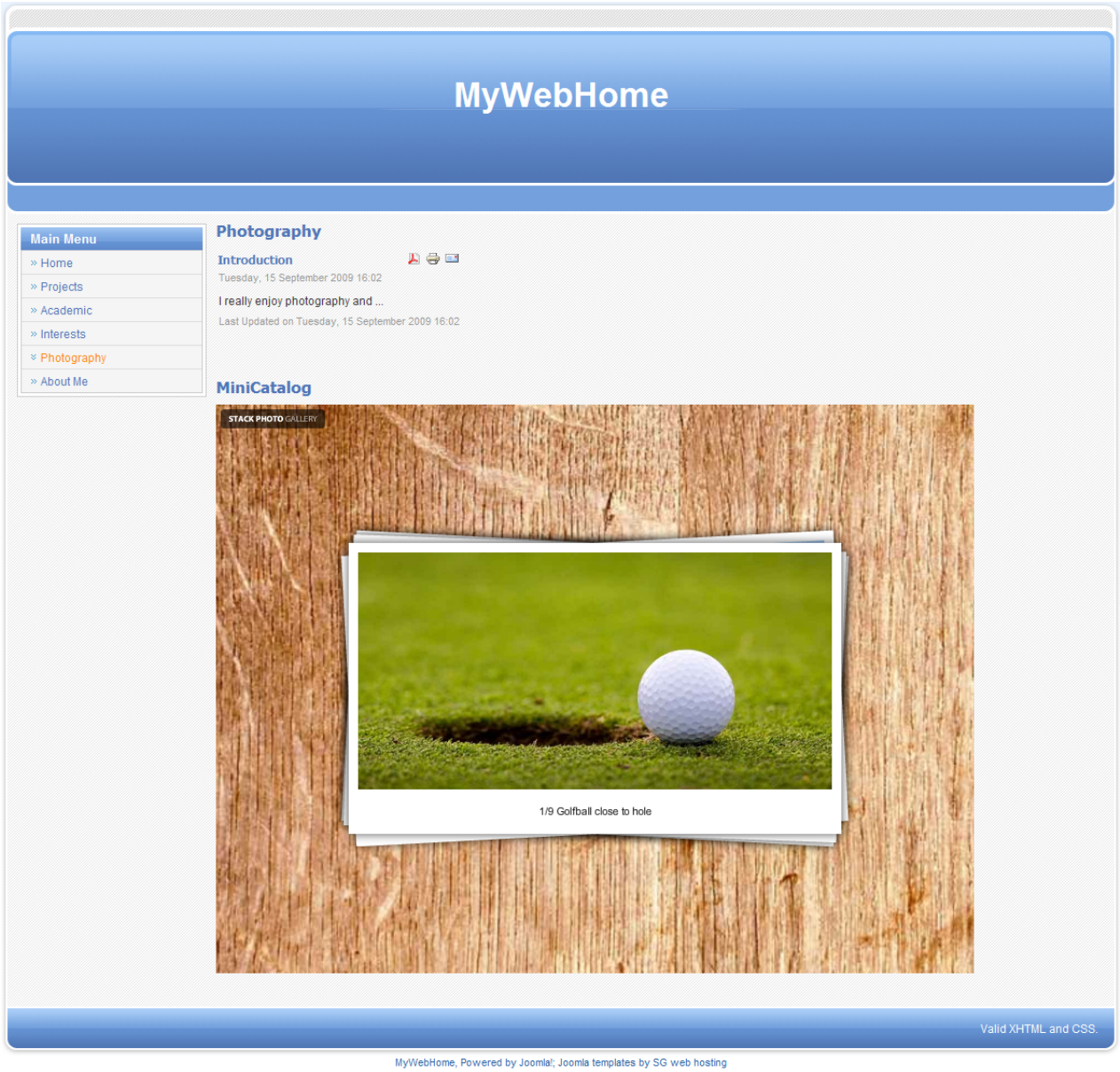


Figure A.28: *Photography* web page on joomla - figure A.12

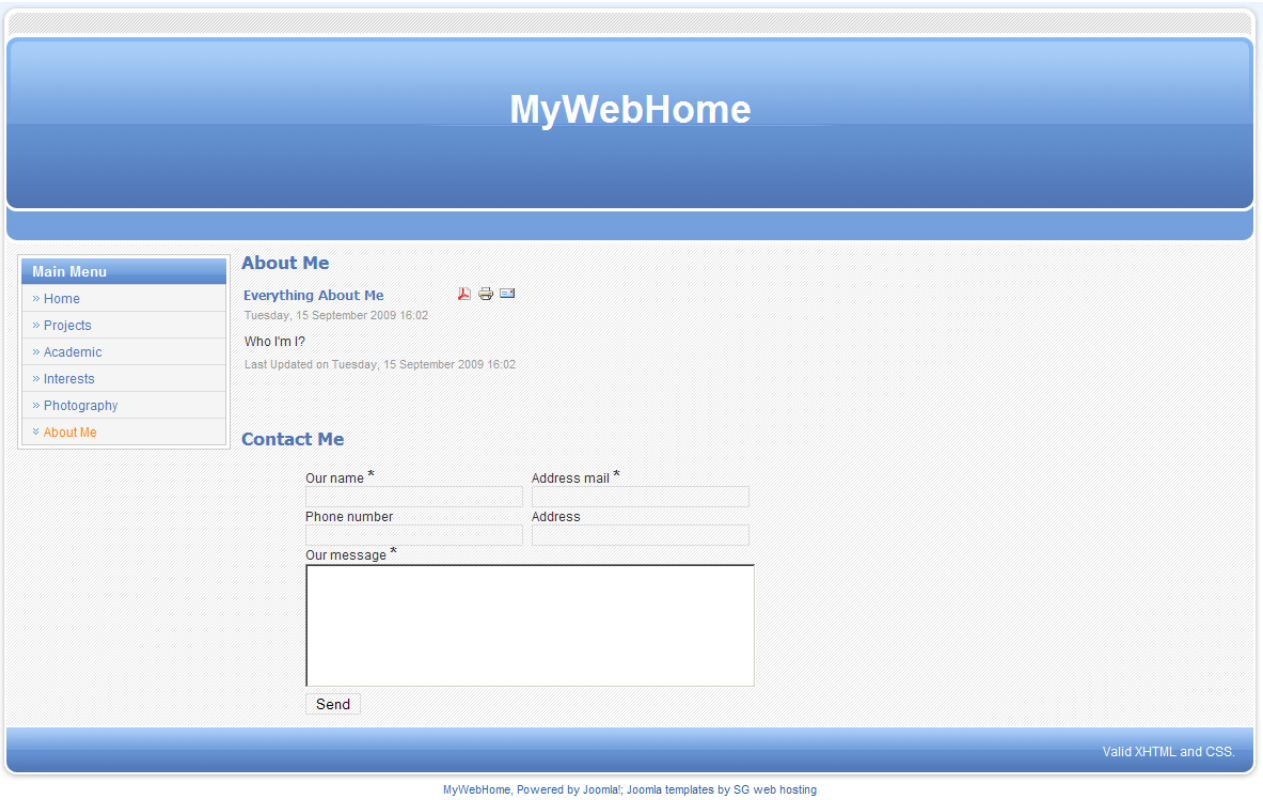


Figure A.29: *About Me* web page on joomla - figure A.13

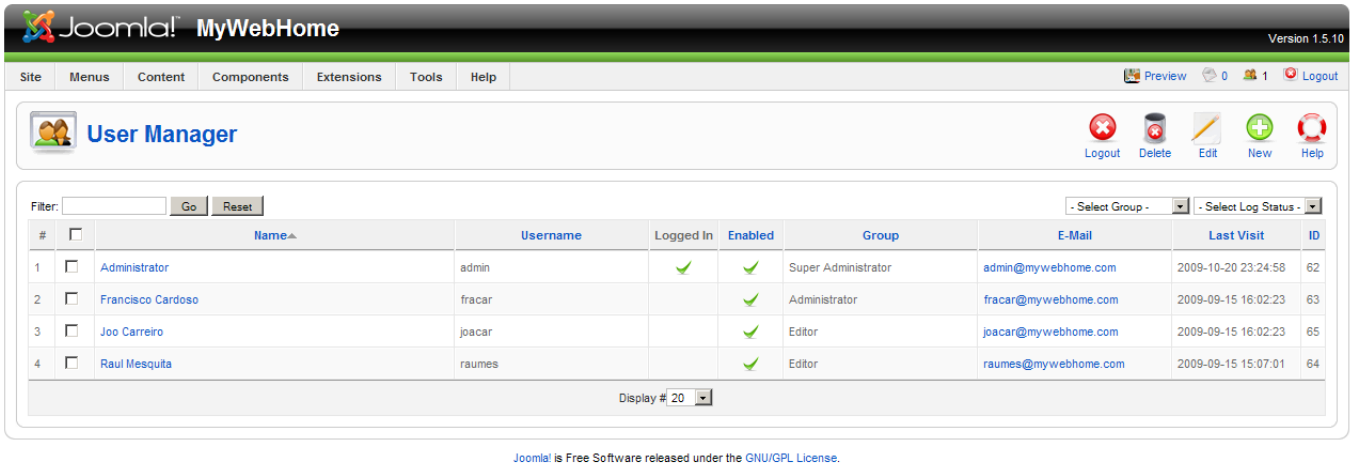


Figure A.30: *MyWebHome* roles on joomla - figure A.19. The joomla CMS doesn't support roles creations, so both the roles defined in the *MyWebHome*, "author" and "collaborator", where mapped to the joomla roles "Administrator" and "Editor", respectively.

A.3 *MyWebHome* on Drupal

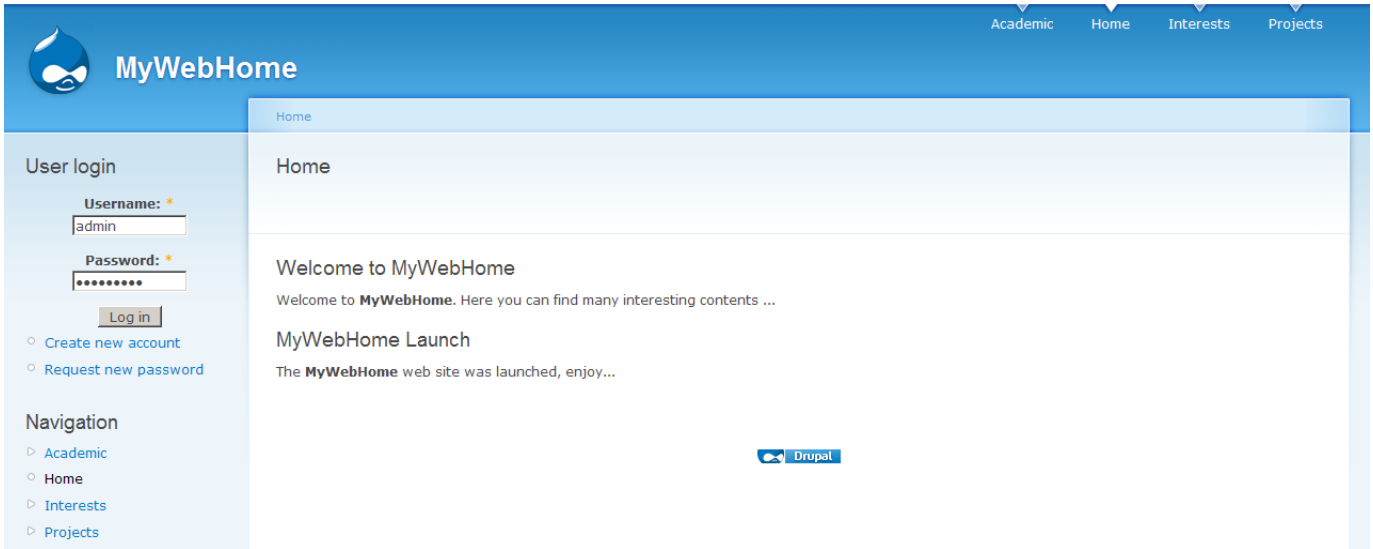


Figure A.31: *Home* web page on drupal - figure A.4

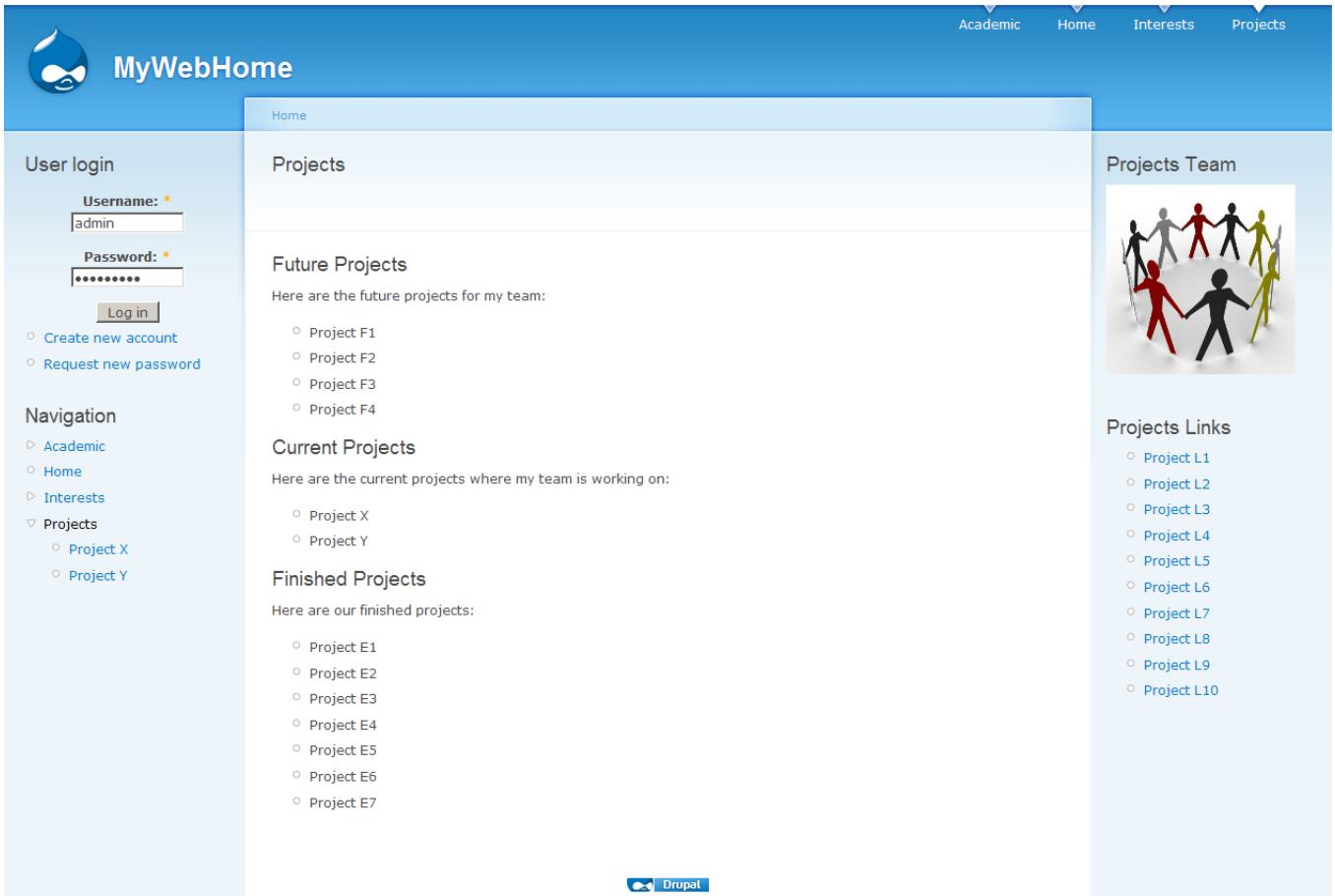


Figure A.32: *Projects* web page on drupal - figure A.5

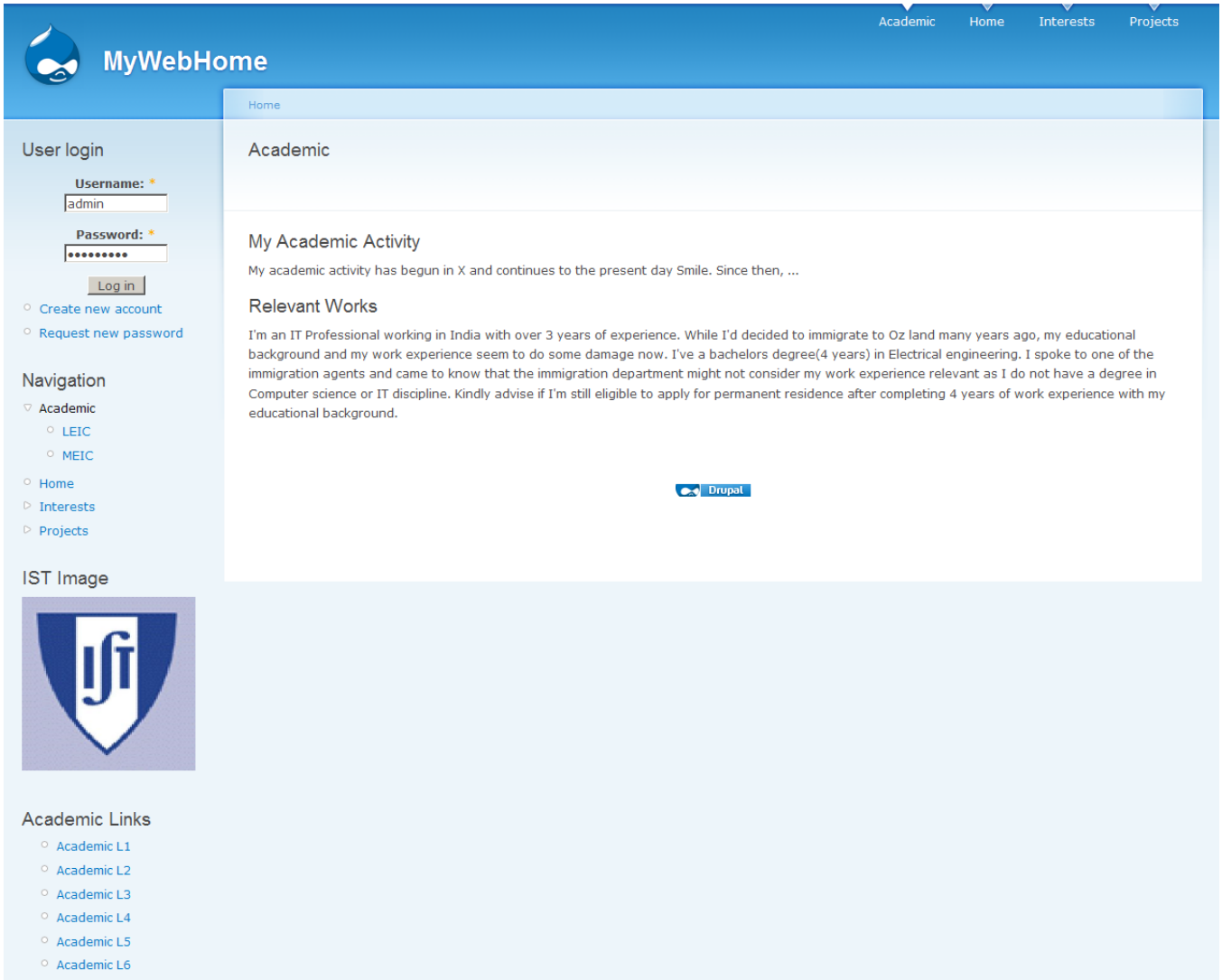


Figure A.33: *Academic* web page on drupal - figure A.6

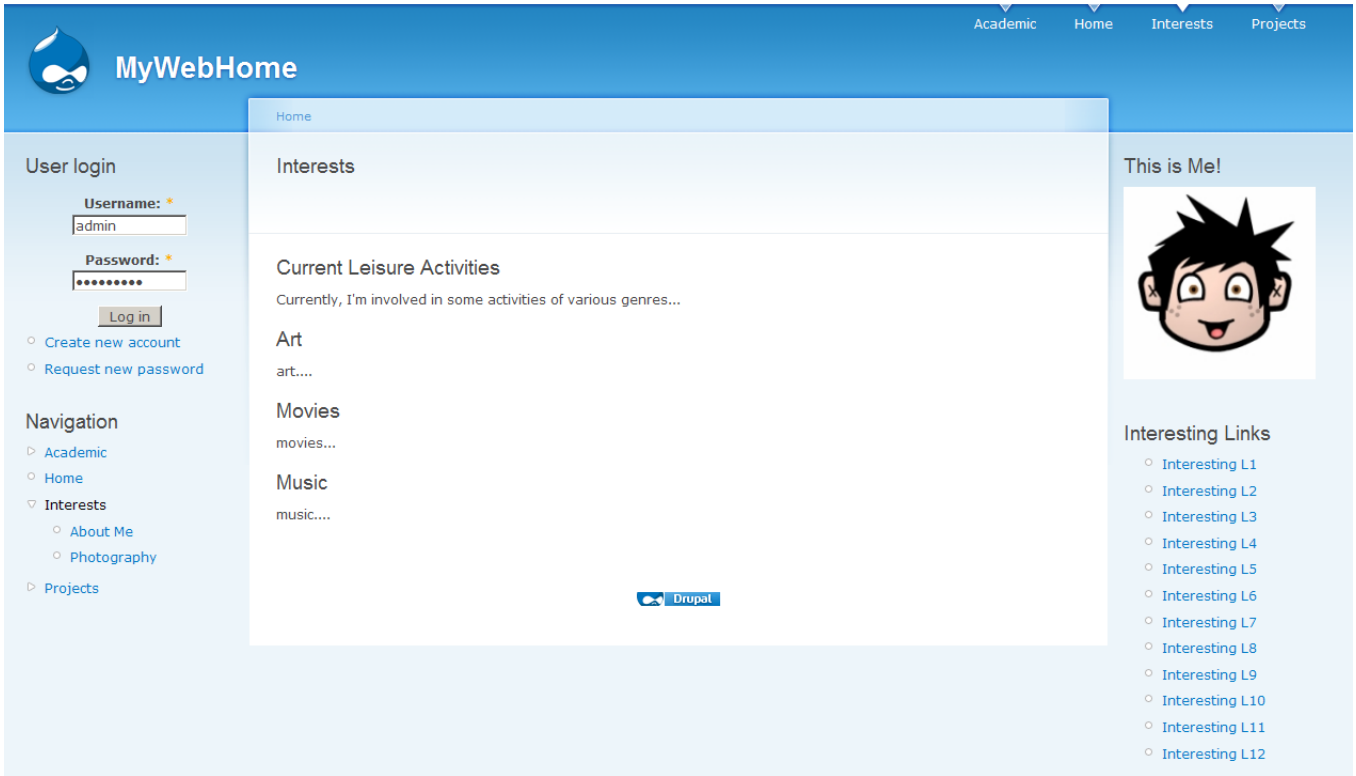


Figure A.34: *Interests* web page on drupal - figure A.7

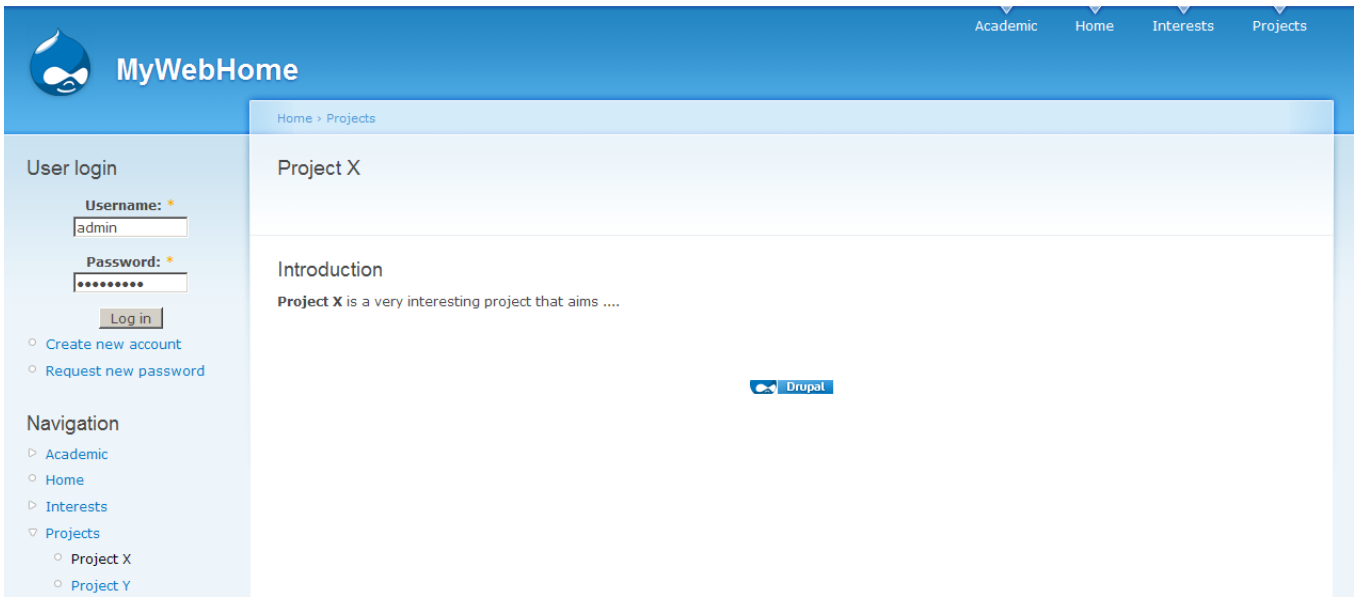


Figure A.35: *Project X* web page on drupal - figure A.8

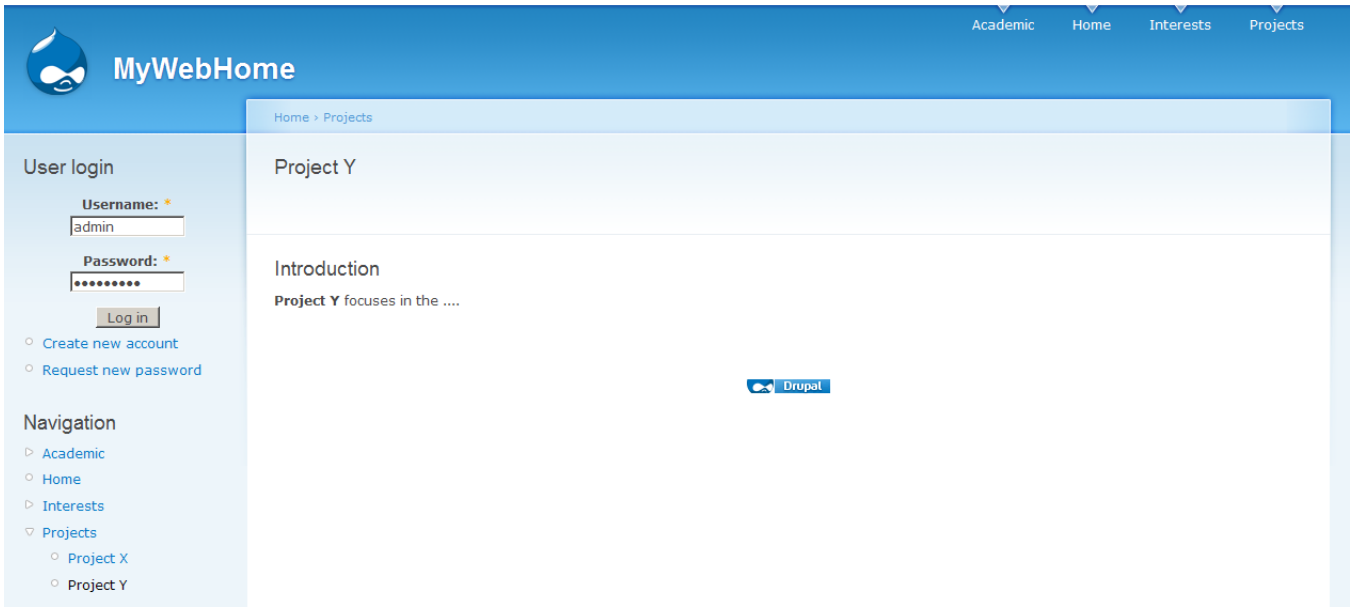


Figure A.36: *Project Y* web page on drupal - figure A.9

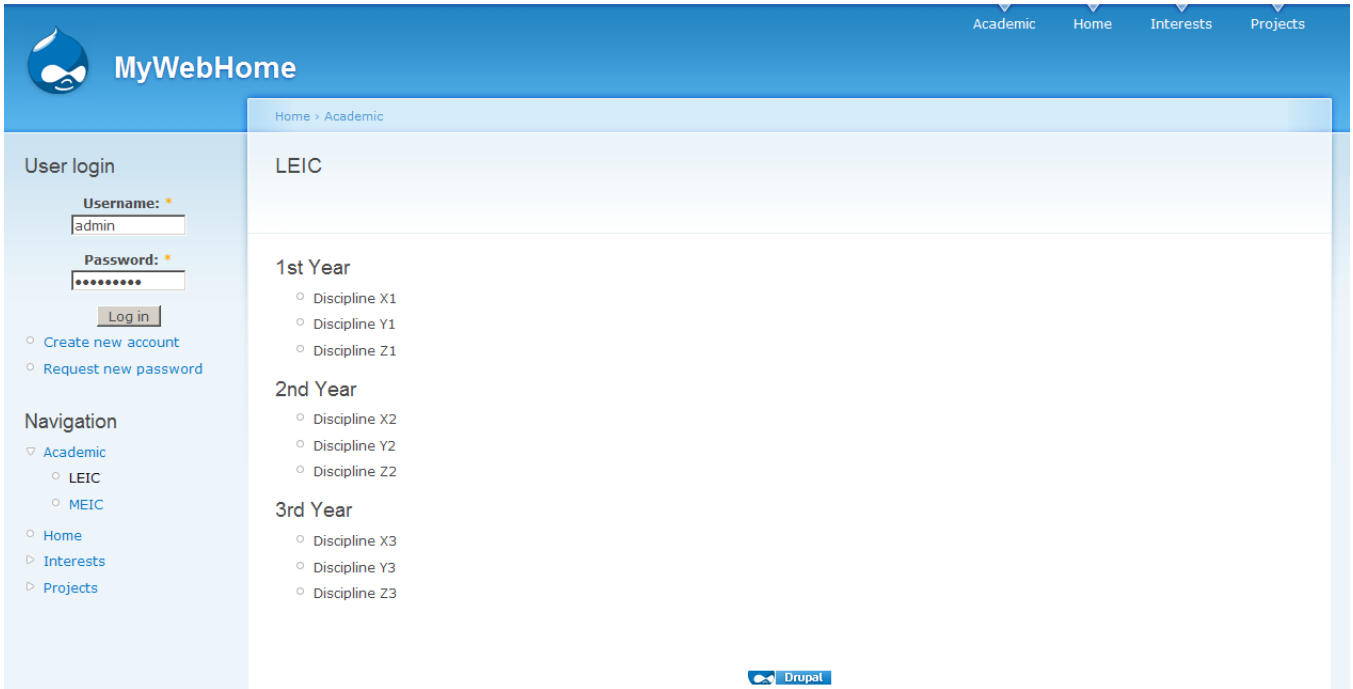


Figure A.37: *LEIC* web page on drupal - figure A.10

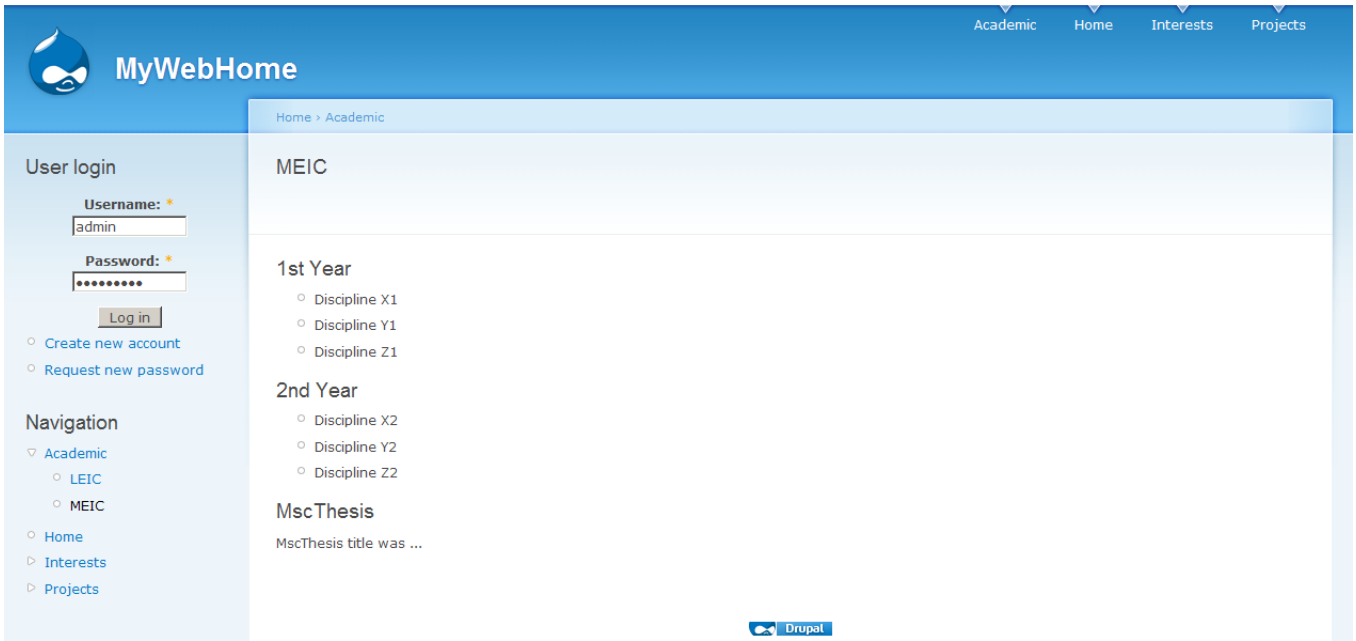


Figure A.38: *MEIC* web page on drupal - figure A.11

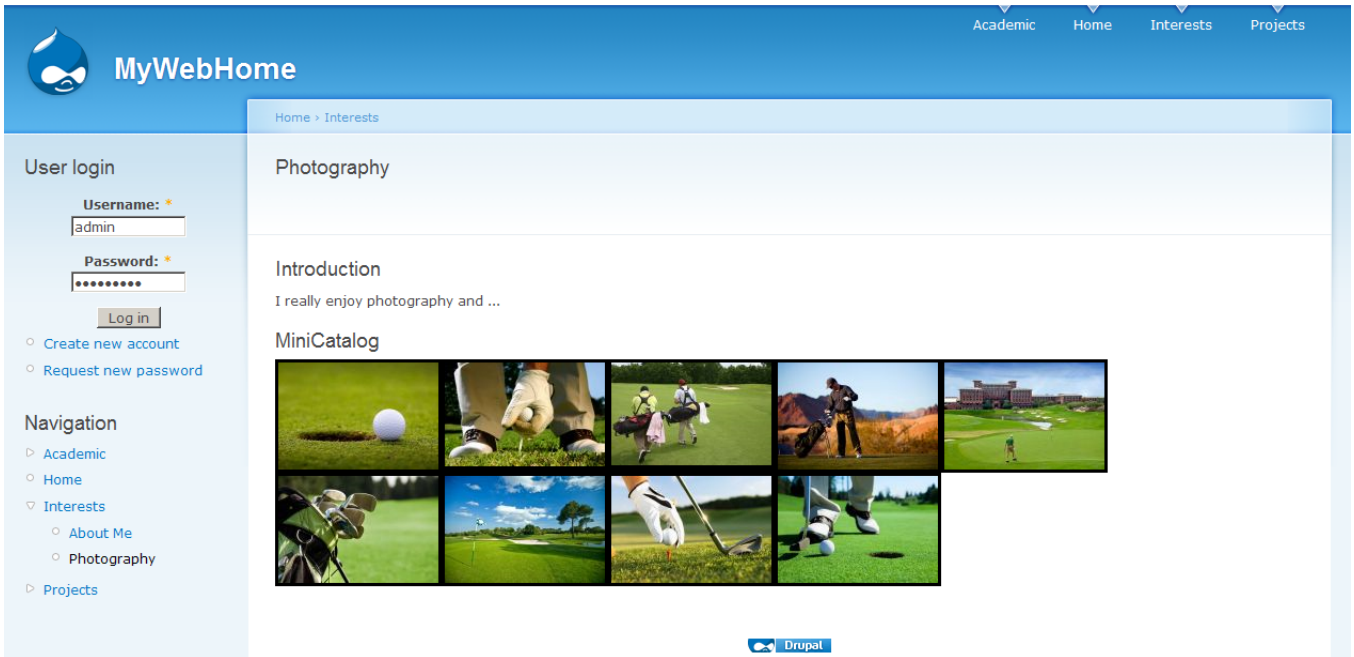


Figure A.39: *Photography* web page on drupal - figure A.12

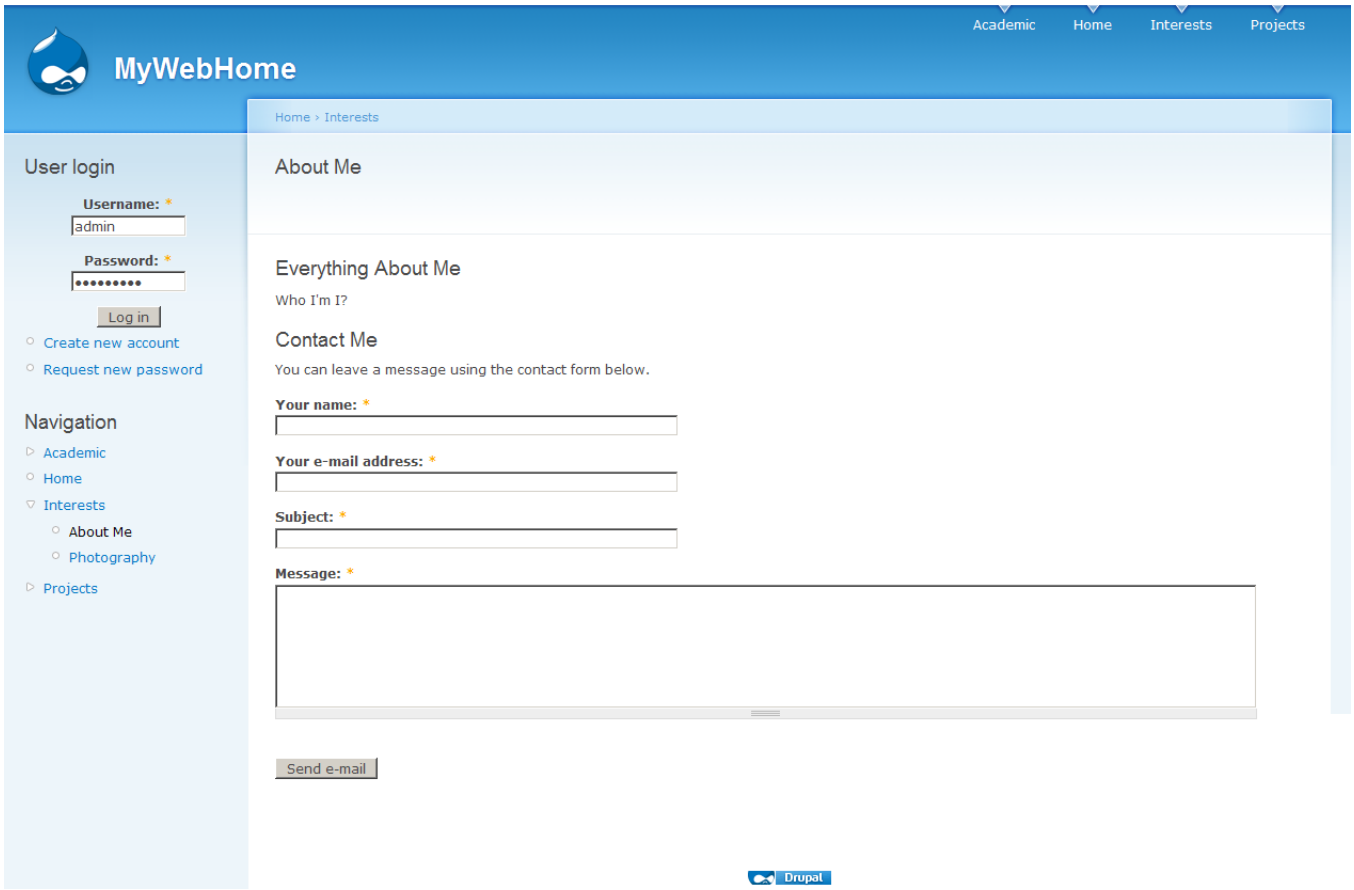


Figure A.40: *About Me* web page on drupal - figure A.13

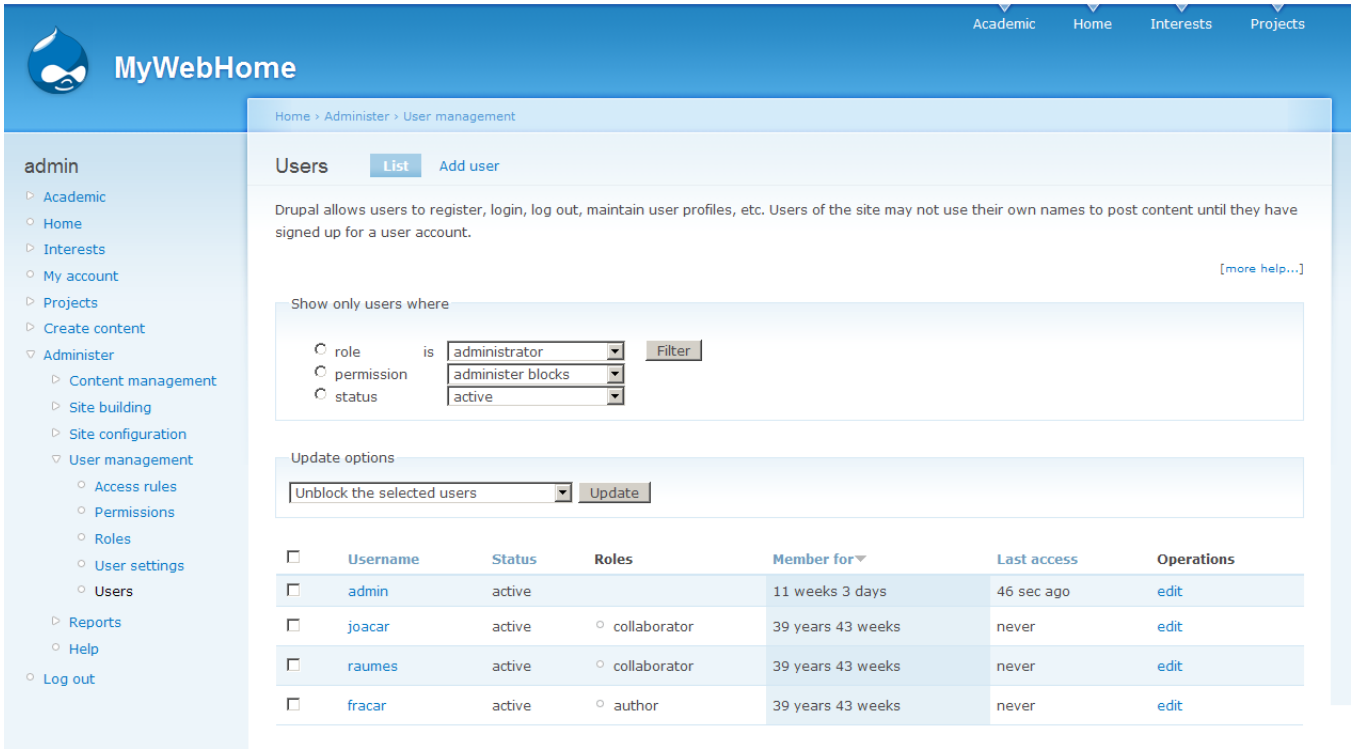


Figure A.41: *MyWebHome* roles on drupal - figure A.19.

Bibliography

[1] Drupal 5 - Getting Started. Technical report, Drupal CMS.

[2] Acceleo.org. Retrieved Monday 10th August, 2009 from http://www.acceleo.org.

[3] D. Anderson. Extending UML for UI. 2000.

[4] ArgoUML Project home. Retrieved Monday 5th June, 2006 from http://argouml.tigris.org.

[5] ArgoUWE. Retrieved Tuesday 25th August, 2009 from http://swik.net/argouwe.

[6] C. Atkinson and T. Kühne. The Role of Metamodeling in MDA, 2002. Retrieved Thursday 15th June, 2006 from http://www.metamodel.com/wisme-2002/papers/atkinson.pdf.

[7] B. Boiko. *Content Management Bible*. John Wiley & Sons, Hoboken, New Jersey, U.S.A., December 2001.

[8] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, and B. Selic. An MDA Manifesto. *Business Process Trends/MDA Journal*, May 2004. Retrieved Thursday 15th June, 2006 from http://www.bptrends.com/publicationfiles/05-04COLIBMManifesto-Frankel-3.pdf.

[9] P. Browning and M. Lowndes. JISC TechWatch Report: Content Management Systems. 2001.

[10] J. L. Carmo and A. R. d. Silva. The WebComfort Project. In *Proceedings of the Second International Conference of Innovative Views of .NET Technologies (IVNET'06)*. Sociedade Brasileira de Computação and Microsoft, October 2006. Retrieved Monday 17th March, 2008 from http://isg.inesc-id.pt/alb/static/papers/2006/jc-ivnet2006-webcomfort.pdf.

[11] J. L. V. d. Carmo. Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework. Master's thesis, Instituto Superior Técnico, Portugal, December 2006.

[12] L. Constantine and L. Lockwood. Software for Use: A Practical Guide for the Models and Methods of Usage-Centered Design. 1999.

[13] K. Czarnecki and U. Eisenecker. Generative Programming. 2000.

[14] Django Software Foundation. Retrieved Tuesday 17th March, 2009 from http://www.djangoproject.com/.

[15] Drupal CMS. Retrieved Thursday 19th March, 2009 from http://www.drupal.org.

[16] D. Duric. MDA-based Ontology Infrastructure. *Computer Science and Information Systems*, 1(1):91–116, February 2004. Retrieved Thursday 29th June, 2006 from http://www.comsis.fon.bg.ac.yu/ComSISpdf/Volume01/Papers/DraganDjuric.pdf.

[17] eZ Publish CMS. Retrieved Monday 10th August, 2009 from http://ez.no/.

[18] D. d. A. Ferreira. ProjectIT-RSL. Relatório Final de Trabalho Final de Curso, Instituto Superior Técnico, Portugal, October 2006.

[19] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg. Model-Driven Development Using UML 2.0: Promises and Pitfalls. *Computer*, 39(2):59–66, February 2006. Retrieved Monday 5th June, 2006 from http://doi.ieeecomputersociety.org/10.1109/MC.2006.65.

[20] C. Gane and T. Sarson. Structured Systems Analysis: Tools and Techniques. 1979.

[21] G. C. Gardner Tracy. A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard. 2003. Retrieved Tuesday 24th March, 2009 from http://www.omg.org/docs/ad/03-08-02.pdf.

[22] IBM. IBM Software – Rational Rose Data Modeler – Product Overview. Retrieved Monday 5th June, 2006 from http://www-306.ibm.com/software/awdtools/developer/datamodeler/.

[23] INESC-ID, Information Systems Group (GSI) – ProjectIT Website. Retrieved Tuesday 18th August, 2009 from http://isg.inesc-id.pt/alb/ProjectIT@81.aspx.

[24] JBoss CMS. Retrieved Monday 10th August, 2009 from http://www.jboss.org/.

[25] T. Jenkins. *Enterprise Content Management Solutions: What You Need to Know.* Open Text Corporation, April 2005.

[26] Joomla! CMS. Retrieved Wednesday 19th March, 2008 from http://www.joomla.org.

[27] E. Juliout. SOA and MDA with Acceleo. 2007.

[28] E. Juliout and S. Lacrampe. MDE for large projects : Today needs and tomorow standards. 2006.

[29] U. Kampffmeyer. ECM – Enterprise Content Management, 2006. Retrieved Tuesday 17th March, 2009 from http://www.project-consult.net/Files/ECM_WhitePaper_kff_2006.pdf.

[30] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. *Technical Report CMU/SEI-90-TR-21*, 1990.

[31] S. Kelly. Improving Developer Productivity With Domain-Specific Modeling Languages. *Developer.* – The Independent Magazine for Software Developers*, July 2005. Retrieved Friday 16th June, 2006 from http://www.developerdotstar.com/mag/articles/domain_modeling_language.html.

[32] D. Knauss. A Comparison of Capabilities and Features on Drupal, Joomla! And Wordpress. Technical report, New Local Media, August 2008.

[33] N. Koch, A. Knapp, G. Zhang, and H. Baumeister. *Web Engineering: Modelling and Implementing Web Applicationss.* Springer London, November 2007.

[34] N. Koch and A. Kraus. The Expressive Power of UML-based Web Engineering. In *Proceedings of the Second International Workshop on Web-Oriented Software Technology (IWWOST'2002)*, June 2002. Retrieved Wednesday 18th March, 2009 from http://www.pst.informatik.uni-muenchen.de/personen/kochn/IWWOST02-koch-kraus.PDF.

[35] N. Koch, A. Kraus, and R. Hennicker. The Authoring Process of the UML-based Web Engineering Approach. In *Proceedings of the First International Workshop on Web-Oriented Software Technology (IWWOST'2001)*, June 2001. Retrieved Wednesday 18th March, 2009 from http://www.dsic.upv.es/~west/iwwost01/files/contributions/NoraKoch/Uwe.pdf.

[36] G. Lemos and T. Matias. Projecto XIS - Abordagem e Ferramenta de Desenvolvimento (Case Tool e UML). Relatório Final de Trabalho Final de Curso, Instituto Superior Técnico, Portugal, July 2003.

[37] MagicDraw. Retrieved Tuesday 25th August, 2009 from http://www.magicdraw.com/.

[38] MagicUWE. Retrieved Tuesday 25th August, 2009 from http://uwe.pst.ifi.lmu.de/toolMagicUWE.html.

[39] Microsoft. Enterprise Content Management: Breaking the Barriers to Broad User Adoption. Retrieved Tuesday 3rd June, 2008 from http://www.microsoftio.com/content/bpio/prospect_and_demand/ecm_wp2.pdf, June 2006.

[40] J. Miller and J. Mukerji. MDA Guide Version 1.0.1. 2003.

[41] Model transformation at Inria / Introduction to Model-Driven Engineering. Retrieved Saturday 24th June, 2006 from http://modelware.inria.fr/article65.html.

[42] N. Moreno, P. Fraternalli, and A. Vallecillo. A UML 2.0 profile for WebML modeling. In *ICWE '06: Workshop proceedings of the sixth international conference on Web engineering*, New York, NY, USA, July 2006. ACM. Retrieved Wednesday 18th March, 2009 from `http://doi.acm.org/10.1145/1149993.1149998`.

[43] N. Moreno, P. Fraternalli, and A. Vallecillo. WebML modelling in UML. *IET Software*, 1(3):67–80, June 2007.

[44] Object Management Group. Retrieved Monday 5th June, 2006 from `http://www.omg.org`.

[45] Object Management Group. Model Driven Architecture. Retrieved Tuesday 17th March, 2009 from `http://www.omg.org/mda`.

[46] PHP. Retrieved Tuesday 25th August, 2009 from `http://php.net`.

[47] Plone CMS. Retrieved Monday 10th August, 2009 from `http://plone.org/`.

[48] R. B. Queiroga. The XIS Case Tool. Relatório Final de Trabalho Final de Curso, Instituto Superior Técnico, Portugal, October 2004.

[49] Ruby on Rails. Retrieved Tuesday 17th March, 2009 from `http://rubyonrails.org/`.

[50] J. d. S. Saraiva and A. R. d. Silva. Eclipse.NET: An Integration Platform for ProjectIT-Studio. In *Proceedings of the First International Conference of Innovative Views of .NET Technologies (IVNET'05)*, pages 57–69. Instituto Superior de Engenharia do Porto and Microsoft, July 2005. Retrieved Wednesday 21st June, 2006 from `http://w2ks.dei.isep.ipp.pt/labdotnet/files/IVNET/EclipseNet_p.pdf`.

[51] J. d. S. Saraiva and A. R. d. Silva. Evaluation of MDE Tools from a Metamodeling Perspective. *Journal of Database Management*, 19(4):21–46, October/December 2008.

[52] J. d. S. Saraiva and A. R. d. Silva. The ProjectIT-Studio UMLModeler: A tool for the design and transformation of UML models. In M. P. Cota, editor, *Actas de la 3ª Conferencia Ibérica de Sistemas e Tecnologias de Informação (CISTI 2008)*, pages 687–698, June 2008.

[53] J. d. S. Saraiva and A. R. d. Silva. The WebComfort Framework: An Extensible Platform for the Development of Web Applications. In IEEE Computer Society, editor, *Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2008)*, pages 19–26, September 2008.

[54] J. P. P. M. d. S. Saraiva. Automatic Systems' Development (Desenvolvimento Automático de Sistemas). Final Graduation Report (Relatório Final de Trabalho Final de Curso), Instituto Superior Técnico, Portugal, July 2005.

[55] J. P. P. M. d. S. Saraiva. The UML Modeling Tool of ProjectIT-Studio. Master's thesis, Instituto Superior Técnico, Portugal, December 2006.

[56] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006. Retrieved Tuesday 17th March, 2009 from `http://doi.ieeecomputersociety.org/10.1109/MC.2006.58`.

[57] L. Schou. Creating a model-driven Web application framework. Master's thesis, Technical University of Denmark, Denmark, March 2008. Retrieved Wednesday 18th March, 2009 from `http://www.imm.dtu.dk/English/Research/Software_Engineering/Publications.aspx?lg=showcommon&id=213586`.

[58] P. Selonen, K. Koskimies, and M. Sakkinen. Transformations Between UML Diagrams. *Journal of Database Management*, 14(3):37–55, 2003.

[59] A. R. d. Silva. The XIS Approach and Principles. In *Proceedings of the 29th EUROMICRO Conference*. IEEE Computer Society, September 2003. Retrieved Wednesday 14th June, 2006 from `http://doi.ieeecomputersociety.org/10.1109/EURMIC.2003.1231564`.

[60] A. R. d. Silva, G. Lemos, T. Matias, and M. Costa. The XIS Generative Programming Techniques. In *Proceedings fo the 27th COMPSAC Conference*. IEEE Computer Society, November 2003. Retrieved Monday 5th June, 2006 from `http://doi.ieeecomputersociety.org/10.1109/CMPSAC.2003.1245347`.

[61] A. R. d. Silva, J. Saraiva, D. Ferreira, R. Silva, and C. Videira. Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools. *IET Software: On the Interplay of .NET and Contemporary Development Techniques*, 1(6):294–314, December 2007.

[62] A. R. d. Silva, J. Saraiva, R. Silva, and C. Martins. XIS – UML Profile for eXtreme Modeling Interactive Systems. In *Fourth International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007)*, pages 55–66, Los Alamitos, CA, USA, March 2007. IEEE Computer Society. Retrieved Friday 6th April, 2007 from `http://doi.ieeecomputersociety.org/10.1109/MOMPES.2007.19`.

[63] A. R. d. Silva and C. Videira. *UML, Metodologias e Ferramentas CASE*. Centro Atlântico, Portugal, April 2001.

[64] A. R. d. Silva and C. Videira. *UML, Metodologias e Ferramentas CASE, 2ª Edição, Volume 2*. Centro Atlântico, Portugal, March 2008.

[65] A. R. d. Silva, C. Videira, J. Saraiva, D. Ferreira, and R. Silva. The ProjectIT-Studio, an integrated environment for the development of information systems. In *Proceedings of the Second International Conference of Innovative Views of .NET Technologies (IVNET'06)*, pages 85–103. Sociedade Brasileira de Computação and Microsoft, October 2006. Retrieved Thursday 14th September, 2006 from `http://berlin.inesc-id.pt/alb/static/papers/2006/ivnet2006-pit-v1.0c.pdf`.

[66] R. M. T. d. Silva. ProjectIT – Produção Automática de Software. Relatório Final de Trabalho Final de Curso, Instituto Superior Técnico, Portugal, October 2006.

[67] M. Simos, D. Creps, C. Klinger, L. Levine, and D. Allemang. Organization domain modelling (ODM) guidebook version 2.0. *Technical Report STARS-VC-A025/001/00*, 1996.

[68] SIQuant – Engenharia do Território e Sistemas de Informação. Retrieved Thursday 19th March, 2009 from `http://www.siquant.pt`.

[69] SparxSystems. Enterprise Architect – UML Design Tools and UML CASE tools for software development. Retrieved Monday 5th June, 2006 from `http://www.sparxsystems.com/products/ea.html`.

[70] P. Suh, D. Addey, D. Thiemecke, and J. Ellis. *Content Management Systems (Tools of the Trade)*. Glasshaus, October 2003.

[71] Sun Microsystems, Inc. Retrieved Tuesday 25th August, 2009 from `http://www.mysql.com/`.

[72] R. N. Taylor, W. Tracz, and L. Coglianese. Software development using domain-speci?c software architectures. *ACM SIGSOFT Software Engineering Notes*, pages 27–37, 1995.

[73] The CMS Matrix. Retrieved Tuesday 17th March, 2009 from `http://www.cmsmatrix.org`.

[74] The Object-Oriented Hypermedia Design Model (OOHDM). Retrieved Monday 10th August, 2009 from `http://www.telemidia.puc-rio.br/oohdm/oohdm.html`.

[75] The Official Microsoft ASP.NET Site. Retrieved Monday 17th March, 2008 from `http://www.asp.net`.

[76] D. Thomas. MDA: Revenge of the Modelers or UML Utopia? *IEEE Software*, 21(3):15–17, May/June 2004. Retrieved Monday 5th June, 2006 from `http://doi.ieeecomputersociety.org/10.1109/MS.2004.1293067`.

[77] J.-P. Tolvanen. Domain-Specific Modeling for Full Code Generation. *Methods & Tools*, 13(3):14–23, 2005. Retrieved Friday 16th June, 2006 from `http://www.methodsandtools.com/archive/archive.php?id=26`.

[78] P. V. G. Tom Mens. A Taxonomy of Model Transformation. 2005. Retrieved Tuesday 24th March, 2009 from `http://tfs.cs.tu-berlin.de/gramot/Gramot2005/FinalVersions/PDF/MensVanGorp.pdf`.

[79] Typo3 CMS. Retrieved Thursday 20th March, 2008 from http://www.typo3.org.

[80] UWE – UML-based Web Engineering. Retrieved Wednesday 18th March, 2009 from http://www.pst.ifi.lmu.de/projekte/uwe.

[81] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. pages 26—-36, 2000. Retrieved Monday 10th August, 2009 from www.program-transformation.org/.

[82] A. van Deursen and T. Kuipers. Building documentation generators. *In Proceedings International Conference on Software Maintenance*, pages 40–49, 1999.

[83] WebComfort.org. Retrieved Thursday 19th March, 2009 from http://www.webcomfort.org.

[84] WebML.org. Retrieved Wednesday 18th March, 2009 from http://www.webml.org.