Web-Application Modeling With the CMS-ML Language[⋆]

João de Sousa Saraiva, Alberto Rodrigues da Silva

INESC-ID / Instituto Superior Técnico
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal,
`joao.saraiva@inesc-id.pt, alberto.silva@acm.org`

Abstract. The Model-Driven Engineering paradigm has become increasingly popular due to its advocation of using models as first-class citizens in the software development process, while artifacts such as documentation and source-code can be produced from those models by using automated transformations. On the other hand, we are currently witnessing the rise in popularity of a particular kind of web-application, Content Management Systems (CMS). This paper overviews the CMS Modeling Language (CMS-ML), a graphical language for the high-level modeling of CMS-based web-applications. CMS-ML is oriented towards enabling non-technical stakeholders to rapidly model a web-site supported by a CMS system. The language also allows for its extension, in order to support the modeling of more complex web-applications.

Resumo O paradigma da Engenharia Conduzida por Modelos tem-se popularizado devido à sua utilização de modelos como cidadãos de primeira classe no processo de desenvolvimento de software, enquanto artefactos como documentação e código-fonte podem ser produzidos a partir desses modelos através de transformações automatizadas. Por outro lado, estamos actualmente a assistir à ascensão de um determinado tipo de aplicação-web, os Sistemas de Gestão de Conteúdos (CMS). Este artigo apresenta o CMS Modeling Language (CMS-ML), uma linguagem gráfica para a modelação a alto nível de aplicações-web baseadas em CMS. Esta linguagem tem como objectivo permitir que os interessados não-técnicos possam rapidamente modelar um web-site suportado por um sistema CMS. A linguagem também permite a sua extensão, de modo a suportar a modelação de aplicações-web de maior complexidade.

[⋆] This work was supported by FCT (PhD Scholarship SFRH/BD/28604/2006 and INESC-ID multiannual funding) through the PIDDAC Program funds.

1 Introduction

The global expansion of the Internet has led to the appearance of multiple web-oriented Content Management Systems (CMS) [1,2] platforms. CMS systems are web-applications oriented towards the dynamic management of web-sites and their contents, providing concepts such as User, Role, Language, WebComponent, Dynamic WebPage and Visual Theme [3,4]. These systems typically present 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.

Development of web-applications supported by CMS platforms is usually done via traditional development processes, in which source-code is the primary artifact, and design models and documentation are considered only as support artifacts. As is already well-known in the Software Engineering community, such processes are typically time-consuming and error-prone, because they rely heavily on programmers and their execution of repetitive tasks. Also, the source-code and the design models are often out of sync, because changes to source-code are not automatically propagated to the models.

On the other hand, Model-Driven Engineering (MDE) [5] development processes consider models as the primary artifact, and other artifacts (such as source-code or documentation) are produced automatically from those models via automatic model transformations. Besides leaving most of the repetitive tasks to those transformations, these processes 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.

In this paper we present the CMS Modeling Language (CMS-ML), a graphical modeling language oriented towards the high-level modeling of CMS-based web-sites and web-applications. CMS-ML has a number of aspects that distinguish it from other web-engineering-oriented modeling languages and approaches, namely: (1) it is CMS-independent, and so it does not address implementation details; (2) it allows language users to extend it (albeit in a controlled manner) with new concepts; and (3) it is meant to allow regular stakeholders (e.g., users not aware of software development problems) to easily understand and change the model. This language was developed within the context of our research regarding the usage of multiple modeling languages to address the various stakeholder perspectives of a web-application's development [6].

The remainder of this paper is structured as follows. Section 2 provides a brief overview of our approach for the development of CMS-based web-applications, in the context of which CMS-ML was created. Section 3 presents the CMS-ML modeling language, as well as the underlying metamodeling rationale. Section 4 presents a discussion of CMS-ML and our approach, and compares it with some related work. Finally, Section 5 presents the conclusions for our research so far as well and points out some future work.

2 Context

The CMS-ML modeling language was created in the context of our proposed model-driven approach for the development of CMS-based web-applications [6], which is illustrated in Figure 1. Instead of defining a single CMS-oriented modeling language, our approach defines two languages: (1) CMS-IL (CMS Intermediate Language), a common low-level language for CMS platforms; and (2) CMS-ML, which provides a set of elements that are used to quickly model a typical web-application.

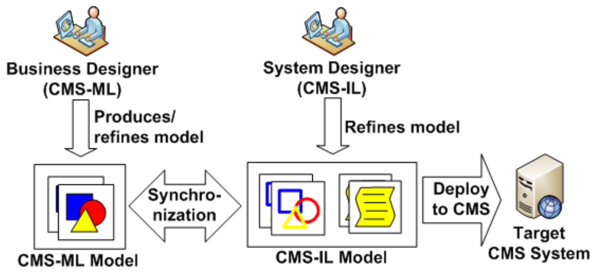


Fig. 1: The proposed MDE-oriented approach.

The Business Designer (a generic term to identify non-technical stakeholders) creates a CMS-ML Model that represents the intended web-application, according to some predetermined business requirements. After applying an automatic transformation from that CMS-ML Model (and obtaining a corresponding CMS-IL Model), the System Designer determines whether that CMS-IL Model is satisfactory, namely by identifying any particular requirements that could not be addressed by CMS-ML alone; if any such requirements exist, the obtained CMS-IL Model must be modified/refined by the System Designer to address them. After this refinement, the CMS-IL Model should be an accurate (and correct) representation of what the intended web-application should be. This CMS-IL Model is then deployed onto a target CMS in one of two ways, depending on the CMS: (1) importation to a CMS Model Interpreter component, or (2) generation of low-level artifacts and subsequent installation. The first alternative is preferable, as it only requires that a CMS Administrator (with administrative privileges) upload the CMS-IL Model into a CMS Model Interpreter, but it will not be feasible in CMS platforms that do not have that component available. In such cases, the second alternative (not illustrated, for simplicity reasons) requires the intervention of a software developer – to perform the compilation of the generated artifacts – and of a CMS Administrator, in order to both deploy the compiled artifacts and make any necessary configuration changes.

This paper will not describe the approach further, as it has been described in [6], and the main objective of this paper is to present the CMS-ML language in greater detail.

3 The CMS-ML Modeling Language

The CMS Modeling Language (CMS-ML) is a graphical modeling language for the high-level specification of CMS-based web-sites and web-applications. Its main objective is to allow regular non-technical stakeholders to look at a web-site's model, *understand* it, and *make changes* to it.

CMS-ML modeling is focused on two different (and complementary) types of model, (1) Web-Site Templates and (2) Toolkits. A *Web-Site Template* (or just *Template*) is a model that reflects the intended web-site's structure and behavior; this Template is modeled using *CMS elements* – such as Role, DynamicWebPage, WebComponent – that are provided by CMS-ML. On the other hand, a *Toolkit* allows the addition of new modeling elements to the set of CMS elements that are available for modeling a Web-Site Template, in a controlled and reusable manner (due to text size constraints, we will not be going into detail regarding the metamodeling rationale behind this language extension capability).

3.1 Roles

Because of the multiple web-site and web-application concerns that CMS-ML addresses, the modeling effort for creating Web-Site Templates will be divided among different kinds of roles, according to the "separation of responsibilities" principle. CMS-ML considers the following modeling roles, depicted in Figure 2: (1) the *Toolkit Architect*, who specifies Toolkits; (2) the *Web-Site Template Creator* (usually just called "Template Creator"), who models a Web-Site Template; (3) the *Web-Designer*, who defines visual themes and graphics for the Template; and (4) the *CMS Administrator*, who instantiates the elements defined in the Template. Of these roles, the most relevant are the Toolkit Architect and the Web-Site Template Creator. The remainder of this section will present an overview of their modeling tasks.

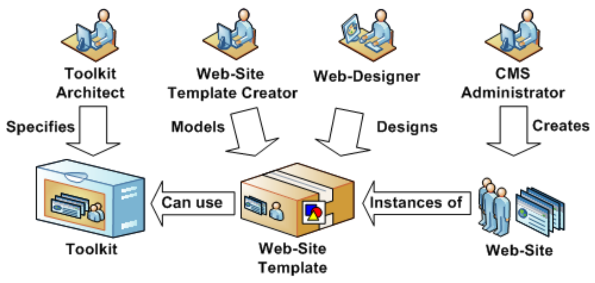


Fig. 2: The modeling roles and artifacts considered by CMS-ML.

3.2 Web-Site Template Modeling

CMS-ML provides a set of generic modeling elements (generically called "CMS elements") that Web-Site Template Creators can use to define their Templates for CMS-based web-sites. A Template is defined according to a set of views (illustrated in Figure 3): (1) the *Structure view*, which specifies the web-site's structural components; (2) the *Navigation view*, specifying the possible navigation flows between the structural components of the web-site; (3) the *Roles view*, which deals with the set of responsibilities that the web-site expects its users to assume; (4) the *Permissions view*, specifying which Roles have access to the web-site's structural components; (5) the *Users view*, which specifies particular CMS users that are considered fundamental to the modeled web-site; (6) the *Languages view*, which deals with internationalization and the languages that the web-site should have available; (7) the *Contents view*, which specifies contents (e.g., pieces of text) that should be available on the web-site; and (8) the *Visual Themes view*, which specifies graphical parameters about how users should view the web-site. The "bootstrapping views" are separated from the other views because they are not necessary for the modeling of a web-site. Instead, the bootstrapping views should only be defined when Template Creators have *a priori* content that should be available in any web-site following the modeled Template.

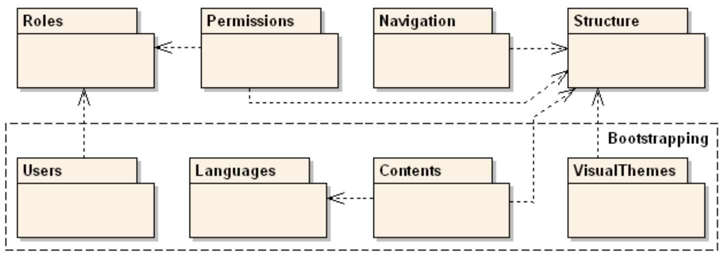


Fig. 3: The views involved in the definition of a Web-Site Template.

The Structure view is the most important, as it conveys the web-site's page structure by using a set of CMS-oriented concepts: (1) *WebSite*, which represents the web-site itself and serves both as a container for Dynamic WebPages and as the element that will import Toolkits (explained further down this text); (2) *Dynamic WebPage*, representing the dynamically-generated pages (in the sense that their contents can be changed through the CMS interface) that users will access; (3) *Container*, which is modeled within a specific area of a Dynamic WebPage and holds a set of WebComponents; and (4) *WebComponent*, representing the "units of functionality" (e.g., Blog, Forum) with which the user will interact. The Structure view is further divided into two smaller views, the *Macro Structure view* and the *Micro Structure view*. The former specifies a "bird's eye" view of the web-site, modeling only the existence of Dynamic WebPages and the relationships between them, while the latter is where each Dynamic WebPage's structure is specified (i.e., what WebComponents are in the Dynamic WebPage, their location, and their order relative to each other). Figure 4 presents the abstract syntax for the Structure view, where the previously-mentioned concepts can be observed.

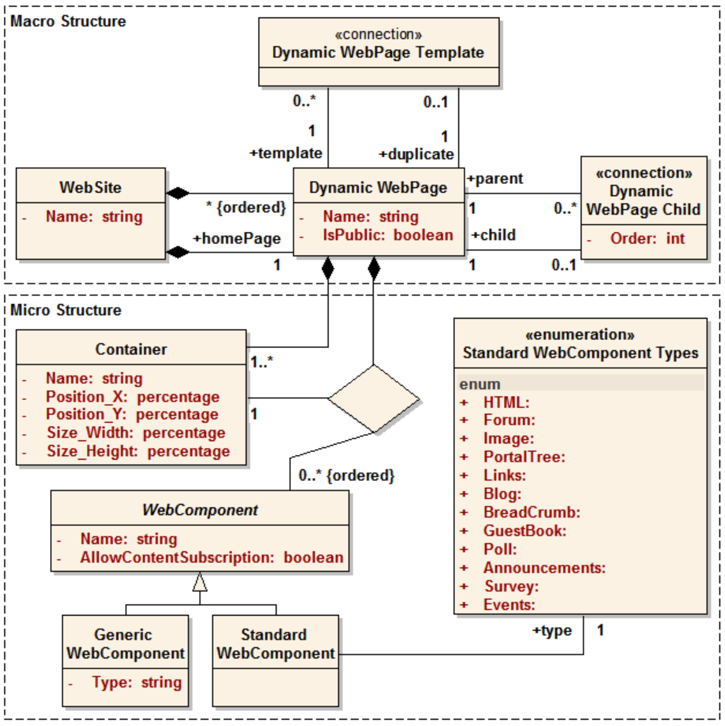
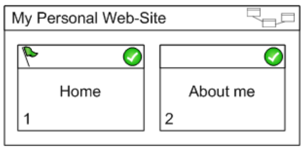


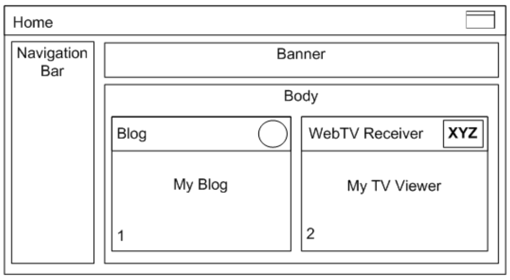
Fig. 4: The abstract syntax for the Web-Site Template's Structure view.

On the other hand, Figure 5 depicts two examples of the Structure view's concrete syntax: Figure 5a illustrates the Macro-Structure view, namely a simple Web-Site containing only two Dynamic WebPages, while Figure 5b shows the definition of a Dynamic WebPage (including Containers and WebComponents) in the Micro-Structure view. The concrete syntax of CMS-ML was defined with the purpose of being easy to understand and to draw manually, without requiring that specialized modeling tools be used in order to create CMS-ML models.

The behavior aspect is only specified in Toolkits (described next) because (1) behavior is usually defined by the WebComponents available in the CMS (e.g., an HTML WebComponent will behave differently than a Forum Web-Component), and (2) even CMS administrators are typically unable to change the web-site's behavior itself, and can only change some parameters regarding specific behavior of the CMS.



(a) Macro-Structure view: Web-Site and Dynamic WebPages.



(b) Micro-Structure view: Dynamic WebPage with Containers and WebComponents.

Fig. 5: The concrete syntax for the Web-Site Template's Structure view.

Due to text size constraints, the abstract and concrete syntaxes of CMS-ML will not be presented in greater detail in this paper, although they will be made available in the very near future at our research group's web-site[1].

3.3 Toolkit Modeling

A Toolkit can be regarded as a "task-oriented extension of CMS elements", as it enables the addition of new CMS-related concepts (namely Roles and WebComponents) oriented towards supporting a particular set of tasks and the corresponding domain model. Like a Web-Site Template, a Toolkit is defined according to a set of views (shown in Figure 6): (1) the *Tasks view*, which deals with the user tasks that the Toolkit should support; (2) the *Roles view*, specifying the Roles that are to perform those tasks; (3) the *Domain view*, which specifies the domain model that is subjacent to the Toolkit's tasks; (4) the *States view*, dealing with the lifecycle of the entities that the tasks are to manipulate; (5) the *WebComponents view*, specifying the WebComponents that will support the tasks; (6) the *Task Interface view*, which establishes mappings between Roles, Tasks and WebComponents, and determines which Roles can do what actions with each of the Toolkit's WebComponents; and (7) the *Side-Effects view*, which establishes side-effects that the modeled Tasks and WebComponents will have.

The Tasks, Roles and WebComponents views are the most important in a Toolkit. The Tasks view allows the Architect to define user tasks as orchestrations of *Steps* which may involve user interaction (similarly to UML's Activity diagrams). The Roles view (not directly related to CMS Roles) models the different kinds of behavior – *Roles* – that the web-application should expect. Finally, the WebComponents view is where the Toolkit's UI (*WebComponents* and *Support Pages*) is specified using *WebElements*, by creating complex UI structures from simpler ones; in turn, WebElements are further divided into *Simple WebElements* (e.g., button, image), *WebContainers* (e.g., DIVs, pop-ups) and *HTML*

[1] `http://isg.inesc-id.pt`

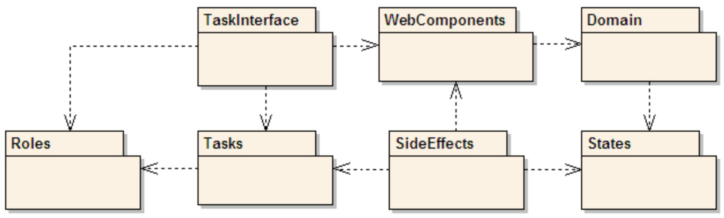


Fig. 6: The views involved in the definition of a Toolkit.

Elements (for cases in which Simple WebElements are not sufficiently adequate). This variety of web interface elements allows the modeling of relatively complex web interfaces using CMS-ML.

3.4 Importing Toolkits

Toolkits can be used in Web-Site Templates or even in other Toolkits, by means of the "Toolkit Import" modeling element, a relationship between a Toolkit (the "imported" element) and either a Web-Site Template or a Toolkit (the "importer"). This relationship is transitive, which means that importing a Toolkit T_1 will automatically import all Toolkits that have been imported by T_1. Also, it is possible to import more than one Toolkit into a Web-Site Template or Toolkit, enabling the composition of Toolkit functionalities in a simple manner.

When importing a Toolkit into a Template, the elements defined in the Toolkit's Roles and WebComponents views become available as new Template modeling elements. When importing a Toolkit into another Toolkit, the elements in the imported Toolkit's Tasks, Domain and States views (but not the Roles or WebComponents views) can be used or specialized by the importer.

It is very important to highlight that Web-Site Templates and Toolkits are located in *different conceptual levels*. While Web-Site Templates are meant to create abstractions of concrete web-sites (i.e., models of those web-sites) by using CMS-oriented elements, Toolkits use generic modeling elements (e.g., *Entity*, *Task*) to create new CMS-oriented modeling elements (namely Roles and WebComponents). Because instances of Toolkit *Role* and *WebComponent* are also automatically considered as specializations of the Web-Site Template's *Role* and *WebComponent* concepts (much like *Generic WebComponent* and *Standard WebComponent* are specializations of *WebComponent*, as the reader can see in Figure 4), Template Creators can then use those Toolkit Roles and WebComponents to create Web-Site Templates exactly in the same manner as when using the pre-defined Template modeling elements.

Figure 7 depicts the metamodel levels that are considered by CMS-ML: the "Toolkit", "Web-Site Template" and "Web-Site Instance" models are created (and changed) by designers, while the "Task Modeling", "Domain Modeling" and "CMS" models are fixed and cannot be changed by designers. In level ML3, Toolkit designers can create instances of generic modeling elements (from Task

Modeling and Domain Modeling, located in ML4) in order to define new elements (Roles and WebComponents) that specialize CMS modeling elements. In level ML2, Template Creators can then use the CMS modeling elements, as well as other modeling elements created in Toolkits, to define a Web-Site Template. In level ML1, the Web-Site Template will be used to create an instance model for a particular CMS installation; this instance model, in turn, will be representing concrete entities that are located in ML0 (the "reality" level, so to speak). Note that the metamodel layers from ML2 to ML0 are actually very similar to what can be found in the OMG's specification of UML [7], because their purpose is nearly the same.

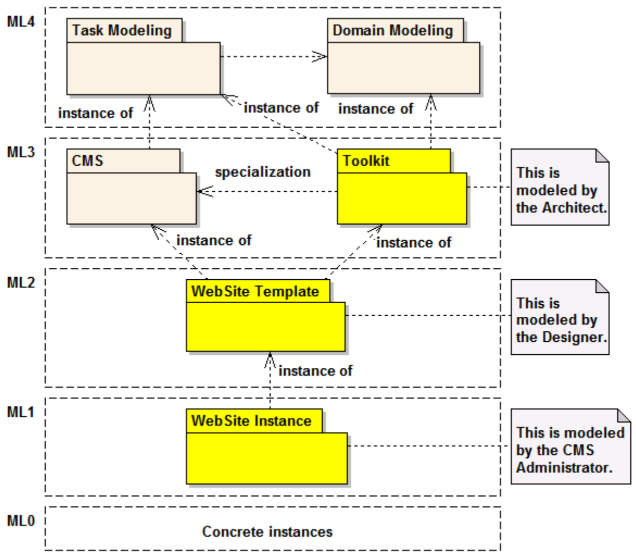


Fig. 7: The metamodel levels considered by CMS-ML.

The rationale for this metamodel level design is to: (1) address language extension in a simple, yet elegant, manner; (2) reduce the accidental complexity [8] that is usually derived from using "type–instance"-like modeling patterns in the same modeling level; and (3) obey the "strict metamodeling" doctrine [9], which states that it should be possible to fully understand any metamodel level as being instantiated from only the metamodel level immediately above it (a consequence of this is that there should be no "instance-of" relationships crossing more than one metamodel-level boundaries).

4 Discussion and Related Work

Besides the CMS-ML presentation done in Section 3, the language does pose some aspects that deserve further discussion. In this section we present and discuss those aspects, while relating them to some external work that we consider relevant for our research, namely the *Web Modeling Language* (WebML) [10,11] and *UML-based Web Engineering* (UWE) [12,13].

One of the first aspects to discuss is the reason why CMS-ML was defined without using any particular meta-metamodel, as opposed to using a mechanism such as a UML Profile [7] (in the same manner that UWE was defined). This is made even more relevant by the fact that, to our knowledge, UML's modeling elements do not present any semantics that contradict the semantics of CMS-ML. However, it would be problematic to represent the Toolkit aspect as a UML Profile, in such a way that elements defined in a Toolkit could then be used to define the Web-Site Template (another UML Profile). This problem is due to the fact that UML (and UML-oriented tools) does not explicitly consider *metamodeling* as an important issue [14], which in turn usually leads to a much greater degree of accidental complexity [8] (i.e., making modeling languages more complex than necessary). It should also be noted that other web-engineering modeling languages do not consider their extension – in the sense of adding new modeling elements – as an important concern (although WebML does define some generic Data-Units, which must be later implemented in source-code, to cover cases in which it is not expressive enough).

At first sight, the Web-Site Template may appear to be adequate for modeling page-centric CMS web-sites (e.g., WebComfort [15], DotNetNuke [16]), but not content-centric CMS web-sites (such as Joomla [17] or Drupal [18]). However, this Template ultimately reflects how *users* will see the web-site, instead of reflecting the concepts that the CMS itself is using. Considering that even web-sites using content-centric CMS systems have a certain structure perceived by their users, we believe that CMS-ML can adequately model web-sites based on content-centric CMS systems.

Another aspect to discuss is how CMS-ML deals with the possible semantic gap between the Toolkit's WebComponents (i.e., UI) and Domain views. While UWE sometimes requires that its Content Model be "tweaked" to particular details of other Models (namely the Presentation Model) [19], WebML defines the Derivation Model to define a layer that establishes mappings between the Data Model and other WebML Models (namely the Hypertext Model). CMS-ML Toolkits also do not require the Domain view to be oriented towards the needs of other views, because the WebComponents view contains a set of modeling elements that allow Architects to specify what parts of the Domain view are to be displayed or used, by using a "binding context" mechanism inspired by our previous work in XIS2 [20].

It is also important to discuss the language's expressiveness and its adequacy to model real-world web-applications. CMS-ML is not very expressive when compared to other languages such as WebML or UWE. However, this level of expressiveness is intentional: because CMS-ML is a part of a larger approach – involving a *set of languages* – the rationale was to reduce the number of modeling elements in this language, in order to make it easier to learn. Nevertheless, unaddressed requirements cannot be just "ignored": that is why CMS-ML defines the concepts of *Unaddressed CMS Requirement* and *Unaddressed Toolkit Requirement*, which are just textual segments (similar to UML comments or constraints) that can be associated with any CMS-ML modeling element. These concepts bring added value to the model (and are not just decorative), because they will be translated to "reminders" in the corresponding CMS-IL models.

The final aspect to highlight is the fact that we believe accidental complexity [8] in CMS-ML has been reduced to a minimum. This is mainly due to the fact that the CMS and Toolkit modeling elements do not include the means to establish "instance-of" relationships between elements; this kind of relationship would become necessary to create models using the "type–instance" modeling pattern, which in turn is usually a source of accidental complexity.

5 Conclusions and Future Work

In this paper we have introduced CMS-ML, a graphical language for the high-level modeling of CMS-based web-applications, aimed at allowing non-technical users to easily understand and change a web-site's model. To achieve its goal, CMS-ML defines a set of CMS-oriented views and can be extended with new concepts. This language is a part of a larger approach for the development of this kind of applications, which explains its lack of expressiveness to deal with concrete implementation details, such as algorithm specification.

Regarding future work for CMS-ML, there are still some open issues, of which we highlight here the ones that we consider most important for the time being.

One of those issues is expressiveness. The CMS-ML language is the result of a tradeoff between language complexity, expressiveness, and how often a given pattern can be found in existing web-applications. However, we acknowledge that this tradeoff will always have a certain amount of subjectivity to it. We consider it necessary to try and minimize this subjectivity factor, in order to make the language more practical, adequate and useful for real-world scenarios. Furthermore, the fact that CMS-ML is independent of any particular CMS makes it unable to use CMS-specific concepts (e.g., Workflow), a problem that we wish to address in the future (likely by using an approach similar to what we did with the Toolkit—Web-Site Template metamodels).

The other issue, closely related to the expressiveness issue, is the validation of CMS-ML. To minimize the subjectivity factor and validate the language in case-studies, we intend to use it for modeling sites and applications with a reasonable degree of complexity. Although we are already validating the language in some academic case-studies, we will also use it in some more complex real-world scenarios, such as WebC-Docs [21], a document management system that we have developed in the context of our research regarding CMS-based web-applications.

References

1. Boiko, B.: Content Management Bible. John Wiley & Sons, Hoboken, New Jersey, U.S.A. (December 2001)

2. The CMS Matrix. Retrieved May 31, 2010 from `http://www.cmsmatrix.org`
3. Carmo, J.L.V.d.: Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework. Master's thesis, Instituto Superior Técnico, Portugal (December 2006)
4. Saraiva, J.d.S., Silva, A.R.d.: The WebComfort Framework: An Extensible Platform for the Development of Web Applications. In IEEE Computer Society, ed.: Proceedings of the 34th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2008). (September 2008) 19–26
5. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer **39**(2) (February 2006) 25–31
6. Saraiva, J.d.S., Silva, A.R.d.: CMS-based Web-Application Development Using Model-Driven Languages. In IEEE Computer Society, ed.: Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA 2009). (September 2009) 21–26
7. OMG: Object Management Group – Unified Modeling Language: Superstructure – Specification Version 2.0 (August 2005) Retrieved May 31, 2010 from `http://www.omg.org/spec/UML/2.0/Superstructure/PDF/`.
8. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software and Systems Modeling **7**(3) (July 2008) 345–359
9. Kühne, T.: Contrasting Classification with Generalisation. In Kirchberg, M., Link, S., eds.: Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009). Volume 96 of CRPIT., Australian Computer Society (January 2009) 71–78
10. WebML.org. Retrieved May 31, 2010 from `http://www.webml.org`
11. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann (2003)
12. UWE – UML-based Web Engineering. Retrieved May 31, 2010 from `http://uwe.pst.ifi.lmu.de`
13. Kroiß, C., Koch, N.: UWE Metamodel and Profile: User Guide and Reference. Technical Report 0802, Ludwig-Maximilians-Universität (February 2008) Retrieved May 31, 2010 from `http://uwe.pst.ifi.lmu.de/download/UWE-Metamodel-Reference.pdf`.
14. Saraiva, J.d.S., Silva, A.R.d.: Evaluation of MDE Tools from a Metamodeling Perspective. Journal of Database Management **19**(4) (October/December 2008) 21–46
15. SIQuant: WebComfort.org. Retrieved May 31, 2010 from `http://www.webcomfort.org`
16. DotNetNuke. Retrieved May 31, 2010 from `http://www.dotnetnuke.com`
17. Joomla CMS. Retrieved May 31, 2010 from `http://www.joomla.org`
18. Drupal CMS. Retrieved May 31, 2010 from `http://drupal.org`
19. UWE – Tutorial. Retrieved December 9, 2009 from `http://uwe.pst.ifi.lmu.de/teachingTutorial.html`
20. Silva, A.R.d., Saraiva, J.d.S., Silva, R., Martins, C.: XIS – UML Profile for eXtreme Modeling Interactive Systems. In: Fourth International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), Los Alamitos, CA, USA, IEEE Computer Society (March 2007) 55–66
21. SIQuant: WebComfort.org – WebC-Docs. Retrieved May 31, 2010 from `http://www.webcomfort.org/WebCDocs`