

Model-Driven Development Approach for Content Management Systems based Applications

Paulo Roberto Carvalho Nunes Filipe

Thesis to obtain the Master of Science Degree in
Information Systems and Computer Engineering

Supervisor: Prof. Dr. Alberto Manuel Rodrigues da Silva

Examination Committee

Chairperson: Prof. Dr. João António Madeiras Pereira

Supervisor: Prof. Dr. Alberto Manuel Rodrigues da Silva

Member of the Committee: Dr. João Paulo Pedro Mendes de Sousa Saraiva

November 2015

Acknowledgments

I would like to thank Professor Alberto Rodrigues da Silva, my adviser, for the challenge and confidence that was placed in me. His supervision, availability, insight and support were important to the journey that was this year. Also, to my colleagues at IST, especially André Ribeiro, for all the ideas, feedback and the good mood that made me feel welcomed back.

To all my friends and extended family, that had encourage me and supported through all these years. A high thanks to my mom Manuela Carvalho, João Barquinha and Sancha Azevedo e Silva.

Finally, for without her I had not accepted this challenge, a more than thanks for my amazing wife Marlene Nunes Filipe, and Maria that gave me that extra push. Her patience, support (both physical and mental), encouragement and sacrifices were the essential strength that kept me on track. With all my love, thanks for the blessing that is the family we built.

Lisbon, October 2015

Paulo Nunes Filipe

Resumo

Sistemas de Gestão de Conteúdos (CMS) são plataformas de aplicações web usadas em vários contextos, que estão a ficar mais populares. Os CMS permitem utilizadores não técnicos gerir o conteúdo e as configurações de *websites* usando módulos orientados a dados, que abstraem funcionalidades sem necessidade de conhecimento de programação. No entanto, de modo a suportar cenários mais concretos ou complexos, é necessário o desenvolvimento de módulos específicos, feitos à medida. Para isso, os programadores têm para criar módulos personalizados para o CMS utilizado, e assim, é exigido que conheçam a linguagem de programação e que tenham as competências técnicas desse CMS. Isto é especialmente importante quando já existe um repositório de dados em uso por outras aplicações que CMS deve gerir para visualizar parte da informação.

Esta dissertação propõe o XIS-CMS, uma abordagem de Desenvolvimento Conduzido por Modelos para desenvolver um conjunto de módulos para os CMS. A abordagem XIS-CMS inclui uma Linguagem Específica de Domínio para modelação, definida como um perfil de UML, e um conjunto de ferramentas integradas no Sparx Systems Enterprise Architect e tecnologias Eclipse Modeling Framework. Para aplicar o XIS-CMS, são definidos modelos independentes às plataformas para especificar o módulo do CMS e gerar o código correspondente. Isto aumenta a produtividade e portabilidade para múltiplas plataformas de CMS, facilitando migrações e apresenta uma visão mais adequada para *stakeholders*. A validação do XIS-CMS foi baseada em casos de estudo e sessões de teste, sendo os resultados discutidos e comparados com trabalhos no mesmo contexto de CMS.

Palavras-chave: Sistemas de Gestão de Conteúdos; Modelos Independentes da Plataforma; Desenvolvimento Conduzido por Modelos; Linguagem Específica de Domínio

Abstract

Content Management Systems (CMS) are increasingly popular web application platforms used in multiple domains. CMS allow non-technical users to manage the content and features of websites with data-driven web modules, that abstract functionality without requiring any software programming knowledge. However, without the development of specific modules, a CMS usually cannot support more specific or complex scenarios. For that purpose, developers have to build custom modules using the CMS-specific language, which requires they must master the programming and technical skills. This is especially important when there is already a data repository in use on other applications and the CMS must manage or expose part of the information.

This dissertation proposes the XIS-CMS, a Model-Driven Development approach to develop CMS toolkit, containing a set of modules. XIS-CMS approach includes a domain-specific modelling language, defined as a UML profile, and a companion framework defined on top of Sparx Systems Enterprise Architect and Eclipse Modeling Framework technologies. Using the XIS-CMS, platform-independent models are defined to specify the CMS module and generate the corresponding source code, increasing productivity and portability when developing for multiple CMS platforms, facilitating migrations and give a more abstract vision to non-technical stakeholders. This research presents an evaluation of the XIS-CMS, based on case studies and user test sessions, discuss its results and compare it with other works in the same field of CMS.

Keywords: Content Management Systems; Platform-Independent Models; Model-Driven Development; Domain-Specific Language

Table of Contents

Acknowledgments	iv
Resumo.....	vi
Abstract	viii
Table of Contents	x
List of Figures	xii
List of Tables.....	xiii
List of Acronyms	xiv
1. Introduction	1
1.1. Problem Definition	2
1.2. Thesis Statement.....	2
1.3. Proposed Solution	2
1.4. Context	4
1.5. Methodology	5
1.6. Outline	7
2. Background.....	8
2.1. Model-Driven Development	8
2.1.1. Model-Driven Engineering	8
2.1.2. Domain Specific Languages	9
2.1.3. Model Transformations	10
2.2. Content Management Systems	11
2.2.1. CMS Structure	12
2.2.2. CMS Platforms.....	13
2.3. MDD Approaches for CMS	14
3. XIS-CMS Approach	17
3.1. Background	17
3.2. Overview.....	18
3.3. Design Approaches	19
3.4. Case Studies	20
4. XIS-CMS Language.....	23
4.1. Toolkit View.....	24

4.2.	Entities View	24
4.2.1.	Domain View.....	24
4.2.2.	BusinessEntities View.....	25
4.3.	Roles View.....	27
4.4.	Module View	27
4.4.1.	UseCases View	27
4.4.2.	User-Interfaces View	29
5.	XIS-CMS Framework.....	32
5.1.	Overview.....	32
5.2.	Model Editor	34
5.3.	Model Validator.....	36
5.4.	Model-to-Model Transformations.....	36
5.5.	Model-to-Text Transformations	39
6.	Evaluation.....	43
6.1.	Case Study A – Self-Evaluation.....	43
6.2.	Case Study B – User Session Evaluation	44
6.3.	Discussion of the Related Work	46
7.	Conclusion	49
7.1.	Main Contributions.....	50
7.2.	Future Work.....	50
	References	52

List of Figures

Figure 1: Language dependencies of XIS-CMS.....	3
Figure 2: Action Research methodology cycle phases	5
Figure 3: Relation between Model-Driven Initiatives (Ameller, 2009)	9
Figure 4: Methodology and frameworks classification	15
Figure 5: The multi-view organization of XIS-CMS and the transformations process.....	19
Figure 6: The multi-view organization of XIS-CMS	23
Figure 7: Example of the Toolkit View of the Supplier application.....	24
Figure 8: Metamodel of the Domain View.	25
Figure 9: Example of the Domain View of the Suppliers application.....	25
Figure 10: Metamodel of the BusinessEntities View.	26
Figure 11: Example of the BusinessEntities View of the Suppliers application.....	26
Figure 12: Metamodel of the Roles View.....	27
Figure 13: Example of the Roles View of the Suppliers application.....	27
Figure 14: Metamodel of the UseCases View.	28
Figure 15: Example of the UseCases View for the Product Module	28
Figure 16: Metamodel of the InteractionSpace View.....	30
Figure 17: Example of the Index interaction space of the Product module.....	30
Figure 18: Metamodel of the NavigationSpace View.....	31
Figure 19: Example of the NavigationSpace View of the Product Module.....	31
Figure 20: Development process for XIS-CMS approach.....	32
Figure 21: Profile definition of the XIS-CMS diagrams.....	34
Figure 22: XIS-CMS project structure created by the template.....	35
Figure 23: Launch of the Model-to-Model transformations tool.....	37
Figure 24: Example of a XisModule and the corresponding tagged values.....	37
Figure 25: Example of the Interactions space for the Supplier Module	38
Figure 26: Example of Setting InteractionSpace and the screen	38
Figure 27: Example of Acceleo code to create a class property	40
Figure 28: Organization of the Acceleo project for DNN.	41
Figure 29: Organization of the Visual Studio solution for DNN	42
Figure 30: Example of the Suppliers Toolkit modules index screen.....	43

List of Tables

Table 1: Average score per question of XIS-CMS Language	45
Table 2: Average score per question of XIS-CMS Framework	45
Table 3: Average score per question of XIS-CMS Approach.....	46
Table 4: Average score per section of XIS-CMS	46
Table 5: CMS Model-Driven Development approaches	48

List of Acronyms

CMS	Content Management Systems
DNN	DotNetNuke
DSL	Domain-specific Language
EA	Enterprise Architect
ECM	Enterprise Content Management
EMF	Eclipse Modeling Framework
M2M	Model-to-Model
M2T	Model-to-Text
MDA	Model-Driven Architecture
MDD	Model-driven development
MDE	Model-Driven Engineering
MDG	Model Driven Generation
MOF	Meta-Object Facility
MTL	Model to Text Language
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform-Independent Model
PM	Project Management
PSM	Platform Specific Model
QVT	Query-View-Transformation
RE	Requirements Engineering
RSL	Requirements Specification Language
UI	User Interface
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

1. Introduction

With the increase of performance in hardware and network components, software systems are growing more complex, more dynamic and solving harder problems. Also, with the increase of Internet use, most applications, for business purposes, must be easy to use and remotely accessible, representing an increase of development complexity and cost for these applications. To answer this increase on complexity, Software Engineering tries to accompany the project management from design until development through the use of best-practices, approaches and methodologies. Software Engineering is a multi-disciplinary field, applied to social and technological contexts, from project management using methodologies, such as SCRUM, to programming using best-practices norms or object-oriented patterns.

With the proliferation of information, documents and contents that an organization produces and manages, the need of structured applications that can support these requirements was raised, in particular Content Management Systems (CMS), that offers a web application to support the management of these contents or Enterprise Content Management (ECM) systems (Suh, Ellis, & Thiemecke, 2002).

Content Management Systems offers a web application, which structure can be easily configured without requiring programming effort via configuration properties and can manage the organizational content. Therefore, in the present, CMS represent a critical component to the success of organizational websites, for both internal and external use. CMS content can normally be created or updated by using editor that offer a What-you-see-is-what-you-get (WYSIWYG) view through rich editors or advanced code editors. Also, it has an extensibility capacity of install new plugins, or toolkits, which give new functionalities to the base modules that forms a base installation of a CMS (e.g. Images, HTML, and calendar). Although, to show and manage content that is particular to an application, the developers must customize via programming new modules and make use of the CMS extensibility function. This task can become an Information Systems project with all implications, removing the vantage of the rapid time-to-market and ease of use that a CMS offers. This make the development of new modules a time-consuming task, which is suggested to errors and maintenance of a specialized team to the make changes as needed (Saraiva & Silva, 2009).

Model-driven development (MDD) is a development approach that tries to mitigate the above problems in multiple application domains (Schmidt, 2006). MDD is a software development paradigm that combines Domain-specific Languages (DSLs) with transformations engines and generators. DSLs are textual or graphical languages that specify an application domain with its requirements, functionalities and structure. They are described using metamodels that define the domain concepts and associations between them. Additionally, transformations engines and generators are tools that process a model (defined in a DSL) and generate other models, through Model-to-Model (M2M) transformations, or textual artefacts such as source code or documentation files, through Model-to-Text (M2T) transformations (Fondement & Raul, 2004).

1.1. Problem Definition

This dissertation addresses the following research questions:

R.Q.1: How to apply a MDD approach to the development of CMS modules?

R.Q.2: How to specify a CMS module in a platform-independent way?

R.Q.3: What concepts and view should be used to specify CMS modules applications?

R.Q.4: How to generate the source code artefacts to CMS platforms?

When developing new modules to a Content Management System, the ideal is design the modules with the information of which entities will be manipulated and how that management will be done. If this information can be generated without specify the platform where will be instantiated, it bring the advantages of ease to discuss with higher-level stakeholders and achieve a higher portability in the implementation phase.

1.2. Thesis Statement

This thesis argues that ***MDD can be successfully applied to the development of CMS modules, increasing the comprehension of the application and decreasing the development and maintenance effort.***

These goals can be achieved by identifying the models of the problem domain and using transformations to create new models and source code artefacts, through Model-to-Model and Model-to-Text transformations. This is accomplished using Platform-Independent Models according to the XIS-CMS Language and the transformation of the XIS-CMS framework. This approach generates the entire code and solution to a simple CRUD management CMS module.

1.3. Proposed Solution

The proposed solution of this dissertation is the **XIS-CMS**, a Domain Specific Language, implemented as a UML profile and a companion framework to design and generate the solution of CMS modules, through a Model-Driven Development approach.

XIS-CMS inherits some aspects from other works, such as XIS (Silva A. , Saraiva, Silva, & Martins, 2007), CMS-ML (Saraiva J. , 2013) and XIS-Mobile (Ribeiro, 2014). Its uses the general structure and design approaches from XIS, with the multi-view organization and main packages. Begin developed in the CMS context, CMS-ML offers the basis of concepts and research ground. XIS-Mobile, being a successful implementation of the XIS profile, offers a canvas to the framework methodology and supporting technologies (Ribeiro & Silva, 2014). Figure 1 show the relations, in a loose dependency definition, between these languages.

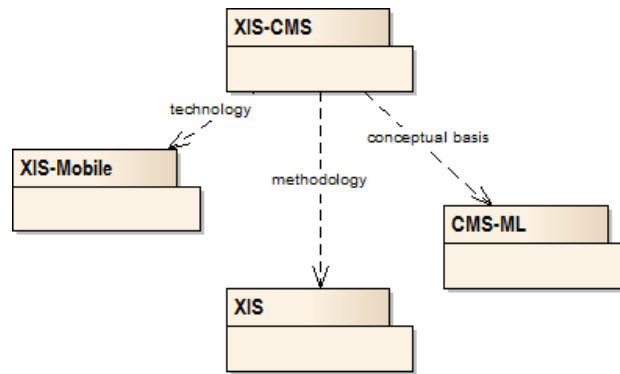


Figure 1: Language dependencies of XIS-CMS.

XIS-CMS Language is a domain specific language, defined in a way so that platform-independent CMS modules can be defined. Some of the concepts of the XIS-CMS language are inherited from the XIS Language, but introduces new views and an appropriate multi-view organization.

The **XIS-CMS Framework** is a MDD-based framework that supports the XIS-CMS Language and allows its use. The suggested development process of a CMS module using the XIS-CMS framework consists in four steps: (1) the definition of the required views using the Model Editor, implemented through the use of the Sparx Systems Enterprise Architect (EA)¹; (2) their validation with the Model Validator, through the EA Validator API; (3) the generation of the User-Interfaces View models with the Model-to-Model transformations; and finally (4) the generation of the module's source code through the Model-to-Text transformations, implemented using the Eclipse Modeling Framework (EMF)², particularly the Acceleo initiative³. After step (3), if the designer finds that the model is not yet completed, the process returns to step (1) for a new iteration.

XIS-CMS also supports two design approaches, the dummy and smart approach, begin the main difference that in the smart approach the framework offers the Model-to-Model transformation tool to create the more complex views: the User-Interface views that represent the interface screens and interactions.

The **goals of this research work** are following:

G1: Design the XIS-CMS Language, a Domain Specific Language that maps the concepts and views needed to the development of CMS modules.

G2: Implement the XIS-CMS Framework, a set of tools that support the language aforementioned with the a Model Editor, Model Validation and generators that create advanced models and source code for one CMS platform.

G3: Evaluate the XIS-CMS Language and Framework, through test sessions and implementation of case studies.

¹ <http://www.sparxsystems.com/products/ea/index.html> (Accessed in October 2015)

² <http://www.eclipse.org/modeling/emf/> (Accessed in October 2015)

³ <https://eclipse.org/acceleo/> (Accessed in October 2015)

1.4. Context

This research work has been conducted at the Information and Decision Support Systems⁴ of INESC-ID (Instituto de Engenharia de Sistemas e Computadores – Investigação e Desenvolvimento) under the supervision of Professor Alberto Rodrigues da Silva, regarding the Master Degree in Information Systems and Computer Engineering at Instituto Superior Técnico. This research work results from the common interest in the area of Model-Driven Development and its application to Content Management System context.

One initiative of the Information Systems Group in the area of Model-Driven Development is the project MDDLingo⁵, before knowned as ProjectIT (Silva A. R., 2004). MDDLingo affirms that Project Management (PM), Requirements Engineering (RE) and design activities should be main focus of an Information Technology project. With the most effort being applied in these aspects, some tasks such as software programming that meets known patterns or testing patterns, should be automated by using Model-Driven Development, releasing time for more complex tasks. MDDLingo defends that this approach will improve the quality (less risk of bugs) and productivity of Information Technology projects, and for that, was created a set of languages and tools:

- ProjectIT-RSL and RSLingo (Ferreira & Silva, 2012), Requirements Specification Languages (RSL) to specify and validate requirements of information systems;
- CMS-ML (Saraiva & Silva, 2010), a modelling language for Content Management System-based (CMS) web applications development;
- XIS profile (Silva A. , Saraiva, Silva, & Martins, 2007), a Domain Specific Language (DSL) for modelling interactive systems at a Platform Independent Model (PIM) level;
- XIS-Mobile (Ribeiro & Silva, 2014), an implementation that derived from the XIS profile in the context of mobile applications.

Therefore, this research work is intrinsically related to the scope of the MDDLingo initiative and can be even encompassed within it. More specifically, the XIS profile served as a starting point for this research due to the similar goal of using a higher level representation, DSL in the form of a UML profile, to design and enhance the development of software applications (Martins & Silva, 2007). Also, the XIS-Mobile served as a reference, due to the successful implementation in the context of mobile applications. The result of this research work is the solution propose as XIS-CMS. XIS-CMS defines and implements a DSL and the tools framework to apply the Model-Driven Development approach to create CMS modules. XIS concepts as well as the relation with this work are described in more detail in Section 3.1.

⁴ <http://www.inesc-id.pt/laboratory.php?lab=IDSS> (Accessed in October 2015)

⁵ <https://github.com/MDDLingo> (Accessed in October 2015)

1.5. Methodology

This dissertation research work was developed using the Action Research methodology (Lewin, 1946), a process that defines a cycle with five phases as showed in Figure 2. The five phases must be performed in the specified order and are the following:

- 1) **Diagnosing**: identify and define the problem domain.
- 2) **Action Planning**: plan the actions that solve the problem identified in the previous phase.
- 3) **Action Taking**: implement the actions defined in the previous phase.
- 4) **Evaluating**: gather information of the resulting solution and if had success in solving the problem.
- 5) **Specifying Learning**: identity the lessons learned and prepare the next iteration.

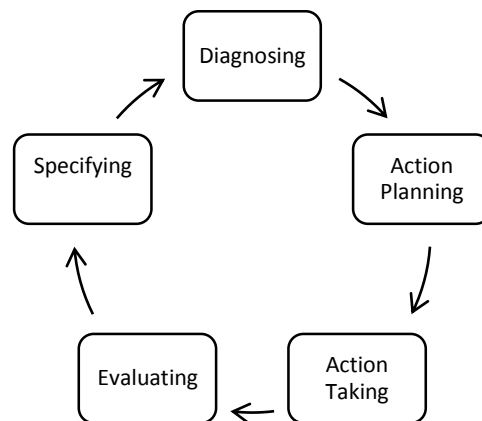


Figure 2: Action Research methodology cycle phases

The research work has been conducted in thirteen months, with the following three iterations:

Iteration 1: September to December 2014

The first iteration defined the initial concepts of the XIS-CMS Language and a case study.

The Diagnosing phase was the review of previous related works, mainly in the fields of Content Management Systems, Model-Driven Development, with particularly focus on the works of the Information and Decision Support Systems projects.

The Action Planning phase was the definition of the organization and concepts of the language and also the case study specification.

The Action Taking phase was the specification of the XIS-CMS Language in the Enterprise Architect (EA) and the manual implementation of the case study to the CMS platform DotNetNuke.

The Evaluating phase was using the XIS-CMS profile defined in the EA to modelling the implemented case study and comparison of the views.

The Specifying Learning phase revealed that the case study could map the generality of the concepts, but that the XIS-CMS language needed other view, such as the toolkit view, and a different multi-view organization.

Iteration 2: January to May 2015

The second iteration consisted of improve the XIS-CMS Language and the initial implementation of the XIS-CMS Framework, namely the Model-to-Text transformations.

The Diagnosing phase was the analysis of the issues and problems learned in the previous iteration and research about the Model-to-Text transformations, primarily in the Acceleo technology.

The Action Planning phase was the definition of improvements to the XIS-CMS language and how the Model-to-Text transformation should be performed.

The Action Taking phase was the refinement of the XIS-CMS Language and the implementation of the Model-to-Text transformations, using the Eclipse IDE, Eclipse Modelling Framework and Acceleo.

The Evaluating phase was performed by using the XIS-CMS Framework tools of the Model Editor and Model-to-Text transformation to model the case study problem and generate the source code solution and files. This solution was compiled and deployed in a test environment.

The Specifying Learning phase revealed a partial success on implementation the case study, with the successfully implement of modules whose entity had simple structure and relationships. For the more complex modules it still have some bugs when executing the resulting code. Also, it revealed the need of general configurations to map the data repository and permissions support.

Iteration 3: June to September 2015

The third iteration consisted of refining the XIS-CMS Language and complete the implementation of the XIS-CMS Framework, improving the Model-to-Text transformations and implementing the Model Validator and Model-to-Model transformations. Also, to better evaluate the performed research work, was defined a new case study and realize test sessions with diverse users.

The Diagnosing phase was the analysis of the issues and problems learned in the previous iteration.

The Action Planning phase was the definition of improvements to the XIS-CMS language, the definition of the Model Validation and how the Model-to-Model transformations should be performed. Also the Model-to-Text transformations needed to improve to support the more advanced patterns.

The Action Taking phase was the refinement of the XIS-CMS Language and the implementation of the Model Validation and Mode-to-Model transformations using the EA's Automation API. Also the improvement of the Model-to-Text transformations to generate source code ready to compile, which support the CRUD operations to entities.

The Evaluating phase was the test session with users of a secondary case study to observe the implementation of three CMS modules with variable complexity.

The Specifying Learning phase was the analysis of results from the test session, in the format of an online survey. These results and conclusions and detailed in chapter 6.

1.6. Outline

The remainder of this dissertation is organized as follows:

Chapter 2: This chapter presents the main concepts that served as the basis to this research work, namely Content Management Systems applications and Model-Driven Development.

Chapter 3: This chapter presents an overview of XIS-CMS, the approach used in this research work.

Chapter 4: This chapter presents the XIS-CMS Language and its components.

Chapter 5: This chapter presents the XIS-CMS Framework, the supporting tool that applies the Model-Driven Development to the XIS-CMS Language

Chapter 6: This chapter presents the evaluation performed on the XIS-CMS approach.

Chapter 7: This chapter presents the conclusions of this research work and some possibilities for future work.

2. Background

This chapter presents the main concepts that served as basis to this research work. The focus is on two topics: Content Management Systems (CMS) and Model-Driven Development (MDD).

Section 2.1 describes the main concepts of Model-Driven Development. It presents concepts like modeling languages, domain specific languages and Model-Driven Engineering.

Section 2.2 describes the main concepts of Content Management Systems. It presents the CMS structure, module development and management.

Section 2.3 describes the cross-platform approaches and compares some of the existing tools.

2.1. Model-Driven Development

This section describes the Model-Driven Development approach, and the relation to the Model-Driven Engineering and Model-Driven Architecture. Also, it presents some concepts on Domain Specific Languages and Model Transformations.

Section 2.1.1 describes the Model-Driven Engineering and its relations.

Section 2.1.2 describes the Domain Specific Language concepts.

Section 2.1.3 describes Model Transformations that are used in Model-Driven Development.

2.1.1. Model-Driven Engineering

Model-Driven Engineering (MDE) is a methodology that focuses on designing the problem domain, components and all the topics in models, being these the primary entities to the process. The MDE applies to many context besides software development, through the creation of models to activities, process or requirements. (Schmidt, 2006)

One more concrete initiative is the Model-Driven Development (MDD), which focuses are the MDE methodology subset of models that represent the development process. The MDD approach affirms that the models used during the development of the software system are the primary artefacts, and the use from the design, through code generation and deployment, until the maintenance will improve the quality of the system by provide an environment less error-prone and more productive. (Kuhn, Gotzhein, & Webel, 2006)

Model-Driven Architecture (MDA), from Object Management Group (OMG), is the concrete MDD initiative that proposes a set of guidelines for structuring specifications expressed as models. These guidelines aim to develop models that describe the system at all the levels of abstraction: functionality and implementation.

MDA describes two models, the Platform Independent Model (PIM) and the Platform Specific Model (PSM). A PIM provides a conceptual model of the system that is independent of the technological

platform that may support the functionalities. A PSM provides a concrete model of the system, with concepts and details particularly relevant to the chosen technological platform. These models can be inter-translated between PIM and PSM through the use of Model-to-Model (M2M) transformations and ultimately to the technology support, such as source code files, through the use of Model-to-Text (M2T) transformations. (MDA, 2015)

Figure 3: Relation between Model-Driven Initiatives Figure 3 shows the relations between MDE, MDD and MDA.

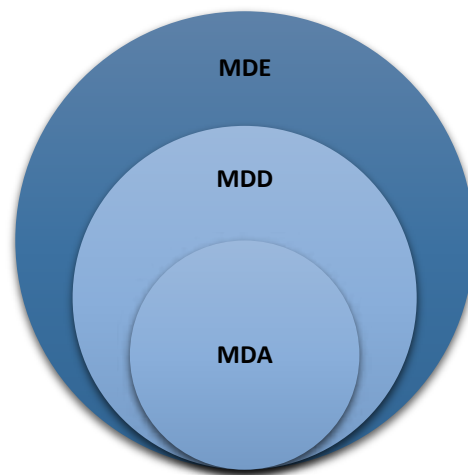


Figure 3: Relation between Model-Driven Initiatives (Ameller, 2009)

Applying a MDD approach, following the MDA standards, an Information Systems development can benefit of various advantages such as:

- Focuses on the models representing the business, instead of technical issues;
- Multi-view that can unite technical and non-technical stakeholders;
- Development task, such as code programming are automated, fully or partially.
- Domain models are faster to produce and can be easier to reuse.
- Standard notations regarded of the supported platform.

2.1.2. Domain Specific Languages

When applying a MDD approach to a specific context, the use of a Domain Specific Language (DSL) is essential. DSL is a specification language that characterize the concepts of a particular problem domain, through the definition of notation and abstractions (Deursen, 2000). A DSL surges from the understanding of the problem domain and an awareness of how to specify the representation in the abstract concepts, either in textual or graphical form.

By using a DSL, programs can be written in high-level domain concepts rather than in low-level implementation concepts narrowing the differential levels of understanding between domain experts, analysts or programmers. DSL also bring several advantages, such as improved development productivity, portability and overall knowledge of the domain concepts (Kiebertz, McKinney, & Bell, 1996).

A UML profile is a mechanism to produce DSL, and are supported by many UML tools (OMG, 2011). UML profiles can customize UML models to a specific domain, extending from the UML metamodels, adding new concepts in the form of stereotypes. A UML profile contains the following components: (1) Stereotypes that extend the metaclasses; (2) Tagged Values that represent a meta-attribute; and Constraints that restrict the use of some relationships.

UML profiles are an excellent tool to define a DSL used in a Model-Driven Development approach, due to the ability of extension, allowing the reuse of one profile to refine the DSL to a more specific context (Giachetti, Albert, Marin, & Pastor, 2010).

2.1.3. Model Transformations

In a Model-Driven Development approach, model transformations are an important component to increase the productivity and reduce errors, by automating steps that are compliant to a pattern. Following the MDA guidelines, there are two model transformations: (1) Model-to-Model (M2M) and (2) Model-to-Text (M2T) transformations (Czarnecki & Helsen, 2003).

Model-to-Model Transformations

Model-to-Model (M2M) transformations takes a significant role in MDD approach to reduce effort wasted on created the most laborious models. M2M is a transformation process that generate models from other models. For example, a M2M transformations can generate a model of requirements or operations from a use cases model (Brambilla, Cabot, & Wimmer, 2012).

One M2M transformation is the Query-View-Transformation (QVT). QVT is the proposition by the OMG (OMG, 2011) to a M2M standard transformations. QVT are composed by three languages based on the Meta Object Facility metamodeling approach (OMG, 2014):

- QVT-Relations, that defines relations between the metamodels of source and target of the transformations, in a textual or graphical form.
- QVT-Operational, which specifies unidirectional transformations in a textual concrete syntax.
- QVT-Core, which is an internal language used by the other two languages.

Model-to-Text Transformations

Model-to-Text (M2T) transformations takes a crucial role in MDD approach by providing mechanisms that reduce the most time-consuming task of producing boilerplate code or standardized documents. Therefore, M2T is normally the process that creates the base source code of an application, and as such, need to be well selected and consistent. There are two approaches to M2T transformations: the visitor and the template approach (Czarnecki & Helsen, 2003).

The visitor-based approaches are used by course the model representation and output the code via text stream. The common approach is implementing via an API and a visitor mechanism based on the Visitor design pattern (Gamma, Helm, Johnson, & Vlissides, 1995). An example of a visitor-based approach is Jamda, which contains a visitor mechanism to generate code, named CodeWriters (Czarnecki & Helsen, 2006).

Template-base approaches are used by executing a set of templates that are organized in a hierarchy which create source code files in a text format. Since the approaches are very similar to an application, the templates are executed by a template engine, which run and interprets the templates. The templates are consisted by verbatim text and annotations that form a pseudo programming language. An example of a template engine is Aceleo, used during this research work.

2.2. Content Management Systems

Content Management Systems are software platforms, used to manage and publish content. The management of content is traditionally done by way of CRUD operations adapted from the normal applicability to database systems (Martin, 1983). CMS abstracts the technical features concerning the development and management of web applications defined as the orchestration of multiple web pages and contents. CMSs are supported and composed of web modules. A CMS module consists in a collection of code and resource files organized in a library that adds new features to the CMS framework and can be instantiated in the pages of a website or web application. CMS allows users without technical knowledge to create and maintain content in a quickly and easily way, helping them, through the system, to manage content organization and visual appearance.

A website is mainly composed of static content that needs to be maintained by a developer or webmaster. On the other hand, a *web application* provides a dynamic experience to the user focusing on his interaction to determine the content that is displayed. A CMS platform supports web applications development and management, because it provides several features to manage the structure, content and presentation of these applications. CMSs are usually used as enterprise portals and web applications that have both static and dynamic content that need to be constantly updated (Boiko, 2001). In a CMS, there are several processes involved in the creation and maintenance of the content:

- Authoring, to create content;
- Acquisition, to acquire information;
- Conversion, to filter the content, created or acquired, from the superfluous layers of information and translate it into a specific mark-up language;
- Aggregation, to separate content into components to which is assigned a tag, so as to be able to insert that content in the chosen metadata system;
- Collection services, programs and functions that support the collection system.

In a CMS approach, content is the most important concept and is the managed entity, and everything in the application focuses in content and content management (Boiko, 2001).

With the use of the process aforementioned, CMS offers a variety of advantages to the users. Because of these advantages and the future perspective for CMS-based web applications, the CMS platforms are becoming crucial to an organizational technical structure (Addey, Ellis, & Suh, 2002). Some of the benefits are the following:

- Ease of use: the creation and management of the content only requires knowledge of the domain context, and do not require the technical knowledge, such as programming code to web applications.
- Management of users and permissions: traditionally, CMS have built in management modules to administer roles, users and access to content.
- Separation of content and visual aspect: through the use of website templates or content template the visual aspect of the content can be easily changed without the need of change the content.
- Extensions: CMS offers an extensibility ability to install new plugins or toolkits. These plugins can offer custom modules with functionalities to a particularly context.
- Reduced time-to-market: by offering a ready to use portal structure, a web application can be instantiated in less time that if was manually created by programming.

The extension functionality in a CMS platform is crucial to enhance the period of support to an application, by providing a mechanism to add and upgrade existing extenders. An extender can be one of the following: a module; a toolkit, which contains a set of modules; or a plugin, which adds some functionality to the CMS portal, such as multi-language or other transversal component (Saraiva & Silva, 2008).

2.2.1. CMS Structure

Content Management Systems are platforms for web applications, and traditionally can support several websites, or portals, in the same platform instance. Each portal is an independent website which its one content. Some CMS platform may share common entities like users, unifying the authoring identification (Carmo, 2006).

A website, supported by a CMS, is structured in multiple levels with several components: navigation menus, pages, templates, modules. The navigation menus follow a hierarchy structure with menus and sub-menus. Also, mostly CMS support navigation in the pages through the use of Breadcrumb navigation.

The CMS pages are composed by containers that follow layout templates. These templates change the number and position of the containers, severing the page into sections. The templates can be individual installed or belong to a website template or a theme template (Saraiva & Silva, 2009).

The CMS modules can be instantiate in the containers. Some CMS support showing the same instance of a module in several containers, through reference associations. A module can be a content module or a management module. Management module are responsible for managing an aspect of the platform such as User, Roles or the layout of the page. Content modules are responsible to manage the content that is show to the users.

It is common that the CMS platforms are installed with an initial set of modules built in and ready to use. Some of the management modules are: User, Role, Permission, Page, Languages, Files or Theme modules. In the category of content modules, they are the following: HTML, Announcements, Newsletters, Lists or Journals modules (Souer & Kupers, 2009).

2.2.2. CMS Platforms

To support a Content Management System, several alternatives are known, either corporative software, open-source software, or in some cases both of them. Below, it is presented some of these platforms with a brief description.

DotNetNuke. DotNetNuke is open source Content Management System Platform supported by the Microsoft .Net Framework. The web application offer built-in functionalities to manage the structure of the website, by customize the menus and pages and instantiating modules. It offers, in a very easy way to use, modules to manage users, roles and the corresponding permissions. Being supported by well-maintained tools, it provides performance process to apply workflows and Search Engine Optimization. Its extensibility ability provides enhanced customization and integration with third-party services. DotNetNuke, although being open-source, is supported by a corporation, which makes it well maintained and with easy access to full technical support. Also, it have a great and active community in online forums (DNN, 2014).

Drupal. Drupal is a CMS system based on open-source technologies such as PHP and MySQL and focuses in collaborative content management between individual and groups of users. This collaboration is done through the production, collection, share and discussion of new information and ideas. With this crucial focus, Drupal is a central interest within communities of online information systems that appreciate the collaborative nature. Drupal offers high scalability, functionality, layout and design customization, module customization beyond the traditional aspects of a CMS, users and roles management and built in basic modules to quick assemble a web application. Drupal principles focus are: being modular and extensible; quality coding; standards-based; low resource demands; open-source; ease of use; and collaboration (Drupal, 2015).

Joomla. Joomla is a CMS platform developed by Mambo. Joomla have an easy to use user interface that offers great functionality in configuring the interface layout and design, even for a non-technical user. For this reason, Joomla gained an extend popularity. Joomla is supported by PHP, MySQL database system, and a web server, such as Apache or IIS. Also, the extension functionality of Joomla, and the many available extensions, makes it possible to deploy a very complete system, from chat rooms, to online auctions, to classified ads, to inventory management. The application development is likely very easy, simple and intuitive for basic features and basic management (Knauss, 2008).

Wordpress. Wordpress is the successor of a blogging tool named b2/cafeloga and the main focus was on provide blogging tools and easy to create blogging websites. With the introduction of plugins, and being open-source, the framework grew and transformed in a fully pledged Content Management System. Its supports authoring, with spell check and other editing transversal tools, administration modules and themes customization, allowing to change the layout and design, without update the content.

2.3. MDD Approaches for CMS

Although the idea of using MDD approaches to develop web applications is not new, it is not widely applied to the domain of CMSs and therefore is still not fully explored and challenging. There were some attempts to apply a MDD approach showed in Figure 4 with the classification of the frameworks and methodologies from the survey (Silva A. , Model-driven engineering: A survey supported by the unified conceptual model, 2015). The approaches are described briefly below:

JOOMDD. JOOMDD is an implementation for a specific CMS platforms, which apply a reverse-engineering method to obtain the model, for the Joomla CMS platform (JooMDD, 2015). JOOMDD has two components: one for developing extensions and another for managing the web application structure. Despite it achieved some success, it is a platform-specific proposal not fulfilling our platform independence goals (Priefer, 2014).

CM Data Architect. The CM Data Architect and the Web Application Architect are tools developed by IBM resulting from the attempt to implement a MDD approach to the IBM Content Management System. These tools separate the views from the architect (to manage the entities involved and the relationships between these entities) and the user interface designer (to create and manage the UI artifacts). Although it performs M2T transformations to accelerate the generation of source code, it does not use M2M transformations to generate the user interface view. Thus, the user interface designer must manually develop it (Deshpande, McNichols, Richmond, Srinivasan, & Zbarsky, 2005).

Web Specific Language. The Web Specific Language (WSL), developed in the University of Oxford, adopts MDD to generate web applications from models with a focus on the data and workflows (based on workflows engines). WSL is a textual language that uses syntax-to-metamodel and M2T transformations engines to create the artifacts needed to run the application. These engines were implemented using the EMF. Although WSL is an easy and comprehensive language due to the closeness to natural language, it lacks the visual modeling tools. Furthermore, despite adopting a platform-independence approach, it is more oriented to applications focused on integration and automatic services than on web applications with user interaction (Svansson & Lopez-Herrejon, 2014).

CMS-ML. CMS-ML approach is based on two modeling languages: CMS-ML and CML-IL, which reside at different levels of abstraction, and therefore have different notations and purposes.

CMS-ML describes a web application in a high-level and platform-independent way using a graphical notation. It is oriented for non-technical stakeholders and aims to allow them to quickly and easily model their applications. CMS-ML is organized in three major types of views (Saraiva J. , 2013):

1. WebSite Templates;
2. Toolkits;
3. WebSite Annotations.

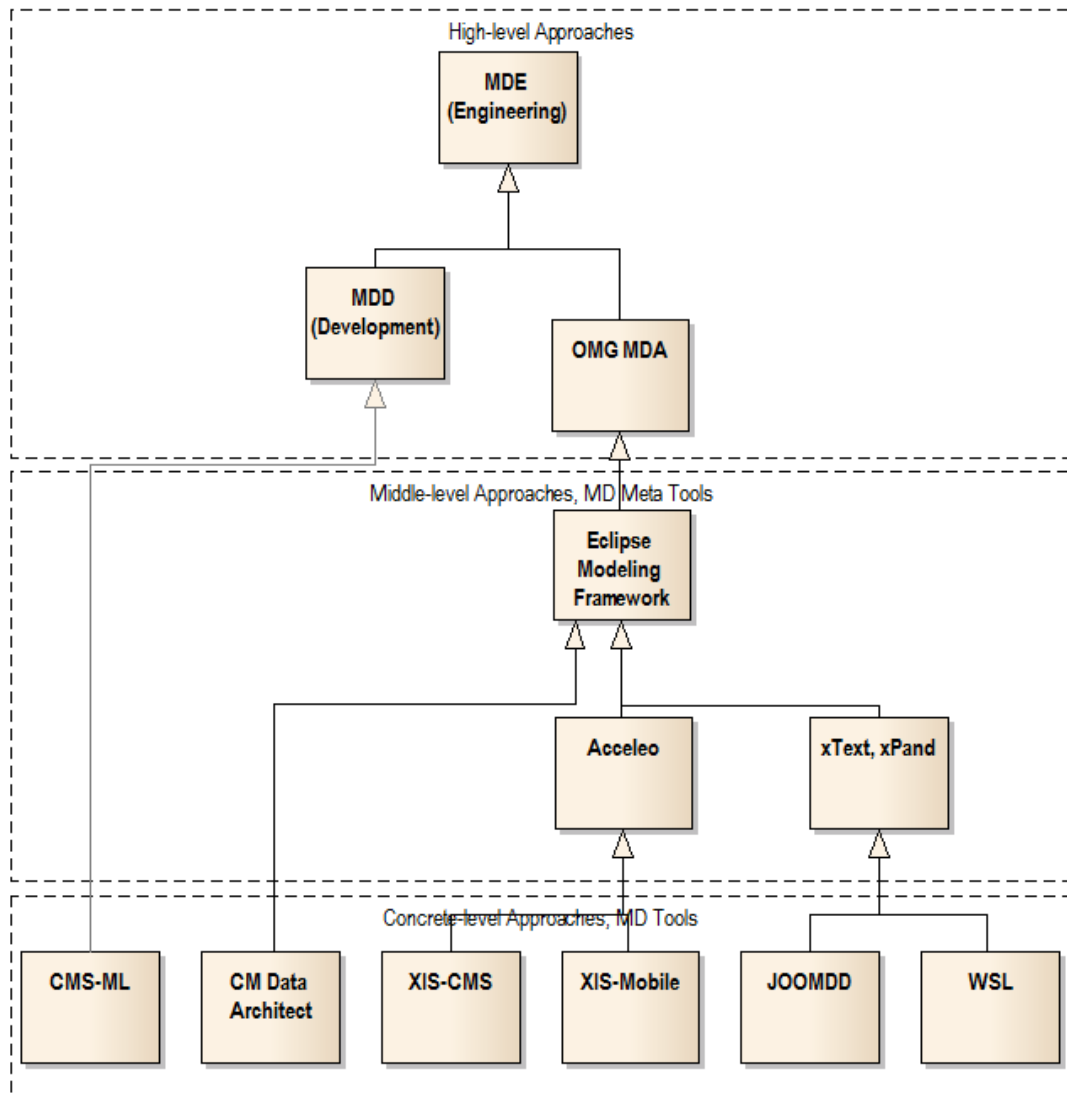


Figure 4: Methodology and frameworks classification

First, a WebSite Template (or simply Template) defines the structure of the application. It comprises the following views: (1) Structure view, which specifies the structural components of the web application: WebSite, Dynamic WebPage, Container and WebComponent; (2) Roles view, which defines the responsibilities of the web application expects its users to assume; and (3) Permissions view, which defines which roles have access to each Dynamic WebPage and WebComponent.

Second, a Toolkit allows extending the set of CMS elements that are available for modeling a WebSite Template. A Toolkit can also reference other toolkits. A Toolkit is composed of the following views: (1) Roles view, which represents the roles responsible for performing the Toolkit's tasks; (2) Tasks view, which represents the user tasks that the Toolkit should support; (3) Domain view, which represents the domain entities and their relationships underlying those tasks; (4) States view, which depicts the lifecycle of the entities manipulated by the tasks; (5) WebComponents view, which specifies the WebComponents that support the tasks; (6) Side Effects view, which enables the definition of desired side effects, or actions, for the modeled user tasks and WebComponents; (7) Interaction Access view,

which defines what Roles can access each WebComponent; and (8) Interaction Triggers view, which establishes what WebComponents trigger the execution of a given user task.

Third, WebSite Annotations (or just Annotations) allow decorating the elements of a WebSite Template by means of specifying CMS-specific properties (e.g. configuration settings).

In turn, CMS-IL is a textual language used for specifying computational aspects of the web application that could not be captured by CMS-ML. It uses low-level concepts, but also independent of any particular CMS framework. CMS-IL provides a common ground for the development of higher level languages, like CMS-ML, at a near-implementation abstraction level. For that reason, it is mainly intended for developers. A developer can design web applications using a high-level language (CMS-ML), then “compile it” to a lower-level language (CMS-IL) representation through model-to-model transformations (“ML2IL”), and finally generate source code or use the created models as input to a “CMS Model Interpreter” component that will handle their runtime execution (Saraiva & Silva, 2010).

3. XIS-CMS Approach

This chapter presents an overview of the approach developed in this dissertation, known as XIS-CMS approach. XIS-CMS approach produced two components: a DSL, known as XIS-CMS Language and a set of tools, known as the XIS-CMS Framework. The XIS-CMS Language define the concepts used to defining the CMS modules. The XIS-CMS Framework supports the MDD approach to develop CMS modules using the XIS-CMS Language.

Section 3.1 presents the background from which the XIS-CMS approach derives, the XIS profile.

Section 3.2 provides an overview of the XIS-CMS Language: the views constituting the profile and their dependencies between each other.

Section 3.3 describes the design approaches that can be used while developing using the XIS-CMS approach.

Section 3.4 presents the case studies applications used during this research work and that are used to demonstrate some concepts of the language and the framework.

3.1. Background

This section presents the background from which the XIS-CMS approach derives, the XIS profile. XIS served as the basis of this research work to the development of the DSL and framework to support CMS modules development.

The research work presented in this dissertation is implemented as an extension of the DSL named XIS (eXtreme modelling of Interactive Systems) (Silva A. , Saraiva, Silva, & Martins, 2007). This DSL is defined as a UML profile that focus on the design of software systems with Platform-Independent Models (PIM) and apply a MDD approach do generate other models and artefacts. This was used in the ProjectIT approach mentioned above.

XIS is constituted by three major views: the Entities, UserCases and User-Interfaces views. The Entities View is organized in two views: the Domain and BusinessEntities views. The Domain View represent the entities and relationships of the context domain. Also, the Domain View can define attributes and enumerations used in these entities. The BusinessEntities View represent a more high-level entities that are compose by several entities of the Domain View. These higher-level entities are used in the use cases to indicate the item that is manipulated. The UserCases View is composed by the Actors and the UseCases views. The Actors View represent the entities that can initiate an action in the system. The UseCases represent the action over the BusinessEntities that the actors can perform. The User-Interfaces views are organized in the InteractionSpace and the NavigationSpace views. The InteractionSpace represents the user-interfaces that the user interacts, the graphical elements with the attributes that permit configure the layout and access. The NavigationSpace view present the navigation flow between the InteractionSpace elements.

There are two modelling approaches defined by the XIS profile: the dummy and the smart approach (Silva A. , Saraiva, Ferreira, & Silva, 2007). In the dummy approach, the designer must create and configure all the elements of all the models. In the smart approach, the designer only need to implement the Domain, BusinessEntities, Actors and UseCases view, and generate the InteractionSpace and NavigationSpace view using Model-to-Model transformations.

An implementation of XIS for mobile applications, the XIS-Mobile, already was developed with success, and demonstrated good practices of adapting the XIS profile to a more specific context (Ribeiro & Silva, 2014).

Although XIS could be extended to collect the context of CMS modules, the current implementation of XIS language was developed in a proprietary tool that has not been maintained. But, because XIS is a UML profile can be implemented in other CASE tools. Following the success of the XIS-Mobile, XIS-CMS is implemented in tools that already given proof of meet the challenges. The XIS-CMS Language and the supporting framework are based on the Sparx Systems Enterprise Architect (EA), the EA plugin API and using the Eclipse Modelling Framework and Acceleo. Also, with these tools was possible to implement the smart approach and Model-to-Text transformations to create the source code artifacts.

3.2. Overview

XIS-CMS proposes a Domain Specific Language, in the form of a UML profile, to design and generate CMS modules in a platform-independent way through a MDD approach. XIS-CMS allows the specification of Platform-Independent Models that represent the diverse views of a module. Then, XIS-CMS also proposes a companion framework that permits the application of Model-to-Model transformations to generate the more time-consuming models and Model-to-Text transformations to generate the source code for the CMS platform DotNetNuke.

As shown in Figure 5, XIS-CMS is organized in a Toolkit View that includes three major views: Entities, Modules and Roles views. In turn, the Entities view contains two sub-views: Domain and BusinessEntities views. The Modules view contains two sub-views: UseCases and User-Interfaces views. At last, User-Interfaces view comprises NavigationSpace and InteractionSpace views. These views depend on each other and that fact influence the design approaches and the transformation stages. The Toolkit View does not depend on any other view as is the aggregator of all the other views. The Domain View does not depend on any other view, since it is the starting point to define the problem domain. In turn, the BusinessEntities View depends on the Domain View, because business entities aggregate domain entities. The Roles View does not depend on any other view, only representing the actor that can perform actions in the toolkit. The UseCases View depends on the BusinessEntities View and the Roles View, since each module definition must be connected to a business entity (specified in the BusinessEntities View) and a role (specified in the Roles View). The InteractionSpace and the NavigationSpace views depend on each other. The InteractionSpace View also depends on the UseCases View, because an interaction space belong to a module. Each one of these views is detailed in Chapter 4.

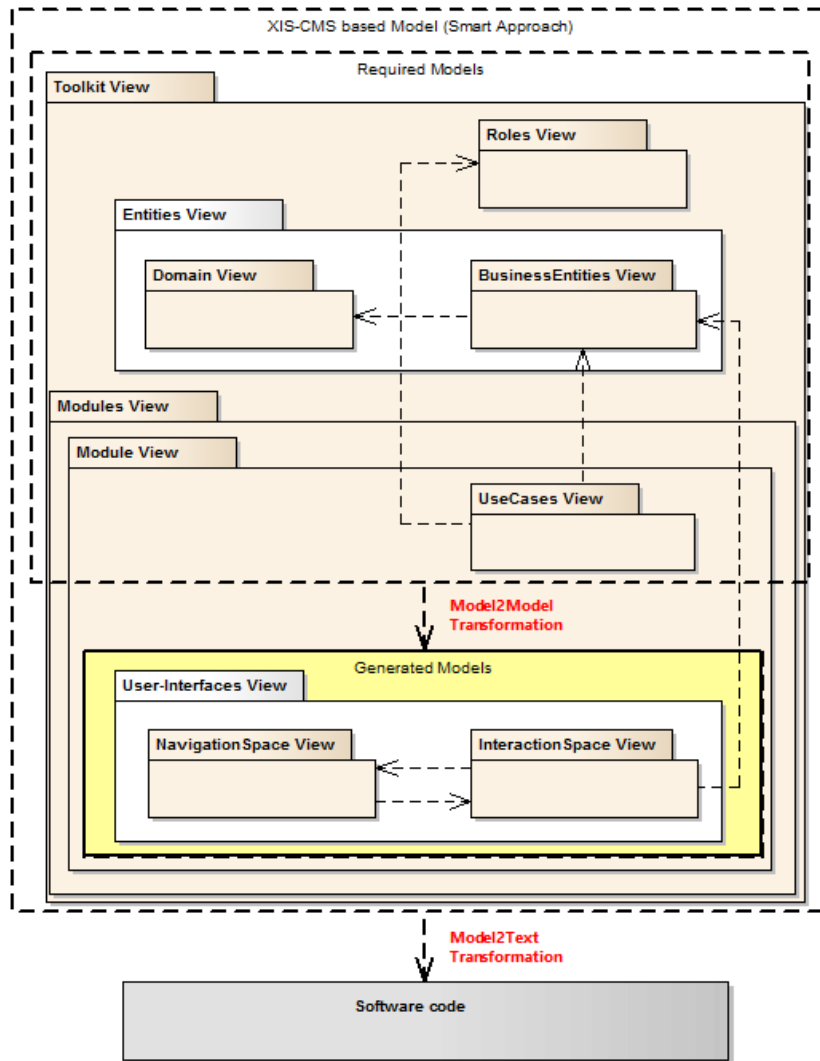


Figure 5: The multi-view organization of XIS-CMS and the transformations process

3.3. Design Approaches

There are two design approaches supported by the XIS-CMS: the dummy approach and the smart approach. The difference between the two design approaches is the use of Model-to-Model transformations in the smart approach, supported by the XIS-CMS framework. By using this kind of transformations, XIS-CMS accelerates the creation of the models of some views, avoiding the manual work that are the most effortful, time-consuming and error-prone: the User-Interfaces Views.

In the dummy approach, all the view must be defined manually by the developer, specifying all the components, their tagged values and even relative positions from each other. On the other side, in the smart approach, the developer only needs to define the following views: Domain, BusinessEntities, Roles and UseCases views. Then, by using the Model-to-Model transformations, the XIS-CMS framework automatically generate the User-Interfaces View: InteractionSpace and NavigationSpace. This transformations applies a pattern consistent with the definitions represented in the UseCase view, namely the module specification. Therefore, this approach can be much less time-consuming than the dummy approach, since the more complex views are automatically generated. This approach should

be used whenever possible, because it speeds up the modeling process and do not remove the opportunity of further customization of the generated models by the developer, if needed. After defining all these view models either using the dummy or the smart approach, it is possible to generate the source code from them using Model-to-Text transformations.

3.4. Case Studies

In the development of this work, two case studies were used to test and evaluate the XIS-CMS approach. In the following chapters, these case studies are used to demonstrate various concepts of the XIS-CMS Language, and support a more practical and simpler explanation.

The Case Study A is a Suppliers management application with two modules to manage respectively the entities Product and Supplier. This case study was used in the first two iterations of the research methodology to define the XIS-CMS language and is presented in this dissertation to exemplify the language in Chapter 4.

The Case Study B is a Medical Clinic application with three modules to manage respectively the entities Physician, Patient and Appointment. This case study was used in the third and last iteration of the research methodology as support to the user test sessions.

The case studies are the following:

Case Study A – Suppliers Management Toolkit

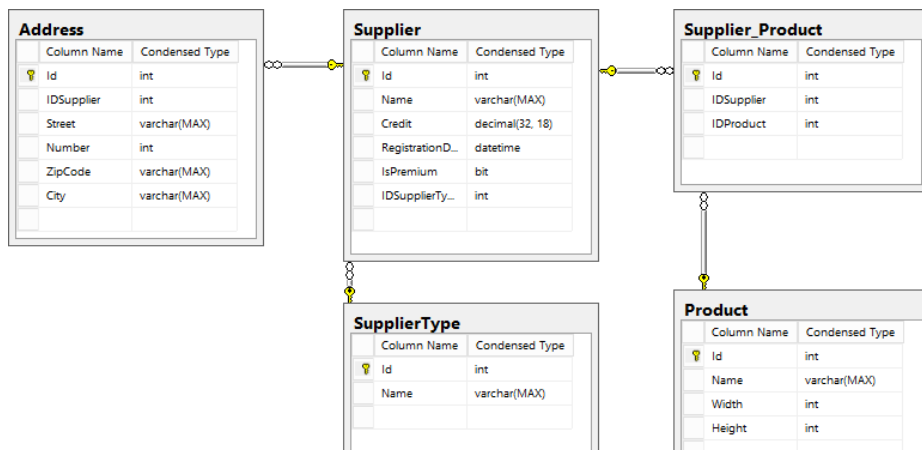
The Suppliers Management Toolkit is a set of modules designed to manage the suppliers and products of a company. A supplier can have a number of addresses and be associated to various products.

The Suppliers Management Toolkit has two roles: the Product Manager and the Supplier Manager.

The following tasks may be performed by each role:

- The Product Manager can create new products and manage it, but cannot delete a product.
- The Supplier Manager can create new suppliers and manage, including the addresses.

The following image shows the database structure:



Case Study B – Medical Clinic Toolkit

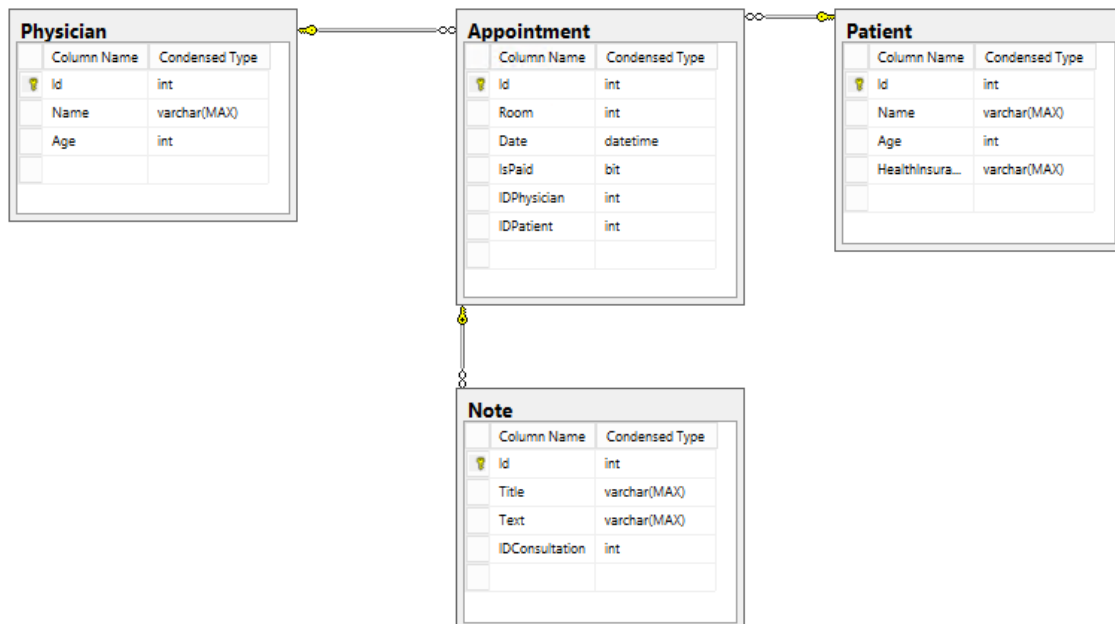
The Medical Clinic Toolkit is a set of modules designed to help manage a Clinic and schedule its appointments. This application has registers of patients and physicians and each record of appointment has one patient, one physician and several notes associated.

The Medical Clinic Toolkit has two roles: the Clinic Manager and the Receptionist.

The following tasks may be performed by each role:

- The Clinic Manager can create new entries of appointments, patients and physicians and manage its records, which includes the notes.
- The receptionist can create new appointments and manage it, including the notes.

The following image shows the database structure:



4. XIS-CMS Language

This chapter presents the XIS-CMS Language and its components. XIS-CMS is a domain specific language, defined in a way so that platform-independent CMS modules can be defined. Some of the concepts of the XIS-CMS language are inherited from the XIS Language, but introduces new views and an appropriate multi-view organization. XIS-CMS models are organized in a Toolkit View that includes three major views: Entities, Modules and Roles views. In turn, the Entities view contains two sub-views: Domain and BusinessEntities views. The Modules view contains two sub-views: UseCases and User-Interfaces views. At last, User-Interfaces view comprises NavigationSpace and InteractionSpace views. Figure 6 shows this structure.

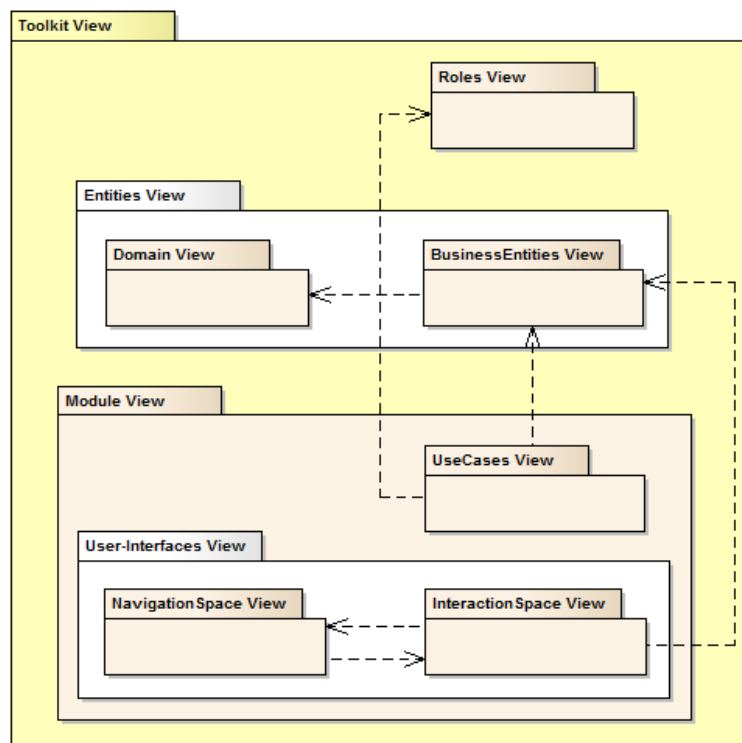


Figure 6: The multi-view organization of XIS-CMS

Section 4.1 describes the Toolkit View and its role.

Section 4.2 describes the Entities View and its components: the Domain and BusinessEntities Views.

Section 4.3 describes the Roles View.

Section 4.4 describes the Module View and its components: the UseCases and User-Interfaces Views. Also, the User-Interface View comprises of the InteractionSpace and the NavigationSpace Views.

Most sections will contain the following structure: a description of the view; an enumeration of the stereotypes; a description of each stereotype; additional information, if applicable; the metamodel; and an example applied to the Case Study A.

4.1. Toolkit View

The Toolkit view includes all the other views and represents the logical aggregation of shared business concepts and modules. The Toolkit view has tagged values to identify the owner of the toolkit, the type of repository data and the connection information to access the repository. These configurations are shared among all the modules defined and to guarantee that the toolkit uses the same data source repository.

Figure 7 shows the toolkit view diagram for the Case Study A. This view provide a general overview of the application. It can be observed the entities in each package, such as the Product and Supplier entities in the Domain View.

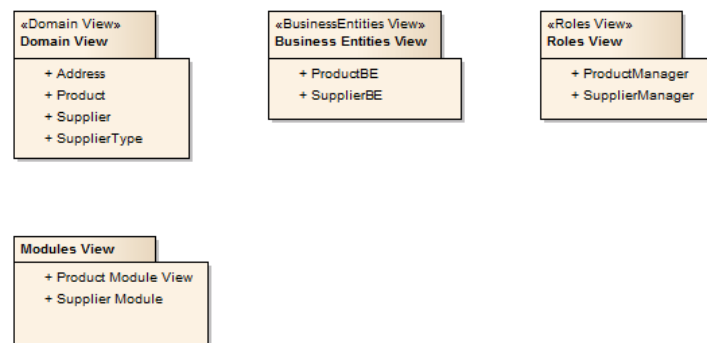


Figure 7: Example of the Toolkit View of the Supplier application.

4.2. Entities View

The Entities view contains the Domain view and the BusinessEntities view, and describes the domain and business entities that represent the concepts of the application context. It also contains their relationships at different abstraction levels. The relationships can be in the domain view, or the between the BusinessEntities and the Domain views.

4.2.1. Domain View

The Domain view defines the problem domain entities commonly captured from a domain analysis. It is possible to specify the attributes of the domain entities and the relationships among them using associations, aggregations or inheritances.

The XIS-CMS language define the following stereotypes in this view: (1) XisEntity to represent the domain entity; (2) XisAttribute to define the entity attribute. A XisEntity can contain one or more XisAttributes; (3) XisEntityAssociation for associations and aggregations; (4) XisEntityInheritance for inheritances; (5) XisEnumeration for enumerations; and (6) XisEnumerationValue for enumeration literals or values.

Some of the stereotypes defined in the XIS-CMS Domain View, such as XisEntity and XisAttribute are used instead of just simple UML classes or Attributes to provide useful information to Model-to-Text transformations.

Each domain entity is represented by a XisEntity stereotype, which in turn contains one or more attributes, defined as XisAttributes stereotypes. Both stereotypes are used, instead of just simple UML Classes or Attributes, to provide useful information to M2T transformations.

The XIS-CMS Domain View metamodel is show in Figure 8.

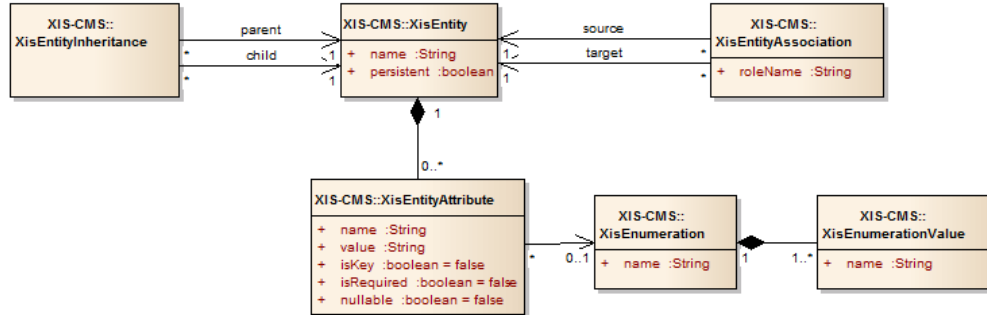


Figure 8: Metamodel of the Domain View.

Figure 9 shows the Domain of the Case Study A, defining the classes as entities, with the relationships between representing the foreign key columns or the many-to-many relationship table.

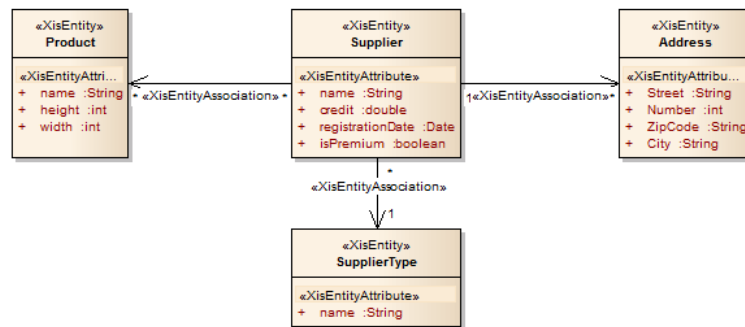


Figure 9: Example of the Domain View of the Suppliers application

4.2.2. BusinessEntities View

The BusinessEntities view defines higher-level entities, called business entities that aggregate one or more domain entities. This BusinessEntity is used to define the concept that the module manages and can interact with.

The XIS-CMS language define the following stereotypes in this view: (1) XisBusinessEntity to represent the business entity; (2) XisMasterAssociation, (3) XisDetailAssociations and (4) XisReferenceAssociation to connect the domain entities with the corresponding category (master, detail or reference).

The XisBusinessEntity stereotype aggregates one of more XisEntities through the associations defined in this view. A XisBusinessEntity must have one, and only one, master XisEntity associated through the XisMasterAssociation, that represent the main entity that will be managed by module using this business entity. The details and reference entities can be associated by the XisDetailAssociation and XisReferenceAssociation, being that a detail entity is a concept that

intrinsically belongs to the master entity, where the reference does not have to. A business entity can have zero to many details and references.

The definition of a master entity restricts the set of detail and reference entities that can be used by a XisBusinessEntity. Both detail and reference entities must be associated to the master entity in the Domain View, respectively, through aggregations and associations.

All the associations defined in this view contain a tagged value named “filter” that allows the restriction of the entity’s attributes that can be used in the context of the respective XisBusinessEntity.

The XIS-CMS BusinessEntities View metamodel is show in Figure 10.

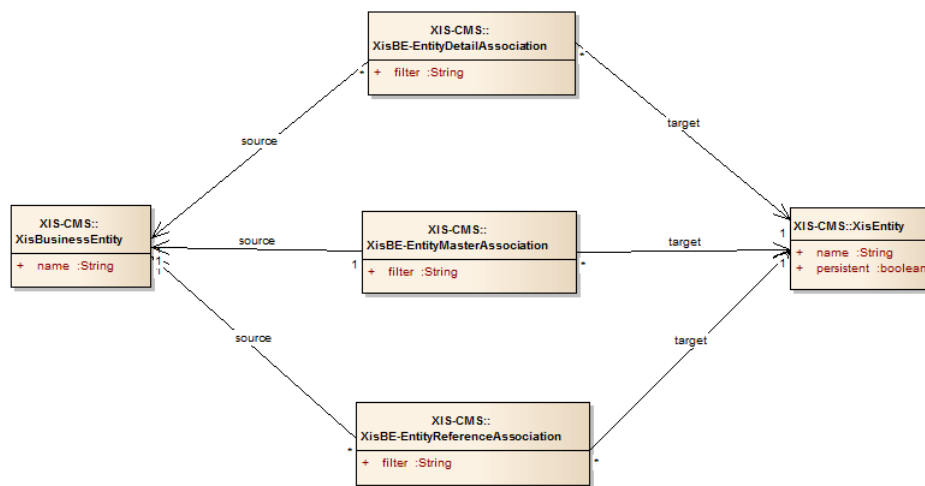


Figure 10: Metamodel of the BusinessEntities View.

Figure 11 shows the BusinessEntities view for the Case Study A. The SupplierBE have a master association to the Supplier entity, a details association to the Address due to the addresses belongs to the entity of the Supplier; and have two reference associations to the SupplierType and Products.

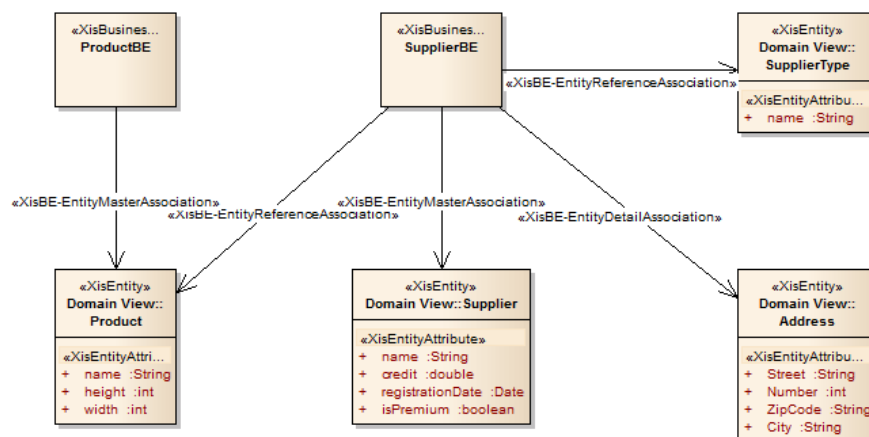


Figure 11: Example of the BusinessEntities View of the Suppliers application.

4.3. Roles View

The Roles view defines the actors or roles that interact with the system and that are specific to the defined toolkit.

The XIS-CMS language define the following stereotype in this view: (1) XisRole to represent the actor that operate the module.

Each role is defined as a XisRole stereotype, and can be organized in a hierarchy of sub-roles. These roles will be instantiated in the CMS as roles or role groups, but during the M2T transformation the system may detect that some CMS standard roles already exist (e.g. Administrators, Registered users) and will just add the corresponding permissions to these roles.

The XIS-CMS Roles View metamodel is show in Figure 12.

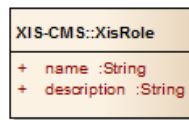


Figure 12: Metamodel of the Roles View.

Figure 13 shows the roles of the Case Study A.



Figure 13: Example of the Roles View of the Suppliers application.

4.4. Module View

The Modules view gives an overview of existing modules of the CMS application. Each module contains a UseCases and User-Interfaces views, which define its management operations over a business entity and its module interface elements (e.g. screens, fields and buttons), respectively.

4.4.1. UseCases View

The UseCases view provides the higher-level information of a CMS module, namely: which data, the business entity, is manipulated; the operations that can be performed; and who, the roles, can perform which operation.

This view is the most important one to process the Model-to-Model transformations through the use of the tagged values of the stereotype XisModule that define which interaction spaces are generated.

The XIS-CMS language define the following stereotype in this view: (1) XisModule to represent the module; (2) XisModule-BEAssociation to connect the module with the business entity that is managed; (3) XisRole-ModuleAssociation to connect the roles and what actions these roles can perform on the

module; (4) XisModuleConfiguration to define specific settings of the module; (5) XisModuleAction to define custom actions.

A module is represented by the XisModule stereotype and its associations. The XisModule contains several boolean tagged values to indicate which of the CRUD operations can be performed to the type of entities (i.e. CreateMaster, ReadMaster and CreateDetail). These operation can be performed on the master, detail and reference entities, and are the more common used in entity management and fit with the View/Edit mode pattern of CMS (Boiko, 2001). A XisModule can contain several XisModuleConfiguration that manage specific settings of the module and can be managed within the CMS platform (DNN, 2015). A XisModule can also contain several XisModuleAction that define additional permission levels that can be used on the generated code to filter access of the module functionalities.

XisModule-BEAssociation serves to connect the XisModule to the business entity. This connection is important for the Model-to-Model transformation generates the InteractionSpace diagrams.

XisRole-ModuleAssociation defines the level access that each role have within the module. It have a tagged value named “actions” that indicate which permissions the role has access. These actions must have the same names that the XisModuleActions concatenated with commas. Following the View/Edit mode pattern, if the association have the action “View” can access to the permission of View, and if the association have the action “Edit” can access to the permissions of Create, Edit, Delete and Update.

The XIS-CMS UseCase View metamodel is show in Figure 14.

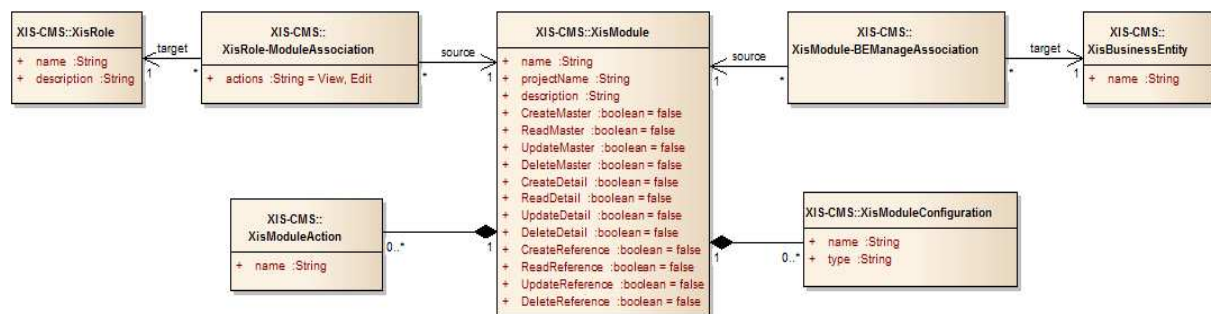


Figure 14: Metamodel of the UseCases View.

Figure 15 shows the UseCase for the Product module. Only the Product Manager can manage the products, and therefore only he have access to this module.

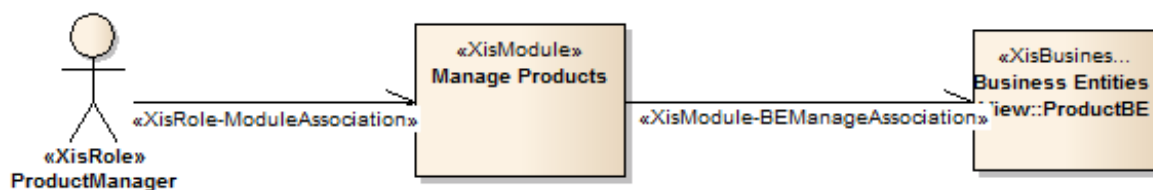


Figure 15: Example of the UseCases View for the Product Module

4.4.2. User-Interfaces View

The User-Interfaces view comprises the InteractionSpace and the NavigationSpace views, which define the UI screens of each module and the corresponding flow between them. This view is part of the Module view, begin that can be on the same diagram as show in the Figure 6.

InteractionSpace View

The InteractionSpace view defines each screen in detail, containing all UI elements and associated events. It is the more extensive view of the XIS-CMS language and makes use of the Composite design pattern (Gamma, Helm, Johnson, & Vlissides, 1995) which allows aggregating multiple UI elements. These elements and their layouts are used on the M2M transformation stage, defining which web components are generated and their hierarchy.

The XIS-CMS language define many stereotypes in this view, so this dissertation will only enumerate the abstract and most relevant. The most relevant stereotypes are: (1) XisInteractionSpace to represent each interface control; (2) XisIS-ModuleAssociation to connect the interface control to the module; (3) XisWidget, a high-level abstract representation of elements existed in the interface control; (4) XisSimpleWidget and (5) XisCompositeWidget, an abstract representation that categorize the elements of the interface.

The XisInteractionSpace can be defined with the “isEntry” and “isDefault” tagged values. Only one XisInteractionSpace per module can have the “isDefault” value set to true, causing the CMS to initiate the module in this screen. But more than one XisInteractionSpace can be defined as an entry screen (if “isEntry” is set to true), allowing the CMS to define multiple views to the same module.

The XisWidget is an abstract definition of the XisSimpleWidget and XisCompositeWidget, and represents a web control on the screen. A XisSimpleWidget can be for instance a button (XisMenuItem), a label (XisLabel), or an input field (i.e. XisTextbox and XisCheckbox). XisSimpleWidget contains two tagged values: “label” to configure the text of the label of the field and “entityAttributeName” to configure the domain entity attribute that the XisSimpleWidget manages. In turn, a XisCompositeWidget is a control, such as a grid (XisGrid) or a panel (XisFieldBoundary), which groups other XisWidgets fulfilling the composite design pattern. XisCompositeWidget contains the tagged value “entityName”, whose value must be an entity contained in the business entity associated to the module. Some XisWidgets can have actions associated with them that trigger events or navigation flows.

Because this view is the more labour-intensive view of the XIS-CMS language, it is recommended to use the smart approach to generate it by a M2M transformation. However, the designer can customize and refine directly the generated views if they do not fulfil the desired requirements.

The XIS-CMS InteractionSpace View metamodel of the stereotypes described above is show in Figure 16.

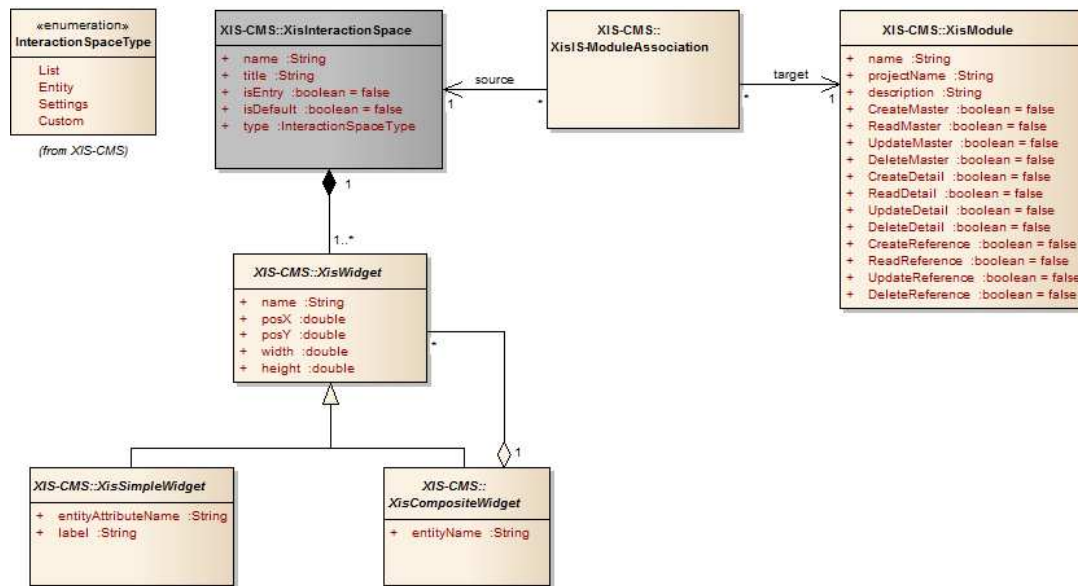


Figure 16: Metamodel of the InteractionSpace View.

Figure 17 shows an Interaction Space for the Product Module. This *XisInteractionSpace* defines the Index screen, the entry screen of this module. It contains a grid with the main details of the entity and two menus. One associated to the line of the grid with the operations that depend of an existent entity, such as View and Edit operations. The other menu is independent with the operations that do not depend of the entity, such as the Create operation.

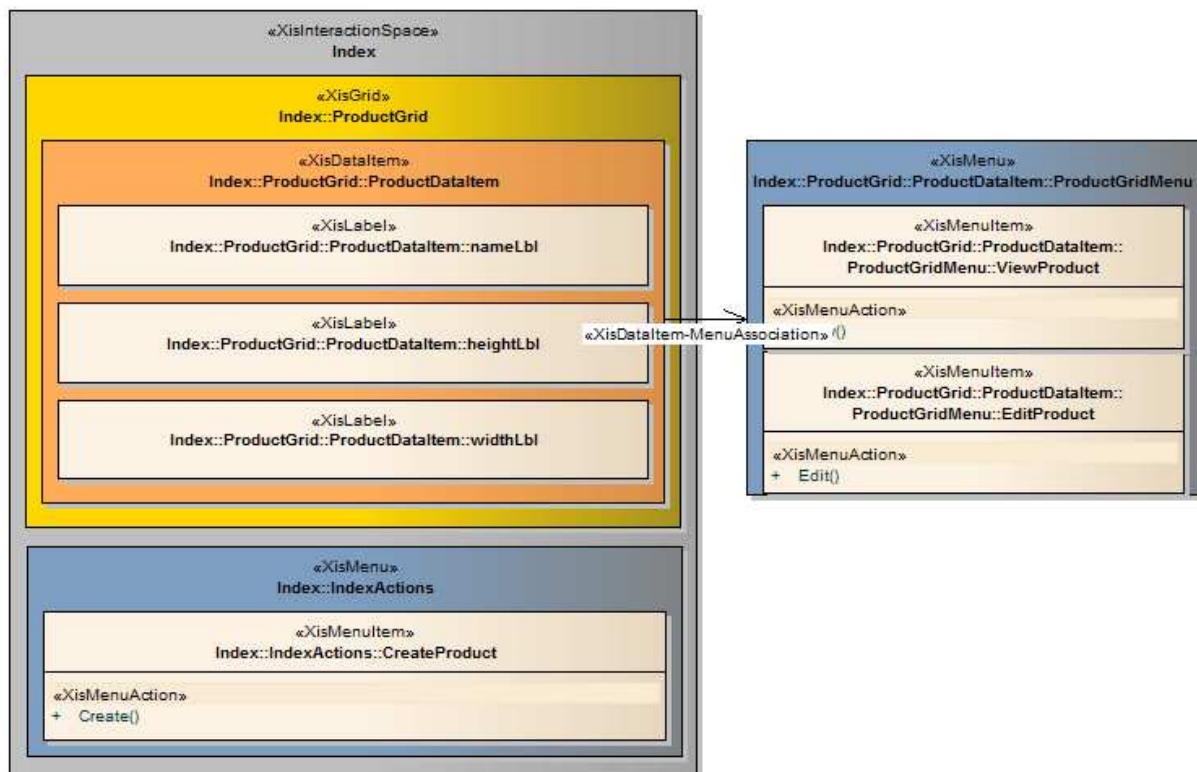


Figure 17: Example of the Index interaction space of the Product module

NavigationSpace View

The NavigationSpace View defines the navigation flow between several screens, represented by a XisInteractionSpace that are accessible to the end-user when interacting with a module.

The XIS-CMS language define the following stereotype in this view: (1) XisInteractionSpaceAssociation to represent the navigation flow between the XisInteractionSpaces.

The XisInteractionSpace Association has the “actionName” tagged value to define the action that triggers the navigation between the screens. This tagged value must be the same as a XisAction defined in the XisInteractionSpace origin of the navigation flow. It also can be a permission that is accessible through the UseCase View XisRole-ModuleAssociation stereotype if it corresponds to a custom XisModuleAction or is one of the standard actions (View, Create, Update, Delete, and Back).

The XIS-CMS NavigationSpace view provides an overview of the structure of a module and its interactions, which is useful both to technical and non-technical stakeholders.

The XIS-CMS NavigationSpace View metamodel is show in Figure 18.

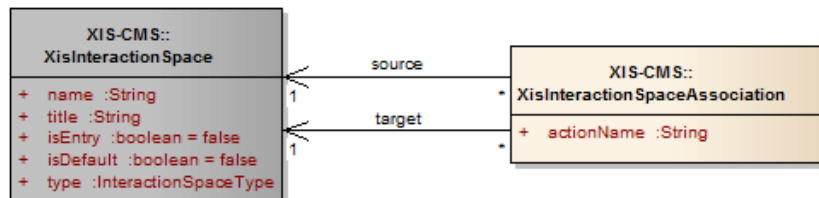


Figure 18: Metamodel of the NavigationSpace View.

Figure 19 shows the navigation flow between the screens for the Product module. The delete action flow is absent, because a product cannot be deleted.

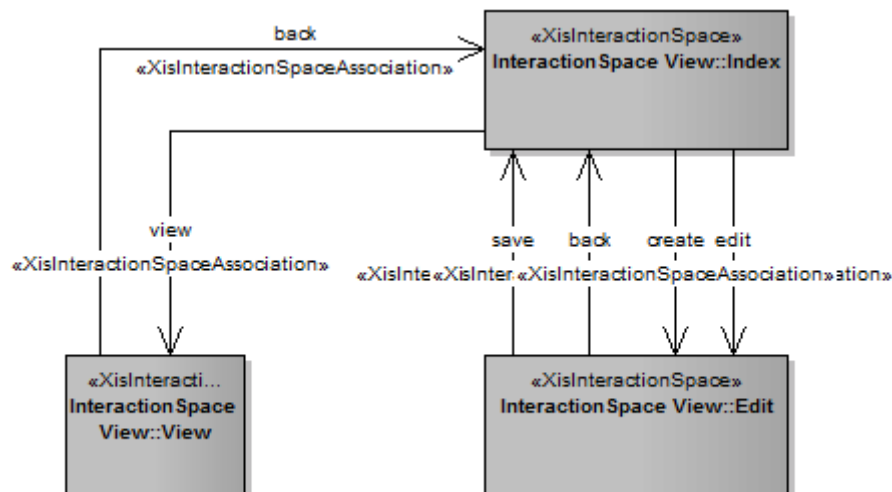


Figure 19: Example of the NavigationSpace View of the Product Module

5. XIS-CMS Framework

This chapter presents the XIS-CMS Framework, the supporting tool that applies the Model-Driven Development to the XIS-CMS Language. XIS-CMS Framework is implemented using the Model Driven Generation (MDG) Technologies provided by Sparx Systems EA along with the EMF (Eclipse Modeling Framework).

Section 5.1 presents an overview of the XIS-CMS Framework, its architecture, technologies and the process to generate the code artifacts.

Section 5.2 describes how the XIS-CMS Framework integrates and customizes the Enterprise Architecture to permit design and generate code with XIS-CMS Language.

Section 5.3 describes the Model validator and its role in the framework.

Section 5.4 describes the Model-to-Model transformations of the XIS-CMS.

Section 5.5 describes the Model-to-Text transformations of the XIS-CMS.

5.1. Overview

A MDD-based framework is the support for the XIS-CMS language and creates an improvement on this proposed DSL, allowing its use and making it more relevant.

The suggested development process of a CMS module uses this XIS-CMS framework taking account the four following phases, as shown in Figure 20:

- (1) The use of Model Editor for the definition of the required views;
- (2) The application of the Model Validator for their validation;
- (3) The use of Model-to-Model transformations for the generation of the User-Interfaces View models;
- (4) The generation of the module's source code through the Model-to-Text transformations.

These phases are automatic, except the Model Editor phase that is manual.

When after the Model-to-Model phase, the designer discover an uncompleted model, the process returns to a new iteration, i.e., to Model Editor phase.

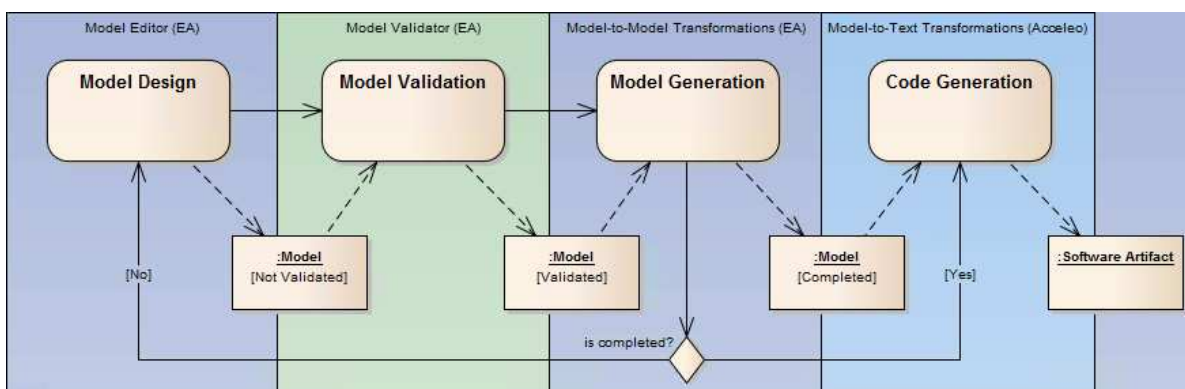


Figure 20: Development process for XIS-CMS approach.

The Model Driven Generation (MDG) Technologies provided by Sparx Systems Enterprise Architect (EA) and the Eclipse Modeling Framework (EMF) are the technologies used for the implementation of the XIS-CMS framework and it leverage the environment they provide and some compatible plugins. The XIS-CMS framework is composed by four modules:

- (1) Model Editor;
- (2) Model Validator;
- (3) Model-to-Model transformations;
- (4) Model-to-Text transformations.

Initially, the implementation of the Model Editor is carried out on top of EA using an MDG Technology plug-in particularly created for the XIS-CMS language.

The XIS-CMS language defined as a UML profile fully compliant with the OMG specification for UML2, and the conception of toolboxes, diagram types, model patterns and model templates customized to this kind of language is due to the use of EA's MDG Technologies.

Second, occurs the implementation of the Model Validator as a plugin using EA's Model Validation API, that have a crucial role preventing errors from the designer, enhancing the quality of the models and, consequently improving the quality of the generated models and code.

OCL is the standard language generally used in the UML models validation (OMG, Object Constraint Language (OCL), 2015). However, the use of OCL was not successful due to several limitations with stereotypes validation and ongoing developments of the OCL plugin for EMF. Therefore, this solution, even not being a standard as OCL, allows the definition of rules or restrictions, custom error messages for each restriction, the assignment of severity levels (error or warning) to them, and the instant navigation to the element that broke the rule and brought the message.

The Model-to-Model transformations is the third to be implemented as a plugin leveraging EA's Automation Interface. It allows the access to the repository where is the created diagrams and their elements, as well as generating new diagrams and elements. Thus, using the information of the Domain, BusinessEntities, Roles and UseCases views, can perform M2M transformations to create elements in the InteractionSpace and NavigationSpace views. Typically a single instance of each of these diagrams yields to multiple InteractionSpace View diagrams and a single NavigationSpace View diagram. Section 5.4 details the process of M2M transformation

Fourth, the Model-to-Text transformations is based on Acceleo, a template-based code generator framework available as an Eclipse plug-in. Acceleo is responsible for the implementation of the MOF Model to Text Language (MTL) standard (OMG, 2008) and allows the definition of code templates for any type of model compatible with the EMF. The code templates are constituted by a regular text (static part of the template) and several annotations (dynamic part of the template) that during the generation time are replaced by values of the model. In this moment, the XIS-CMS framework sustains the generation of modules for the CMS platform DotNetNuke. Is important to note that every time that the user wants add support for other platforms, he shall to define the corresponding code templates.

Additionally, and before the performance of code generation, the models designed in EA are exported in a XMI file, which contains the representation of the models and consists in the input to the Model-to-Text transformations. It was essential to process the exported model before starting the code generation, once that the XMI format used in EA and the XMI format supported by the EMF and consequently expected by Acceleo, are not fully compatible. For this process, is used a Java program that replaces EA's XMI namespaces by EMF's XMI namespaces, establishes the positioning and dimension of each InteractionSpace View element by setting the respective tagged values (posX, posY, width and height of the XisWidget stereotype) using the layout information stored by EA's XMI format. In the end, the additional irrelevant data stored by EA is removed. Setting these tagged values is crucial, once that it is the basis to the generation of the layout source code files.

5.2. Model Editor

As already explained, the Model Editor of the XIS-CMS Framework has been designed considering the maximum advantage of the resources provided by EA Model Driven Generation (MDG) Technologies. The MDG technology particularly created for XIS-CMS has been designed in order to adjust the EA environment to the XIS-CMS language. In addition to the definition of the XIS-CMS profile, the MDG technology that was implemented includes the defined custom diagrams, custom toolboxes, a project template and custom quicklinks. Each component is described below.

Concerning the defined XIS-CMS diagrams, EA permits the definition of custom diagram profiles as its inclusion in a MDG technology. For each type of diagram is provided specific metaclasses by this custom diagram profile (e.g. the metaclass Diagram_Logical represents a Class diagram). It is allowed too the association between custom toolboxes and diagrams. The Figure 21 shows the Profile diagram of the XIS-CMS's diagrams.

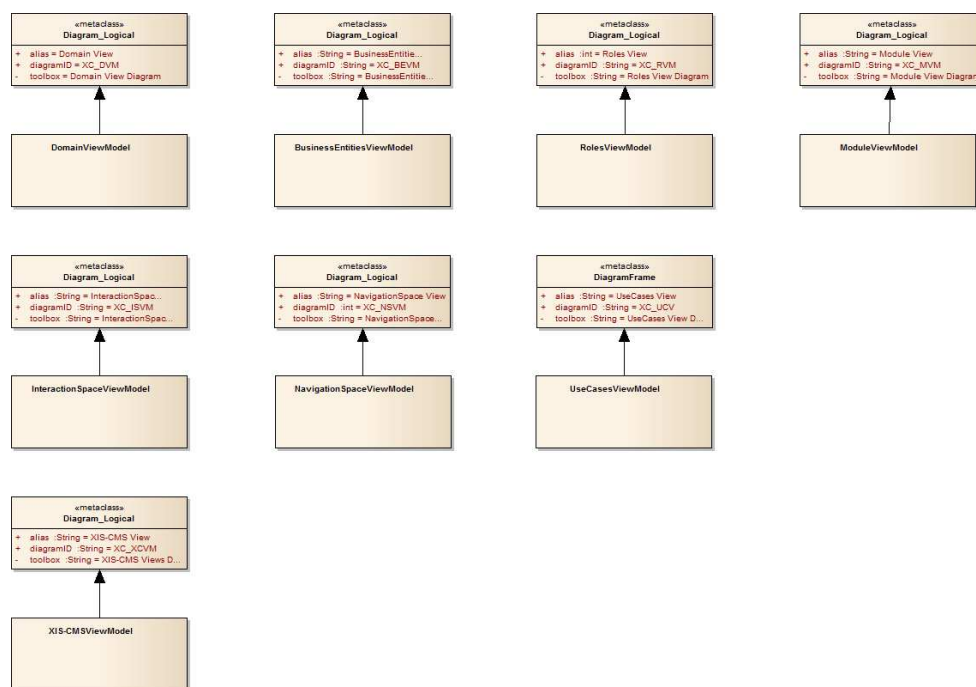


Figure 21: Profile definition of the XIS-CMS diagrams.

The creation profile diagrams also allowed the implementation of the XIS-CMS diagrams toolboxes. For that, eight profile diagrams were created (one for each view of XIS-CMS and to manage the packages) and after included in the implemented MDG technology. EA supplies a custom toolbox which includes the metaclass `ToolboxPage` which allows the definition of the elements and associations of a custom diagram toolbox. Additionally, it is possible associate icons to specific toolbox items using the metaclass `ToolboxImageItem`.

It is also possible create models of project templates using EA MDG Technologies. After the addition to the MDG technology, these templates can be selected through the list of standard model templates that was provided by EA. For XIS-CMS, the model template where it was created, is also responsible for the creation of the package structure with all the views and respective diagrams of a XIS-CMS project with a module ready to be used. Figure 22 represent its use and the project structure is highlighted in red.

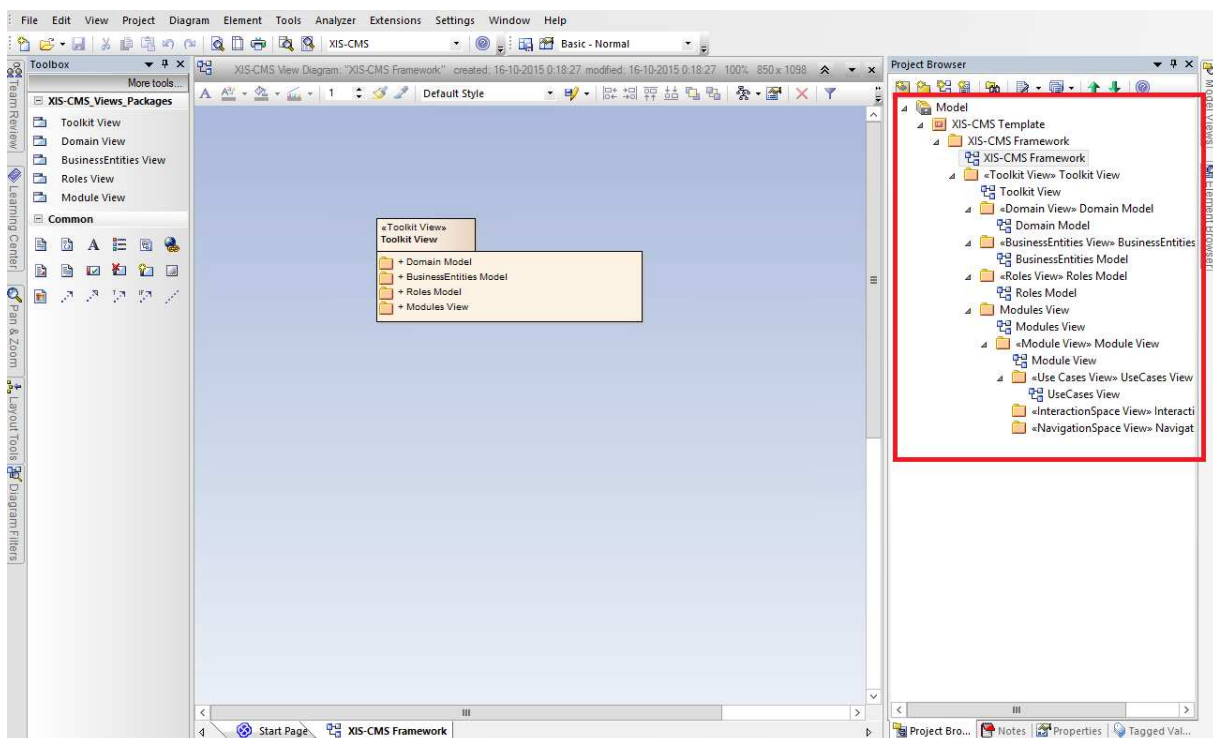


Figure 22: XIS-CMS project structure created by the template.

Finally, custom Quick Linker menus also were defined for the XIS-CMS elements. It is stated by addition of a QuickLink Document Artifact element to the UML profile diagram. This QuickLink Document Artifact element contains the Quick Linker connections for each element and diagram in a CSV (Comma-separated values) format. The main purpose of this creation in XIS-CMS is the errors reduce, as long as it restricts the type of associations and elements created from a given element. The use of QuickLinks also makes the creation and adjustments of the models more expedite.

5.3. Model Validator

The Model Validator is implemented as a plugin leveraging EA's Model Validation API. This solution allowed programmatically defining validation constraints, assigning severity levels (error or warning) to them, and in runtime it allows navigating directly to the error-causing element. Despite that, it is not an OMG standard, like OCL (OMG, 2015), but is performed through a library implemented in the Visual Studio IDE by Microsoft, navigating the model provided by the EA's Automation API. The Model Validator plays a decisive role in the XIS-CMS approach, namely detecting errors produced by the designer at an early stage of development, improving the quality of the models and, consequently, enhancing the quality of the generated models and code.

Below are some of the validations done by this tool:

1. The source and target of a XisEntityAssociation must be a XisEntity;
2. The source of a XisBE-EntityMasterAssociation must be a XisBusinessEntity;
3. The target of a XisBE-EntityMasterAssociation must be a XisEntity;
4. A XisModuleConfiguration can only belong to a XisModule.

5.4. Model-to-Model Transformations

The implementation of XIS-CMS's Model-to-Model transformations stage had as target to offer support to the proposed smart design approach and proving its feasibility, as proof of concept. This approach takes advantage from M2M transformations to automatically generate the NavigationSpace and InteractionSpace views, which are the most time-consuming views of the XIS-CMS language, to be designed. In this case, the M2M creates additional new models based on designed models instead convert it into other ones. For the generation of the NavigationSpace and InteractionSpace views, the M2M transformations use specifically the information defined in the Domain, BusinessEntities, Roles and UseCases views. The selection of the option "Generate Models" in the XIS-CMS Plugin menu triggers the M2M transformation in the XIS-CMS framework. The launching of this process is represented in the Figure 23. The implementation of all the M2M transformation is carried out through C# code using EA's Automation API that allows manipulating the repository that contains the models.

The UseCases View is the great support for the M2M transformation stage of XIS-CMS, because it supplies the abstractions, such stereotypes, tagged values and associations, which configure this process. Specifically, the XisModule and its relationships establish the sort of InteractionSpace view's models that will be generated. For reasons of simplicity, better understanding and due to space, the generated NavigationSpace diagrams were omitted.

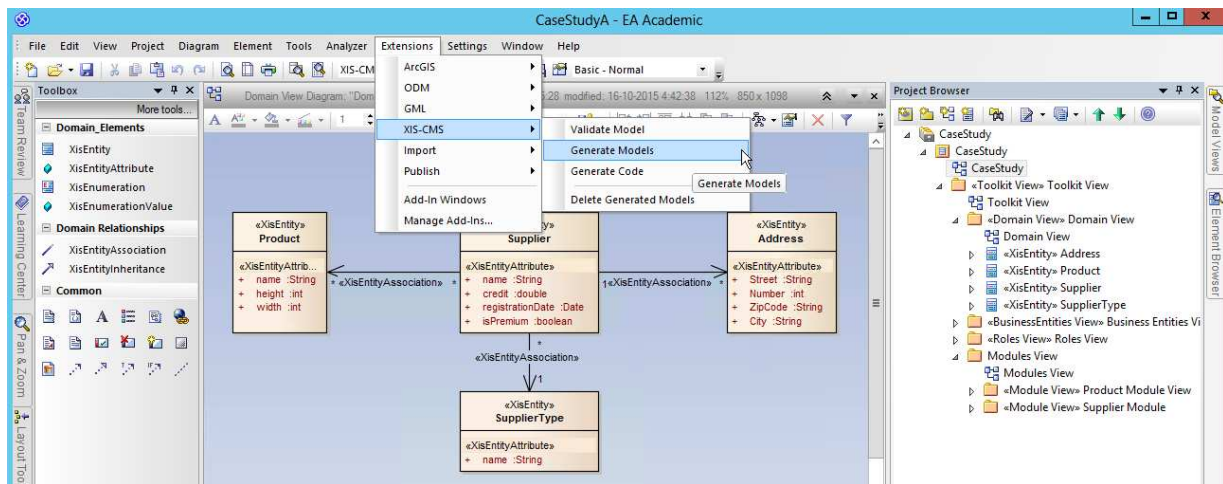


Figure 23: Launch of the Model-to-Model transformations tool

XisModule

A XisModule, as its name suggests, is a demonstration of the module definition. A XisModule must be connected to a business entity (XisBusinessEntity). In turn, the business entity allows the connection of the module to the domain entities, since it adds one or more domain entities. The business entity also allocates tasks to each one of the entities it aggregates. These tasks define domain entities as Master, Detail and Reference. The main domain entity is a Master domain entity, manipulated by the XisModule and around which the User-Interfaces Models will be generated. Each Master domain entity is defined for only one business entity. The domains entities that are associated to the specified Master through aggregations and associations respectively are represented by Detail and Reference domain entities. Thus, it is considered further information of its Master domain entity.

A XisModule holds a set of tagged values representing the CRUD (Create, Read, Update and Delete) and the Search operations for the master, detail and reference entities. These tagged values function as flags and it will result in the appearance of options (menus, menu items, buttons, and interaction spaces) present in the generated models, if it is set to true. Figure 24 represents one example of a XisModule configuration with EA with the tagged values view highlighted in red.

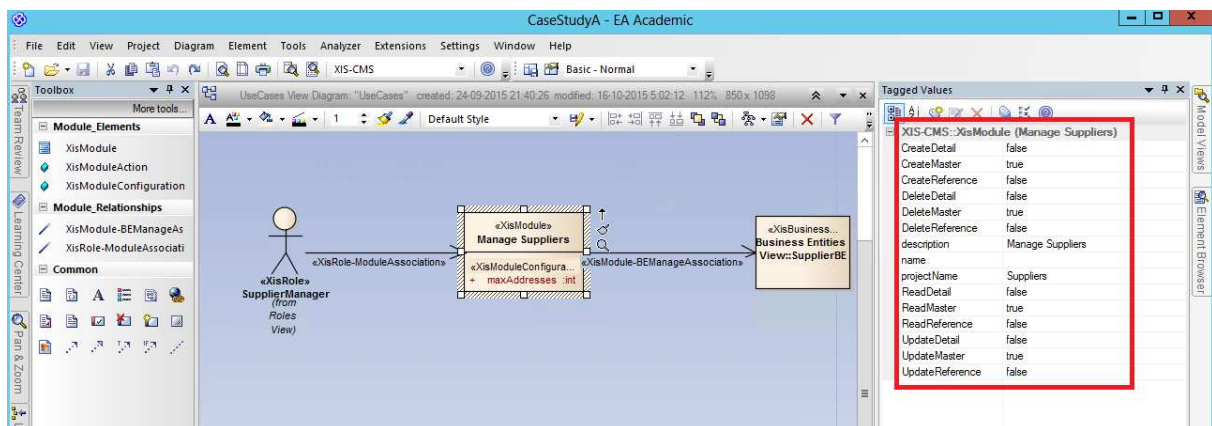


Figure 24: Example of a XisModule and the corresponding tagged values.

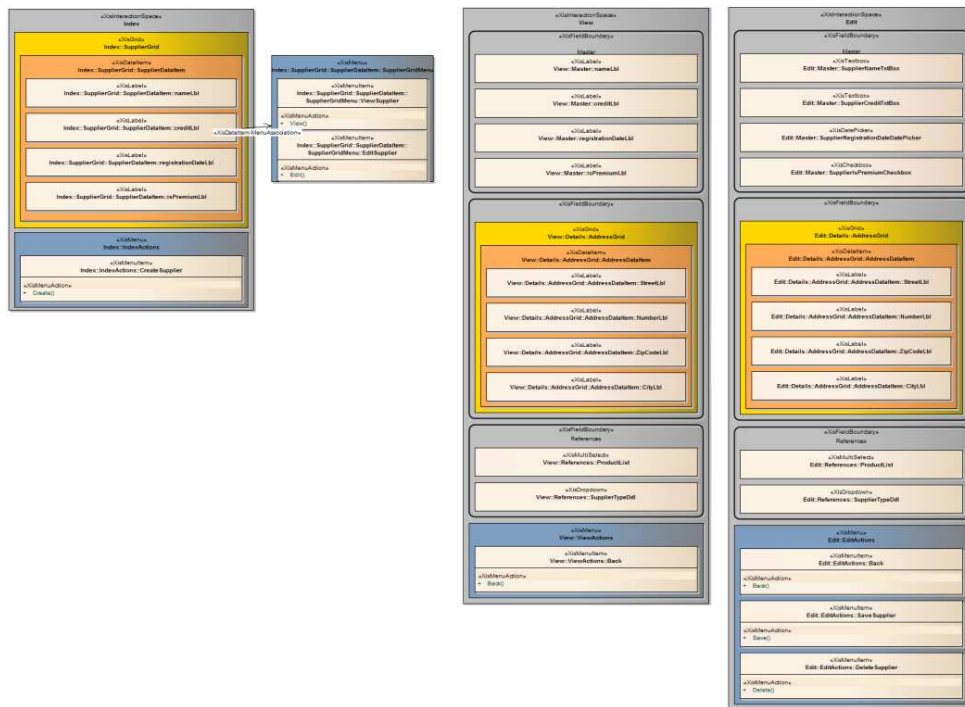


Figure 25: Example of the Interactions space for the Supplier Module

For managing a list of multiple instances of the Master entity (from the associated business entity), will be generated several interaction spaces by a XisModule. The XisInteractionSpaces generated as the entry point in this case is represented in Figure 25. The fourth screen has been omitted from this figure, but it is showed in Figure 26 described later. Furthermore, the associations between interaction spaces and modules also have been omitted.

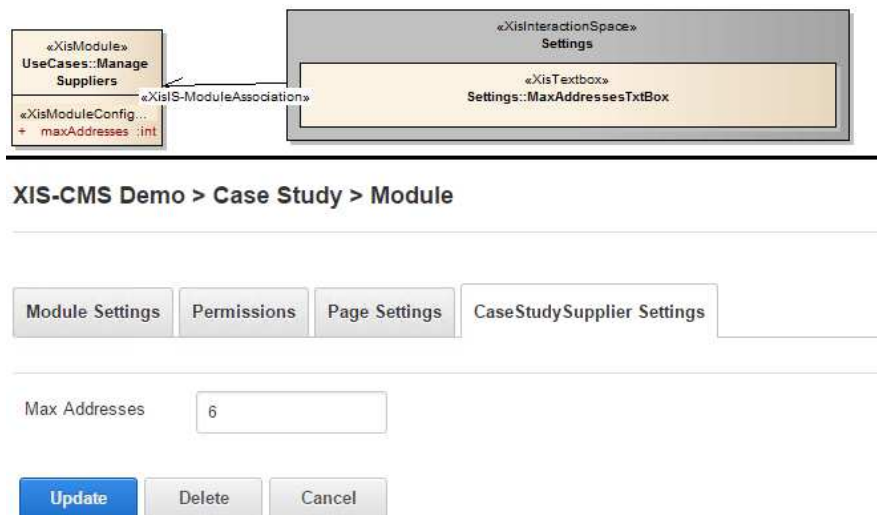


Figure 26: Example of Setting InteractionSpace and the screen

The first screen allows managing all the instances of the Master entity (in this case, a Supplier). The second screen is an editor screen of the Master entity (Supplier) and its Detail entities (Address) and Reference entities (Products). Since a Supplier can have one or more Addresses, the edition of Addresses are implemented by using the visibility boundaries that translate in panels that can be

visible or not depending of the action done. Therefore, the third screen allows view the Master entity (Supplier) and its Detail entities (Address) and Reference entities (Products) with all the fields as read-only as the suggested pattern for CMS modules

The fourth screen is the module configuration interaction space to manage the XisModuleConfigurations. The XisInteractionSpace and the screen that is used on the custom settings are showed in Figure 26.

5.5. Model-to-Text Transformations

The implementation of the Model-to-Text transformations phase of XIS-CMS is fully carried out by using Acceleo, a plugin provided by Eclipse, which promotes the creation of template-based code generators. Acceleo is an open source framework designed by Obeo⁶, a French company dedicated to software solutions to create and transform complex systems as industrial systems (e.g. avionics, space or defense), or IT applications. In 2006, it was released as a plugin for Eclipse, and three years later, in 2009, it was part of an Eclipse's M2T project. Acceleo is responsible for the implementation of the MOF Model to Text Language (MTL) standard and allows the definition of code templates (also known as modules and MTL files) for all EMP-based models, like UML models, for example. As explained before, the mechanism for code generation is supported on templates. It is important to also explain that a template is a special file constituted by regular text (static part of the template) and several annotations (dynamic part of the template). During generation time are these annotations that will be replaced by values of the model elements.

The structure of the Acceleo templates contains two main types of constructs:

- i) Templates, which are used to generate any kind of text. It is sets of Acceleo statements and is delimited by the [template ...][[/template] tags;
- ii) Queries, which do not generate text and is used to extracts information from the model. Queries always returns values and are specified with [query .../] tags that use OCL inside.

Moreover, Acceleo promotes the use of java service wrappers when the definition of the intended template using templates or queries constructs is not possible. In turn, a java service wrapper is a java file that promotes the navigation in the model. The Acceleo non-standard library offers a service invoker which allows the invocation of the java service as a traditional query. Figure 27 shows an extract of an Acceleo template for generating properties in C# classes that support the data repository.

⁶ <http://www.obeo.fr/en/> (Accessed on October 2015)


```

[template private generatePartialClassPropertiesAssociationId(a : Association, idx : Integer, c : Class) post (trim())]
[let prop : String = a.endType->at(idx).name.toUpperFirst()]
  private int _ID[prop/];

  #region Extensibility Method Definitions
  partial void OnID[prop/]Changing(int value);
  partial void OnID[prop/]Changed();
  #endregion

  ["/global::System.Data.Linq.Mapping.ColumnAttribute(Storage="_ID[prop/]", DbType="Int NOT NULL")["]/]
  public int ID[prop/]
  {
    get
    {
      return this._ID[prop/];
    }
    set
    {
      if ((this._ID[prop/] != value))
      {
        if (this._[prop/].HasLoadedOrAssignedValue)
        {
          throw new System.Data.Linq.ForeignKeyReferenceAlreadyHasValueException();
        }
        this.OnID[prop/]Changing(value);
        this.SendPropertyChanging();
        this._ID[prop/] = value;
        this.SendPropertyChanged("ID[prop/]");
        this.OnID[prop/]Changed();
      }
    }
  }
[/let]
[/template]

```

Figure 27: Example of Acceleo code to create a class property

Additionally, Acceleo provides a template editor through which it is possible to edit templates in an easier and assisted way, a profiler that permits the generation quality analysis and a debugger to test, to put breakpoints and to move among instructions of templates during generation time.

The best practice of the Acceleo were followed for the development of the code templates defined for the XIS-CMS framework, particularly in respect of naming conventions, project and module organization and java services use. The organization of the Acceleo project of the XIS-CMS's code generator for DotNetNuke is represented in Figure 28. The Acceleo recommendation that shall be followed for the naming convention is: <project name><kind of input><input metamodel name>gen<output language name>.

The differences between the packages are described below:

- common: This package contains the utility modules, i.e., the modules with the queries commonly used by templates. For example, this package contains a file with several queries that check the XIS-CMS stereotype of a certain UML element;
- database: This package contains all the modules that will generate database scripts.
- files: This package contains all the modules that will generate CMS module files, i.e., modules like the one exemplified in Figure 27; It contains several sub-packages:
 - Common: this package contains modules that will generate the utility classes relevant to the generated solution.
 - Modules: this package contains modules that will generate the project code files that implement the CMS module.
 - Repository: this package contains modules that will generate the data access classes that provide the management to alter or query the data repository.
- main: This package contains the modules with the main templates and their matching launcher class, i.e., the entry points of the generator;

- services: This package contains all the modules that make use of Java service wrappers, as well as the corresponding Java files.

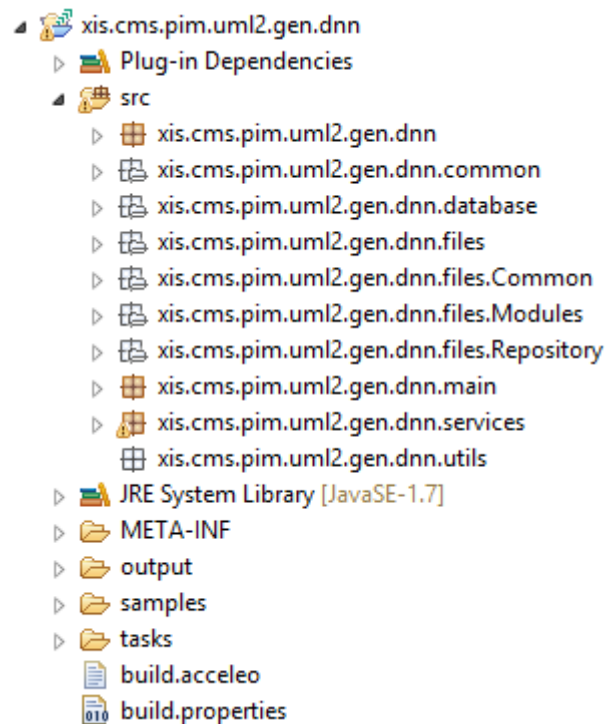


Figure 28: Organization of the Acceleo project for DNN.

For now, the XIS-CMS framework supports the generation of module for the CMS platform DotNetNuke. The XIS-CMS framework generates projects and the corresponding solution ready to be imported by the IDE used in the development of DotNetNuke modules, the Visual Studio from Microsoft. The XIS-CMS framework generates different kinds of files to enable the compilation and deploy of the module, ranging from ASPX, C#, XML, resources or even configuration files specific to the IDE and to the DotNetNuke deploy process, such as license and build scripts. Additionally, it is important to emphasize that whenever the user desires to add support for other platforms, he needs to define the corresponding code templates.

The generated solution was designed to support the development of several modules, while maintain the common domain model. The structure of the solution is showed in the Figure 29, using the case study of the Supplier Management toolkit, and can be distinguished in three components:

1. A project that provides the data access to the data repository. This implementation use the Linq-to-SQL framework to management the operation to the database.
2. A project that provides the utility classes used by other projects, such as function to read screen fields with automatic validation and data type conversion.
3. A project for each CMS module. Each project have all the user controls files and associated classes (designer and code behind), resources, configuration files and support files, such as the license and release notes.

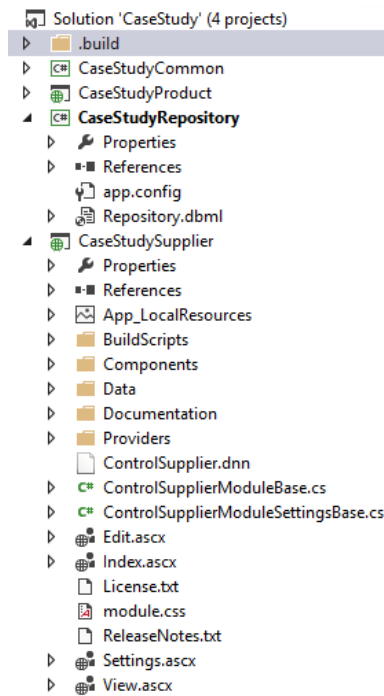


Figure 29: Organization of the Visual Studio solution for DNN

These solutions do not need any additional programming to compile and execute the CRUD operations upon the business entity. In the compile phase, the build process will also archive, using a zip program, the modules projects, creating archive files that are ready to be installed on the CMS platform DotNetNuke.

Complementing the toolkit solution, the XIS-CMS Model-to-Text also produces several scripts to manage the data repository. In the case study of the Supplier Management, is a SQL database, and the sql scripts provided are the following:

- DatabaseScript.sql: contains the commands needed to create the database.
- CreateRoles: contains the commands needed to instantiate the roles modelled in the Roles View, using the stored procedures available in the DotNetNuke installation.
- AssociateRolesToModules: contains the commands needed to instantiate the permissions modelled in the UseCases Views, using the stored procedures available in the DotNetNuke installation.

Due to its complete and updated support of UML profile-based models, as the ones created using XIS-CMS, Acceleo is clearly a wise choice, and was confirmed in the XIS-CMS development as well it was in the XIS-Mobile. The development process is greatly improved by the features mentioned before (editor, profiler, debugger, templates, java services), as well as Acceleo's extensive and good documentation (guides, videos and tutorials). Acceleo is a very active project. This fact can be evidenced by the constant posts in the Eclipse dedicated forum for the M2T project related to Acceleo⁷.

⁷ <https://eclipse.org/forums/index.php/f/24/> (Accessed on October 2015)

6. Evaluation

This chapter presents the evaluation performed on the XIS-CMS approach through the implementation of case studies as proof-of-concept to the MDD approach.

Section 6.1 presents the iterative results of the main case study used during the development of the XIS-CMS approach.

Section 6.2 presents the results of the test sessions by users not involved in this project.

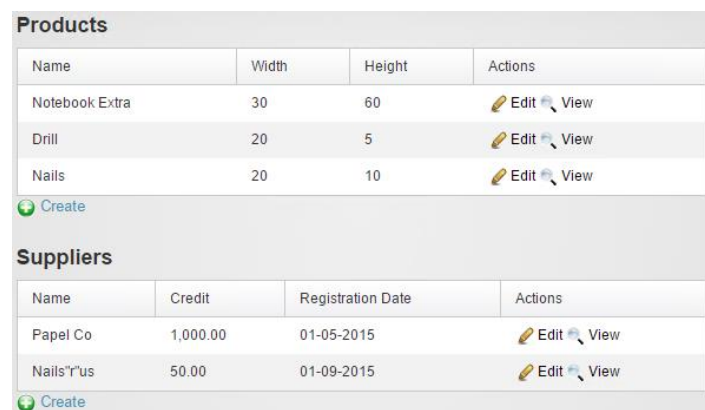
Section 6.3 presents and compares XIS-CMS to some related work found during this research work.

6.1. Case Study A – Self-Evaluation

This section presents the iterative results of the main case study used during the development of the XIS-CMS approach. The case study A, defined in the section 3.4, was the case study used to evaluate the XIS-CMS approach during the Action Research Methodology iterations. The CMS platform chosen to evaluate the XIS-CMS approach was the DotNetNuke framework.

In the first iteration, the Suppliers Management application was manually developed to serve as a control modules and test functionalities upon the business entities. The result of this implementation was a solution with two modules that provide the operations to manage the Product and the Supplier business entity.

In the ensuing iterations, the XIS-CMS language and framework allowed to generate a solution that support the same two module in a fully automated generation. The resulted solution was opened in the Visual Studio 2013 IDE, from Microsoft, and compiled, generating the deploy archive need to the installation of the module in the DotNetNuke platform. Each module was generated with three user controls: “Index” to list the master entity; “View” to show the details of the business entity in a read-only format; and “Edit” to create or update the business entity. The supplier module also had a user control, named “Settings”, to manage the module specific configurations. An example of the index screen of both modules of Case Study A are showed in Figure 30.



The screenshot displays two data tables within a web application interface. The first table, titled 'Products', lists items with columns for Name, Width, Height, and Actions. The second table, titled 'Suppliers', lists suppliers with columns for Name, Credit, Registration Date, and Actions. Both tables include 'Edit' and 'View' links for each entry. A 'Create' button is visible below each table.

Products			
Name	Width	Height	Actions
Notebook Extra	30	60	Edit View
Drill	20	5	Edit View
Nails	20	10	Edit View

[Create](#)

Suppliers			
Name	Credit	Registration Date	Actions
Papel Co	1,000.00	01-05-2015	Edit View
Nails'r'us	50.00	01-09-2015	Edit View

[Create](#)

Figure 30: Example of the Suppliers Toolkit modules index screen.

As a conclusion of this evaluation, the results was successful, given that the generated solution could manage the business entity without need to additional programming. Although, the development of this research work focused on CRUD operations, and gave minimum support to custom operations. The use of specific module configurations and custom operations must be implemented manually by the developer after the code generation.

6.2. Case Study B – User Session Evaluation

In the third and last iteration of this research work, user test sessions were conducted to users that were not involved in the XIS-CMS. These test sessions occurred in variable conditions: on a laboratory session, on one-to-one sessions and through skype. The users did not had previous contact with the XIS-CMS language and framework, although some users had knowledge of similar approaches in other contexts, such as the XIS-Mobile.

The user must had access to a computer with Java, Visual Studio 2013 and the Sparx Enterprise Architect (version 10 or 12) installed. A DotNetNuke portal was used to these test session, and was available online hosted in a Microsoft Azure service. Each user had access to a personal login and page where could instantiate the modules.

The test session was composed by a short presentation of the context problem and the XIS-CMS approach, followed by a demonstration of a simple module: only one entity, without relations. Then, the test users were given a document with instructions to install the XIS-CMS framework and implement the Case Study B. At the end, was asked to the users fill an online questionnaire to collect evaluation data.

The result of the test sessions were collected in 10 responses, wherein one did not answer the first section relative to the XIS-CMS Language. The users had all at least a Bachelor degree and ages between the 22 and 48 years old. Six of the participants had previous experience in the development of CMS applications, namely in module development, and eight have professional experience in the Information Technology field.

The questionnaire was divided in three sections: XIS-CMS Language, XIS-CMS Framework, and XIS-CMS Approach. The answers were given with multiple choice, with the following answers: N/A – Do not know; 1 – Very Low; 2 – Low; 3 – Medium; 4 – High; 5 – Very High.

The XIS-CMS Language section contained the following questions:

Q.L.1: How suitable is the size (number of concepts) of the language?

Q.L.2: How easy to use is the notation used (defined as a UML Profile)?

Q.L.3: How easy to learn is the language without the UI concepts?

Q.L.4: How easy to learn is the language with the UI concepts?

Q.L.5: How suitable is the language for the CMS development domain?

Table 1 shows the results, by applied an average score (with the corresponding standard deviation) to each question regarding the XIS-CMS Language. In general the feedback was positive and reveal a degree of success in the use of the language. Question 1 had two unsatisfactory answers, due to the InteractionSpace possess only the simple XisWidgets. This is due to the iterative process of the research work: the XIS-CMS language is at par with the implementation of the XIS-CMS Framework, namely in the M2T transformations of the defined elements in the language. Questions 2 and 5 got the best feedback indicating a notation easy to use and adequate to the context domain. Question 3 and 4 indicate a moderate easy to learn language. This can be explained by the complexity of the UserCases and the InteractionSpace View. The UserCases View have a large impact on the Model-to-Model transformation, requiring extreme attention, at the risk of the InteractionSpace generation will be inadequate. The difficult with the InteractionSpace view is the high complexity and volume of information in the same diagram.

	XIS-CMS Language				
	Q.L.1	Q.L.2	Q.L.3	Q.L.4	Q.L.5
Average	3,44	4,33	3,67	3,56	4,33
Std. Dev.	0,88	0,71	0,71	1,01	0,71

Table 1: Average score per question of XIS-CMS Language

The XIS-CMS Framework section contained the following questions:

Q.F.1: How do you rate the usability of EA with the XIS-CMS plugin?

Q.F.2: How do you rate the usability of the Model Editor (Stereotypes, Toolboxes, Project template)?

Q.F.3: How do you rate the simplicity of the Model-to-Model transformation (Model generation) process?

Q.F.4: How do you rate the simplicity of the Model-to-Text transformation (Code generation) process?

Table 2 shows the results, by applied an average score (with the corresponding standard deviation) to each question regarding the XIS-CMS Framework. The feedback was positive and indicate a good implementation of the XIS-CMS framework. Question 1 and 2 show a good integration with the Sparx Enterprise Architect tool, but the MDG technology used cab be improved, by adding patterns and quicklinks. Question 3 and 4 show a good usability of the Model-to-Model and Model-to-Text transformations. This can be explained by the M2M and M2T transformation do not require any intervention after the generation.

	XIS-CMS Framework			
	Q.F.1	Q.F.2	Q.F.3	Q.F.4
Average	4,40	4,10	4,70	5,00
Std. Dev.	0,70	0,57	0,48	0,00

Table 2: Average score per question of XIS-CMS Framework

The XIS-CMS Approach section contained the following questions:

Q.A.1: How do you rate the productivity with XIS-CMS comparing to the traditional software development process?

Q.A.2: Would you use such a tool on your own CMS Modules projects?

Table 3 shows the results, by applied an average score (with the corresponding standard deviation) to each question regarding the XIS-CMS Approach. Although the positive score in both the question, the users considerate the productivity high, but less that the applicability in real-world applications. A possible explanation is the need of improve the number of concepts and web elements supported to the users feel comfortable that the XIS-CMS approach can fulfil all the needed requirements.

	XIS-CMS Approach	
	Q.A.1	Q.A.2
Average	4,30	4,10
Std. Dev.	0,67	0,74

Table 3: Average score per question of XIS-CMS Approach

In conclusion, Table 4 summarizes the general results of all the above sections. The results have a general positive score. The XIS-CMS Language had the lowest score, indicated that should be refined and improved in order to support more functionality and present it in a more user-friendly way. Although the sample of participants was small, it can be stated that the questionnaire results can be valid and provide feedback to change the direction, being that was an evaluation to the first proposal of the XIS-CMS approach and following the study performed by (Nielsen & Landauer, 1993), we can conclude, that with ten participants, the results already had passed the point of most benefit/cost ratio.

	XIS-CMS		
	Language	Framework	General
Average	3,87	4,55	4,20

Table 4: Average score per section of XIS-CMS

6.3. Discussion of the Related Work

At the end of this research it is important to compare the XIS-CMS approach with other approaches that exist whose goals are to apply a MDD approach to the CMS domain. These related works were introduced in Section 2.3. Although most approaches have some advantages, it also contain other aspects that are not as complete.

JOOMDD is a MDD approach for Joomla CMS Framework (JooMDD, 2015). Therefore, it is an approach that uses Platform-Specific models. It have an interesting and useful functionality of obtain the models of an existing instance of Joomla, through reverse-engineering the information on the database. It can model the structure and the module information (Priefer, 2014). Comparing with XIS-CMS, JOOMDD supports a bigger scope, by modeling the structure of the website, and the

reverse-engineering functionality is a plus. Although, the fact that is a Platform-Specific proposal do not fulfill our goal of applying MDD to the fullest by aim to a cross-platforms approach.

CM Data Architect and the Web Application Architect also are a Platform-Specific proposal, and thus do not fulfill our goal of applying MDD with a cross-platforms approach. The two tools provide a specialization depending of the user. Although the tools can have more detailed functionalities, create a separation on the models and technical users do not use the same models that non-technical users. This allied with the lack of MTM transformations require that both set of models are created from scratch (Deshpande, McNichols, Richmond, Srinivasan, & Zbarsky, 2005).

WSL adopts MDD with a cross-platform approach, and can generate the application through M2T transformations. The language used is a textual language and do not have a graphical modeling editor. This makes the Web Specific Language more oriented to technical users, without support to high-level design to non-technical users in the form of a tool (Svansson & Lopez-Herrejon, 2014).

CMS-ML and CMS-IL, which served as initial basis for XIS-CMS concepts, is a graphical language with multi-view organization. CMS-ML supports a much wider scope, from the structure of the website, to the template and toolkit deployments. XIS-CMS shares many similar concepts with the Toolkit View of CMS-ML. Namely, both languages have Domain and Roles views; CMS-ML's WebPage and WebComponent concepts are equivalent to XIS-CMS's XisInteractionSpace and XisWidget, respectively; CMS-ML's Tasks view has the same goal of XIS-CMS's UseCases view, differing in the way of representing those concepts. Comparing to CMS-IL, XIS-CMS is in a higher level of abstraction and as so, it does not provide concepts that allow specifying highly technical features, such as algorithm specifications. Another difference resides in the fact that CMS-ML and CMS-IL do not have a technological supporting framework, being validated only in theoretical terms (Saraiva J. , 2013).

XIS-CMS approach supports a MDD approach to cross-platforms, through the Platform-Independent Models. Also share a unique set of models that are used by both technical and non-technical users. And the M2M and M2T transformations in the XIS-CMS framework support all the process in a MDD approach from the design until the generation of application code.

Table 5 compares the discussed attempts to apply a MDD approach to the CMS domain. The Modeling column compares the languages used and the abstraction layer. The Tool Support column indicates the tools used in each stage of the MDD process. The CMS Supported column indicates the CMS platforms the language can support, with the actual implementation within parenthesis, when applicable. The Scope column indicates the focus of the language, which encompasses: the Structure, representing the management of the web application structure in terms of menus, web pages, templates and the module instantiation in each container; the Roles and the Toolkit Modules.

Features Solutions	Modeling		Tool Support			CMS Supported	Scope		
	Language	Abstraction	Design	M2M	M2T		Structure	Roles	Toolkit, Modules
XIS-CMS	DSL, UML	PIM	EA	EA	Acceleo	Any (CMS)	✗	✓	✓
JOOMDD	No	PSM	Eclipse	Eclipse	Xtend	Joomla	✓	✓	✓
CM (IBM)	UML, EMF	PSM	CM Data Architect	Web Application Architect	Eclipse	Content Management	✓	✗	✓
WSL	BNF, EMF	PIM	xText	xText	Xpand, Extend	Any (Plone)	✗	✗	✓
CMS-ML	Yes	PIM	-	-	-	-	✓	✓	✓

Table 5: CMS Model-Driven Development approaches

7. Conclusion

This dissertation presented the XIS-CMS approach, both his language and framework, as a complete Model-Driven Development approach to design and implement Content Management System modules. This research work was developed to reduce the effort of time-consuming task such as produce boilerplate source code and provide a structured path to implement CMS modules directly from the organization data.

XIS-CMS language is a Domain Specific Language, implement as an UML profile, used to design applications in the Content Management System context, mainly at the module implementation scope. The concepts defined by the XIS-CMS language are used to create Platform-independent models. XIS-CMS language contains a multi-view organization, with each view handling different concerns and expose the dependencies between each other. With this separation, each user can concentrate on the view most adequate to his work and function. XIS-CMS proposes eight views: Toolkit, Domain, BusinessEntities, Roles, Modules, UseCases, InteractionSpace and NavigationSpace views. Also, XIS-CMS supports two design approaches: dummy and smart approach.

XIS-CMS framework is the companion framework that provides a set of tools to supply the MDD approach. This set of tools are implemented using the Sparx Systems Enterprise Architect MDG Technologies and Automation API, and the Eclipse Modeling Framework, mainly the initiative Acceleo. XIS-CMS Framework are composed by four tools: Model Editor, Model Validator, Mode-to-Model transformations and Model-to-Text transformations. Each tool correspond to a phase in the XIS-CMS approach using the smart approach: Model Editor to define and design the domain application; Model Validator to validate the correctness of the various models; Mode-to-Model transformations to generate the User-Interfaces views and elements, avoiding manually create the most time-consuming models; and the Model-to-Text transformations to generate the source code.

By using the XIS-CMS approach, a team of technical and non-technical can develop the models, and the developer can increase his productivity by using a MDD approach to generate the base code and refine if needed. And, by using platform-independent models, XIS-CMS can be extended to support various CMS platforms.

XIS-CMS has a work developed in thirteen months by applying the Action Research Methodology by three iterations. The evaluations were determined by applying the XIS-CMS approach to case studies in an iterative way with an increase of the solution complexity. The final result was the generation of a solution ready-to-compile that provides the CRUD operations to a business entity. In the last iteration, test session with users, which were not involved in the development of the XIS-CMS, were performed to better assess the applicability of XIS-CMS. The result questionnaire collected positive scores and gave confidence that XIS-CMS is a practicable, feasible and useful approach to develop CMS modules.

Section 7.1 describes the main contributions of this dissertation and how the XIS-CMS answer the aforementioned research questions.

Section 7.2 describes some ideas that were discussed, but being out of the scope of this research work, are indicated as possible future work.

7.1. Main Contributions

This dissertation presents the XIS-CMS approach, which offers a MDD methodology to implement CMS modules. The graphical edition, through the use of a DSL, of the models can provide common ground to developers and non-technical domain experts, such analysts. Through the use of transformation, both Model-to-Model and Model-to-Text, the implementation of these modules are less time-consuming in boilerplate code, freeing the developers to bigger and more complex requirements.

And, in the following of the research questions presented in section 1.1, XIS-CMS also present the contributions:

R.Q.1: How to apply a MDD approach to the development of CMS modules?

Through the use of the XIS-CMS Framework and Platform-Independent Models. The transformations proved to be very relevant in terms of reduce the effort needed to implement the modules.

R.Q.2: How to specify a CMS module in a platform-independent way?

Through the use of a Domain Specific Language that tries to be the most embracing possible.

R.Q.3: What concepts and view should be used to specify CMS modules applications?

The DSL defines the various views and stereotypes that proved capable of specify and generate a CMS module with success.

R.Q.4: How to generate the source code artefacts to CMS platforms?

The framework provides the Model-to-Text transformations, through the use of Eclipse Modeling Framework, to generate source code to DotNetNuke, being possible to define new transformations to other platforms.

7.2. Future Work

In the development of this research work many avenues of improvement were revealed but not pursued due to the complexity of the time effort required to finish, but mainly because they were not inside the scope this dissertation. Some of these avenues may be developed in future researches and are listed below.

Generalize the common concept of XIS implementations: during this research was identified that many concepts implemented in XIS-CMS were the same of the XIS-Mobile. Therefore, a common language or a language that define core concepts would be of interest.

Extension of the language: XIS-CMS scope is in the module level, but a CMS can be modelled with all its structure of menus, pages and containers. Therefore, a language that could define and model these concepts (probably with basis in the CMS-ML), can offer a more complete solution.

Apply XIS-CMS to other platforms: during this research work, the Model-to-Text transformation only generate solution to the DotNetNuke platform. It would be advantageous to support other CMS platforms.

Reverse-engineering of an existing instance: if an instance of an organizational CMS platform already are implemented, it would be interesting to discover the models and show them in the Model Editor. This would unlock high-level views that could be discussed by non-technical stakeholders.

Support incremental changes: XIS-CMS do not support incremental changes to the module models. If a module already is implemented and there is a change, in the next generation all customization performed by a developer after the code generation is lost. It would be interesting to apply a Hook design pattern⁸ and provide modelling concepts to this functionality.

Generate documentation based on models: MDD was used to generate documentation to specify applications with great success. XIS-CMS will improve if it could be the first step in the design of an application, stating by modelling the application and generate automatic and standard documentation.

Evaluate using real-world applications: though the successfully implementation of two case studies, XIS-CMS would benefit of a real-world application to test the boundaries of the language and evaluate further improvements or new avenues to pursuit.

⁸ <http://stevenblack.com/articles/hooks-and-anchors/> (Accessed on October 2015)

References

- Addey, D., Ellis, J., & Suh, P. (2002). *Content Management Systems (Tools of the Trade)*. Glasshaus.
- Ameller, D. (2009). *Considering Non-Functional Requirements in Model-Driven Engineering (Master Thesis)*. Universitat Politècnica de Catalunya.
- Boiko, B. (2001). *Content Management Bible*. John Wiley & Sons.
- Brambilla, M., Cabot, J., & Wimmer, M. (2012). *Model-Driven Software Engineering in Practice (1st Edition)*. Morgan & Claypool Publishers.
- Carmo, J. (2006). *Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework (Master Thesis)*.
- Czarnecki, K., & Helsen, S. (2003). Classification of Model Transformation Approaches. *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*.
- Czarnecki, K., & Helsen, S. (2006, july). Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal - Model-driven software development*, pp. 621-645.
- Deshpande, P., McNichols, B., Richmond, M., Srinivasan, S., & Zbarsky, V. (2005). Model Driven Development of Content Management Applications. *Proceedings of the 11th International Conference on Management of Data*, (pp. 112-121).
- Deursen, A. (2000, June). Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*, pp. 26-36.
- DNN. (2014). *A Buyer's Guide to Web CMS*.
- DNN. (2015, October). *Giving Our Module a Setting*. Retrieved from DNN: <http://www.dnnsoftware.com/community-blog/cid/155091/giving-our-module-a-setting>
- Drupal. (2015). *Drupal 7 - Getting Started*.
- Ferreira, D., & Silva, A. R. (2012). RSLingo: An information extraction approach toward formal requirements specifications. *Model-Driven Requirements Engineering Workshop (MoDRE)* (pp. 39-48). Chicago: IEEE Computer Society.
- Fondement, F., & Raul, S. (2004). Defining Model Driven Engineering Processes. *Proceedings of the 3rd Workshop in Software Model Engineering (WiSME'04)*. Lisbon.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.
- Giachetti, G., Albert, M., Marin, B., & Pastor, O. (2010). Linking UML and MDD through UML Profiles: a Practical Approach based on the UML Association. *JOURNAL OF UNIVERSAL COMPUTER SCIENCE*, 2353-2373.
- JooMDD. (2015, May). *JooMDD*. Retrieved from JooMDD: <http://icampus.thm.de/joomdd>

- Kieburtz, R., McKinney, L., & Bell, J. (1996). A software engineering experiment in software component generation. *Proceedings of the 18th international conference on Software engineering (ICSE '96)* (pp. 542-552). Washington: IEEE Computer Society.
- Knauss, D. (2008). *A Comparison of Capabilities and Features on Drupal, Joomla! And Wordpress*.
- Kuhn, T., Gotzhein, R., & Webel, C. (2006). Model-Driven Development with SDL – Process, Tools, and Experiences. *Proceeding of the 9th Internacional Conference, MoDELS* (pp. 83-97). Genova, Italy: Springer Berlin Heidelberg.
- Lewin, K. (1946). Action Research and Minority Problems. *Journal of Social*, 34-46.
- Martin, J. (1983). *Managing the Data Base Environment*. Pearson Education.
- Martins, C., & Silva, A. (2007). Modeling User Interfaces with the XIS UML Profile. *Proceedings of the 9th International Conference Enterprise Information Systems (ICEIS 2007)*. Funchal.
- MDA. (2015, October). Retrieved from OMG: <http://www.omg.org/mda/>
- Nielsen, J., & Landauer, T. (1993). A Mathematical Model of the Finding of Usability Problems. *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)* (pp. 206-213). Amsterdam: ACM.
- OMG. (2008). *Object Management Group - MOF Model to Text Transformation Language (MOFM2T), Version 1.0*. OMG.
- OMG. (2011). *Object Management Group - Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, Version 1.1*. OMG.
- OMG. (2011). *Object Management Group - Unified Modeling Language (UML): Superstructure Specification, Version 2.4.1*. OMG.
- OMG. (2014). *Object Management Group - Meta Object Facility (MOF) Core, Version 2.4.2*. OMG.
- OMG. (2014). *Object Management Group - Object Constraint Language (OCL) Specification, Version 2.4*. OMG.
- OMG. (2015, October). *Object Constraint Language (OCL)*. Retrieved from Object Management Group: <http://www.omg.org/spec/OCL>
- Priefer, D. (2014). Model-Driven Development of Content Management Systems based on Joomla. *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, (pp. 911-914).
- Ribeiro, A. (2014). *Development of Mobile Applications using a Model-Driven Software Development Approach*.
- Ribeiro, A., & Silva, A. (2014, October). Evaluation of XIS-Mobile, a Domain Specific Language for Mobile Application Development. *Journal of Software Engineering and Applications*, pp. 906-919.

- Ribeiro, A., & Silva, A. (2014). XIS-Mobile: A DSL for Mobile Applications. *Proceedings of 29th Symposium on Applied Computing (SAC 2014)*, (pp. 1316-1323).
- Saraiva, J. (2013). *Development of CMS-based Web Applications with a Multi-Language Model-Driven Approach (PhD Thesis)*.
- Saraiva, J. d., & Silva, A. R. (2010). Web-Application Modeling with the CMS-ML Language. *Actas do II Simpósio de Informática (INForum 2010)*, (pp. 461-472). Lisbon.
- Saraiva, J., & Silva, A. (2008). The WebComfort Framework: An Extensible Platform for the Development of Web Applications. *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, (pp. 19-26). Parma.
- Saraiva, J., & Silva, A. (2009). Development of CMS-based Web-Applications Using a Model-Driven Approach. *Proceedings of SEDES 2009*.
- Saraiva, J., & Silva, A. (2009). WebC-Docs: A CMS-based Document Management System. *Proceedings of the International Conference on Knowledge Management and Information Sharing (KMIS 2009)*.
- Saraiva, J., & Silva, A. (2010). Web-Application Modeling With the CMS-ML Language”, *Actas do II Simpósio de Informática.*, (pp. 461-472). Lisbon.
- Schmidt, D. (2006, February). Guest Editor's Introduction: Model-Driven Engineering. *Computer, IEEE*, vol.39, no. 2, pp. 25-31.
- Silva, A. (2015, October). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, pp. 139–155.
- Silva, A. R. (2004). *O Programa de Investigação "ProjectIT"*. Lisbon.
- Silva, A., Saraiva, J., Ferreira, D., & Silva, R. (2007). Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools. *IET Software Journal*, 294-314.
- Silva, A., Saraiva, J., Silva, R., & Martins, C. (2007). XIS-UML Profile for eXtreme Modeling Interactive Systems. *Proceedings of the 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007)* (pp. 55-66). Braga: IEEE Computer Society.
- Souer, J., & Kupers, T. (2009). Towards a Pragmatic Model Driven Engineering Approach for the Development of CMS-based Web Applications. *Proceedings of the 5th Model Driven Web Engineering Workshop (MDWE 2009)*, (pp. 31-45).
- Suh, P., Ellis, J., & Thiemecke, D. (2002). *Content Management Systems*. Peer Information.
- Svansson, V., & Lopez-Herrejon, R. (2014). A Web Specific Language for Content Management Systems. *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling*.