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

Francisco Cardoso, Alberto Rodrigues da Silva, João de Sousa Saraiva
INESC-ID/Instituto Superior Técnico,
francisco.cardoso@ist.utl.pt, alberto.silva@acm.org, joao.saraiva@inesc-id.pt

Abstract

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

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

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

1. Introduction

The worldwide expansion of the Internet in the last few years has led to the appearance of many web-oriented CMS (Content Management Systems) [1]–[5] and ECM (Enterprise Content Management) [6]–[10] platforms with the objective of facilitating the management and publication of digital contents.

CMS systems can be used as support platforms for web applications to be used in the dynamic management of websites and their contents [11], [12]. These systems typically present some aspects such as extensibility and modularity, independence between content and presentation, support for several types of contents, support for access management and user control, dynamic management of layout and visual appearance, or support for workflow definition and execution. On the other hand, ECM systems are typically oriented towards using Internet-based technologies and workflows to capture, manage, store, preserve, and deliver content and documents in the context of organizational processes [7]. Nevertheless, these two content-management areas are not disjoint [6]; in fact, it is not unusual to find a CMS system acting as a repository for an enterprise’s documents and contents.

Development of web applications supported by this kind of platforms is typically done using traditional software development processes, in which source-code is the primary artifact, and design models and documentation are considered support artifacts. Such processes are typically time-consuming and error-prone, as they rely heavily on programmers and their execution of typically-repetitive tasks (the so-called “copy-paste development”). Additionally, the source-code and the design models are often out of sync, because changes that are done to the source-code are not propagated to those models.

On the other hand, MDE (Model-Driven Engineering) [13] development processes considers models as the primary artifact, and other artifacts (such as source-code or documentation) are obtained from those models by applying model-to-model or model-to-text transformations, in an automatic fashion. Besides leaving most of the repetitive tasks to the automatic model transformations, these processes also present additional advantages, such as: (1) relieving developers from issues like underlying platform complexity or inability of programming languages to express domain concepts; or (2) targeting multiple deployment platforms without requiring several different code-bases.

Along with MDE, an initiative that has been gaining support recently is the usage of DSLs (Domain-Specific Languages) and its model-oriented perspective, Domain-Specific Modeling (DSM) [14], [15]. Although such languages are not necessarily graphical (textual languages such as SQL, Lex or Yacc have existed for many years), some tools have recently appeared that enable developers to create tools for supporting graphical DSLs (e.g., MetaEdit+ [16], or the Generic Modeling Environment [17]).

In this paper we present a MDE approach to the development of web applications based on CMS systems, mainly based on the usage of a CMS-oriented DSL. This paper is structured in five main sections. Section 1 introduces the concepts of DSL and MDE, as well as the context of CMS systems as support platforms for web applications, and
presents the structure of the paper. Section 2 provides a brief discussion of this CMS DSL approach. Section 3 presents a group of processes, activities and artifacts of the approach on multiple CMS platforms. Section 4 presents related work that we consider relevant for this research. Finally, section 5 presents the conclusions for our research so far, as well as future work.

2. MDE Approach for CMS-based Web Applications

To define a domain specific language, it’s crucial to understand and limit the target domain. The domain analysis was based on the use of some CMS platforms [18]–[20] and through the analysis of scientific literature on the subject [2], [3], [10], [21] and specially the work of João Carmo [11].

The CMS Modeling language, named by CMS-ML, is defined in the form of an UML Profile. The CMS-ML metamodel (which is based on previous work [11], [12]) identifies several elements and relationships that are common to the majority of CMS, figure 1 illustrates the CMS-ML metamodel, showing: (1) all the stereotypes that extends from UML class elements and UML association elements; (2) the enumerations and the stereotypes that extends from UML enumeration literals that are required to set tagged values of some UML class and association elements; and (3) a set of views to be used in application development in order to give several viewpoints of the application.

![Figure 1. CMS-ML Views and Meta-Model](image)

The meta-model identifies the class elements, associations and enumeration literals which when combined provide a web application specification that is simple and enough to produce artifacts in a CMS platform, independently of the platform details. The reader shall see, further in this section, that it’s possible to produce rich platform independent models of CMS-based web applications with the meta-model defined in figure 1 and generate the corresponding web application in different CMS platforms.

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

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

The CMS-ML language pretends to be platform independent, simple, powerful, complete and flexible enough to support various CMS technological platforms and to provide the developers with a complete tool to produce CMS-based web applications in a way that is more oriented to the paradigm of Model Driven Engineering/Development (MDE/MDD). All these characteristics are detailed and proved in the dissertation associated with this paper [22].

One great lack in the CMS-based web applications development approaches, such as the web formulary-based development, is that the stakeholders don’t have the ability to see the web application according to specific viewpoints. To respond to this issue, the CMS-ML approach proposes model views that can be categorized into four main views: WebApplication View, WebPages View, Roles View and Users View.

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

The goal of CMS-ML views is to give simple views that focus in simple concepts in order to give a clear understanding of the web application model parts. To better understand the all these concepts, we shall give a “running” example (the simple example named by SimExe), which is very simple and with the single purpose to guide the model design activity description.

The WebApplication View is the main model view, it contains three specific macro views: the WebPages View, the Roles View and the Users View; and contains two enumer-
The Languages and the VisualStyles enumerations are defined in this view because they represent concepts at the web application level. These concepts are global to all the web application and are used by the WebApplication and the WebPage class elements.

The WebPages View contains two enumerations: the WebPageTypes and the WebComponentTypes; along with two sub views related to web pages: the WebPages Structure View and the WebPages Hierarchy View. Figure 3 shows the WebPages View for the SimExe example.

As the two enumerations of the WebApplication View, the WebPageTypes and the WebComponentTypes enumerations are defined in this view because they are confined to the web page scope. The WebPageTypes contains the literals that represent the type of a web page, used to classify web pages.

On the other hand, the WebComponentTypes enumeration is also related to the web page concept, because the web components are defined in the context of a web page, they are a structural part of a web page. The web components types allows the definition of the component feature that will be mapped to a platform module with an associated functionality.

The WebPages Structure View shows a high level view over the WebPages structural definition. As figure 4 shows, the WebPages Structure View aggregates a set of other views, a structural view for each WebPage. This way, we have a view that focus only in a WebPage structure, with the Containers and its WebComponents.

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

Figure 5 shows a view named by SimExe_Page WebPage View, which represent a view for a specific WebPage. Each [WebPageName] WebPage View will define a WebPage structure, with the Containers defined in the WebPages Structure View, and the WebComponents instantiated in the respective WebPage. The WebComponents used by the WebPage are all defined in the WebPage structural view.
tagged value, despise the displacement of the containers in the diagram.

The visual displacement of the elements in this view, exemplified in figure 5 (b), is crucial to define the containment implicit relationship. All the elements are completely contained in other elements, for example: both containers are completely contained in the web page. For an element to be contained inside another, its visual representation must have all vertices inside the visual representation of the element which must contain it. Example: the Contact Us WebComponent is contained inside the SimExe_C1 Container, which is contained inside the SimExe_Page WebPage.

Figure 5 also shows the web components that are part of the web page. The web components must always have a defined type with one of the literals defined in the WebComponentTypes. In the example of figure 5, each web component type tagged value are: Content, RandomImage and ContactForm; respectively. The programmer will later specify, in the templates, the mapping of these web component types to platform components that already exist in the target platform.

When all the WebPages elements are defined, the analyst can then proceed with the specification of the hierarchical relationships between web pages, through the parent of association. To do so, the analyst must “drag” all the WebPages elements and the WebApplication element into the WebPages Hierarchy View diagram. Then create directed associations between them with the stereotype parent of.

Figure 6 shows the web pages hierarchy for SimExe. The hierarchy is very simple; only one web page exits, the SimExe_Page, that has as parent the web application, this means that the SimExe_Page is at the highest level in the hierarchy.

The Roles View, allows the analyst to create: (1) the RoleTypes enumeration with all the literals required to classify the roles; (2) an UML class element with the Role stereotype for each system role; and (3) two package views: the Roles Hierarchy View and the WebPages Permissions View. Figure 7 shows a Roles View example.

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

It also contains a view for each web page that focuses on the permissions over a specific web page. The view has the name: [WebPageName] WebPage Permissions View, where the [WebPageName] is the name of a web page. Figure 8 illustrates the SimExe_Page WebPage Permissions View for the SimExe example.

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

Figure 9 shows an example of a roles hierarchy. The "author" role will have all the permissions of the role "collaborator", plus some permissions defined for the "author" role.

Figure 6. SimExe: (a) current project structure; (b) WebPages Hierarchy View

Figure 7. SimExe: (a) current project structure; (b) Roles View

Figure 8. SimExe: (a) current project structure; (b) SimExe_Page WebPage Permissions View
Finally, the last view is the **Users View**. This view is very simple and boils down to create UML class elements, apply the *User* stereotype to the elements, define the users tagged values and make a directed association, with the stereotype **assigned**, from a *User* to a *Role*, as figure 10 illustrates.

![SimExe](image)

Figure 10. **SimExe**: (a) current project structure; (b) **Roles Hierarchy View**

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

Our proposed approach for addressing the development of web applications on top of CMS systems consists primarily of two processes (further detailed below): (1) **Platform-Level Development**: which comprises activities that focus on the development of mechanisms that enable the model transformation for new CMS platforms; and (2) **Project-Level Development**: contains activities that aims to produce a CMS-based web application and other artifacts (for example documentation). Figure 11 illustrates the overview of these processes.

![SimExe](image)

Figure 11. **An overview of the proposed MDE approach development processes**.

### 3.1. **Platform-Level Development Process**

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

The **Platform-Level Development** process has a small set of activities with the goal to develop templates for a new CMS platform. This process is very simple in terms of activities and artifacts, it consists of an analysis of a particular CMS platform and later the development and testing of a set of templates that should include all aspects of the platform in analysis. Figure 12 illustrates the **Platform-Level Development** process with all its activities, actors and artifacts.

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

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

After the platform analysis comes the Model-To-Code Templates activity, which is performed by the programmers and aims to develop a set of templates for a specific CMS platform.

A template [23] is a copy of the artifact (e.g. HTML page, source file, etc.) with built-in actions that a template engine evaluates when processing the template, similar to the ASP [24] (Active Server Pages).

By themselves, the templates are sufficient to define these transformations, because they contain all the instructions to transform a model to some type of textual artifact. Rui Silva work [25] presents an extensive description of the templates and the templates engine of ProjectIT-Studio.

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

3.2. Project-Level Development Process

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

Figure 13 shows the overview of the Project-Level Development process with all its activities, actors and the artifacts inputted and outputted for each activity.

In the Model Design activity, the analyst takes the requirements specification and defines a solution in the form an UML2 model with the CMS-ML Profile and the CASE tool. In this activity, the analyst uses the domain specific language for CMS-based web applications modeling, which we call CMS-ML.

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

After the modeling activity, the Model-To-Code Transformation is performed by the programmers and mainly includes tasks as developing web components to support new features and, if required, minor changes to the templates and apply the transformation process. This activity receives the templates and a model representation and produces a set of platform artifacts (ex: code and documentation).

Almost all new projects require new components to include new features in the web application. This way, the programmers need to develop new web components for CMS platforms in the context of distinct projects, and change the templates so that the transformation process can instantiate and configure the new web components.
For example, in a project there might be the necessity for a web component to monitor the energy consumption of a server. This web component is highly specific and probably the template doesn’t have the required information to instantiate it in a web page, nor does the CMS platform have this feature out-of-the-box. So the programmers must develop this component and deploy it in the CMS platform, then change the templates to support the new module. The model-to-code transformations define the required mechanisms to instantiate and configure the concepts, defined in the model that the current templates for the platform doesn’t support.

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

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

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

Finally, the process ends with the deployment of the artifacts on a CMS platform. Deploying CMS artifacts in a CMS platform can take several forms and is directed related with the platform access way chosen by the analysts, in the CMS Platform Analysis activity.

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

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

4. Related Work

Although our approach can address the development of CMS-based web applications, there are also other approaches and proposals that address some of the issues presented in this paper. However, currently there is no significant approach on the model-driven development of CMS-based web applications. Thus, this section discusses some initiatives for the model-driven development of web applications not based on CMS platforms and general purpose applications (for example, windows applications). In this section, we present some of which we consider most relevant in this area.

The Web Modeling Language (WebML) [26] addresses the high-level, platform-independent graphical specification of data-intensive web applications (which can be supported by a CASE tool called WebRatio) and targets web sites that require such advanced features as the one-to-one personalization of content and the delivery of information on multiple devices (e.g., PCs, PDAs, digital televisions, WAP phones) [27]. Thus, it enables designers to express the core features of a site at a high level, without committing to detailed architectural details. WebML also supports an XML syntax, which can be used as an alternative input to software generators for automatically producing the implementation of a web site.

The specification of a site in WebML consists of four perspectives [28]: (1) the Structural Model, which expresses the data content of the site, in terms of the relevant entities and relationships; (2) the Hypertext Model, describing the hypertext that can be published in the site (in turn, each different view of the site’s hypertext defines a so-called Site View); (3) the Presentation Model, which expresses the layout and graphic appearance of pages, independently of the output device and of the rendition language, by means of an abstract XML syntax; and (4) the Personalization Model, in which users and user groups are explicitly modeled in the form of predefined entities called User and Group, whose features can be used for storing individual or group-specific content, respectively. Additionally, a Hypertext Site View description model consist of two sub-models: (1) the Composition Model, which specifies the pages that compose the hypertext, and which content units make up a page; and (2) the Navigation Model, expressing how pages and content units are linked to form the hypertext. Links are either non-contextual, when they connect semantically independent pages, or contextual, when the content of the destination unit of the link depends on the content of the source unit.

On the other hand, the UML-based Web Engineering (UWE) [29] is a software engineering approach for development of applications in the web domain, based on OMG standards (e.g., UML, MDA, OCL, XMI), that focuses on models and model transformation. The UWE notation is defined as a UML profile, and is tailored for an intuitive modeling of Web applications [30]. One of the distinguishing features of UWE is its compliance with standards, which enables its use in existing tools or as plug-ins for tools already in use. UWE comprises [31]: (1) a modeling language for the graphical representation of Web application models; (2) a method (technique) supporting semi-automatic generation; and (3) a process supporting the development life-cycle of Web applications.
UWE focuses on systematization and automatic generation. Its main characteristic is the use of UML for all models, in particular [31]: (1) using “pure” UML whenever possible; (2) for web-specific features, such as nodes and links of the hypertext structure, the UWE profile includes stereotypes, tagged values and constraints defined for the modeling elements.

CMS-ML is more domain-specific than WebML and UWE, because it focuses on CMS-based web-applications.

We also find it interesting to note that similar proposals are now beginning to surface, such as [32]. This work uses the UWE language and presents the creation of a model interpretation web-application (which the author designates “Integration Generator”, conceptually similar to our own notion of “CMS Model Interpreter”) that connects and interacts with the Limestone CMS. This application receives a XML file (which, in turn, results from the processing of a XMI model file with a XSLT style sheet), and is responsible for configuring the CMS, interacting with the backing datastore (such as a database server), and generating the necessary support files (e.g., ASP.NET pages, user controls).

5. Conclusions

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

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

5.1. Future Work

Even though the work described in this paper is self-contained, in the sense that it may be readily used without further developments, it does not constitute the ultimate solution for any of the problems that it addresses.

In particular, we envision that the following main topics, presented in no particular order, will be pursued further in the future: (1) extend the language in order to enable the modeling of web components. To do so, new views must be created, a view for a new web component, where it will be defined. Also new language elements must be defined, such as: text box, button, label, link, and more. This point is by its own a subject of great work and of great interest; (2) provide the tool the capability to monitor the modeling and generate new packages for new elements, eliminating some repetitive tasks that affects the analysts’ productivity; (3) the tool support for the validation of the CMS-ML models based on the language specification. The validation includes: the CMS-ML profile validation, the association member ends are according to the language specification and the CMS-ML elements tagged values correctness; (4) the paper proposes a "pim-to-artifacts" transformation and is effective in this task, it would be desirable to define a mechanism that provides a set of “model-to-model” transformation. The inclusion of "pim-to-psm" transformation should not exclude the "pim-to-artifact" transformation; it’s desirable to provide the developer with the option to choose the model transformation, if "pim-to-psm" then "psm-to-artifact", or only "pim-to-artifact"; and (5) in order to allow the maintenance of web applications, using the CMS-ML approach, it’s required to extend the CMS-ML transformation process in order for it to become a round-trip transformation process, currently it’s only a forward transformation process. This will allow to automatically build the model through a CMS-based web application, i.e., "artifacts-to-PIM" transformation.

References


