Model-Driven Development Approach for Content Management Systems based Applications

PAULO NUNES FILIPE, Instituto Superior Técnico, Universidade de Lisboa, paulorcfilipe@tecnico.ulisboa.pt
(supervised by Alberto Rodrigues da Silva)

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 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 corresponding programming and technical skills. We propose a MDD approach, named XIS-CMS, which aims to reduce this problem, and to increase the productivity and portability when developing for multiple CMS platforms, through a more abstract vision of the CMS-based application being developed. XIS-CMS approach includes a domain-specific modeling language, defined as a UML profile, and a companion framework defined on top of Sparx Systems EA and EMF technologies. This dissertation introduces the XIS-CMS language, its corresponding views, constructs and respective relationships. Also, it presents and evaluation and comparison with other works.

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

1. INTRODUCTION

A Content Management System (CMS) is a popular software framework that abstracts the technical features concerning the development and management of web applications defined as the orchestration of multiple web pages and contents. CMS are supported and composed of web modules. A 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 [1][2].

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. CMS provide a ready-to-use editor for managing and creating new pages and menus, and for applying user interface layouts and visual templates. They also offer out-of-the-box modules with basic functionality, such as HTML editor, images, list of links and news or file management [3][4]. However, custom modules must be implemented whenever more complex functionality or more specific business requirements are needed. This kind of modules requires a separate analysis by software analysts and developers.

CMS show important features and properties such as modularity, extensibility and integration with other services through toolkits. CMS also promote the separation between presentation and content through application skins, page templates and module views. CMS offer great adaptability to define and manage the structure of web applications, but do not provide a high-level view of the data that is being used and/or manipulated. Thus, an approach that closes this gap would be an important contribution, by allowing developers to easily discuss strategic changes or business requirements with non-technical stakeholders. This would result in a shorter time-to-market and an easier maintainability of this kind of applications.

With the evolution and emergence of several CMS frameworks (see for example, the popular CMS Matrix\(^1\) website), the adoption of a development approach that can keep the core development portable and easy to apply to new CMS would be also a great advantage. For this reason, such an approach should be as much cross-platform as possible, meaning it should be independent of any CMS framework.

Model-Driven Development (MDD) is an approach that tries to mitigate the above problems in multiple application domains [5][6][7]. MDD is a software development paradigm that combines domain-specific languages (DSLs) with

---

\(^1\) [http://www.cmsmatrix.org/](http://www.cmsmatrix.org/)
transformations engines and generators. DSLs are textual or graphical languages that specify an application domain with its requirements, functionalities and structure [8]. DSLs might be described by 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 artifacts such as source code, through model-to-text (M2T) transformations [9].

This dissertation proposes the XIS-CMS approach to develop CMS-specific modules by defining platform-independent domain models with simple or medium complexity, and then to generate the corresponding source code automatically. This approach includes the XIS-CMS language and the XIS-CMS framework that is the companion supporting tool. The XIS-CMS language is a DSL defined as a UML profile organized in multiple views, in order to describe several aspects of a CMS-based web application (e.g. domain model, types of users and permissions). In turn, the XIS-CMS framework supports the language by offering features like model design, model validation and model transformations. Currently, the XIS-CMS framework supports the generation of CMS modules for the DNN platform.

The outline of this paper is as follows. Section 2 describes the context in which the XIS-CMS approach is based. Section 3 presents the definition of the XIS-CMS language, describing each view and illustrating their use by applying a very simple case study. Section 4 describes the XIS-CMS framework and the tools used to support the applicability of the XIS-CMS language. Section 5 contains the evaluation and discusses the related work. Finally, Section 6 presents the conclusion.

2. BACKGROUND

XIS-CMS derives from previous work, namely from the XIS UML profile [10]. XIS is focused on the design of interactive systems at a Platform-Independent Model (PIM) level following a MDD approach. XIS comprises three major sets of views: Entities, Use-Cases and User-Interfaces. First, the Entity view is composed of the Domain and BusinessEntities views. The Domain view represents the relevant classes to the problem domain, their attributes and the relationships among them. In turn, the BusinessEntities view defines higher-level entities, known as business entities, that aggregate entities of the Domain view or other business entities that can be more easily manipulated in the context of a given use case. Second, the Use-Cases view contains the Actors and the UseCases views. The Actors view defines the entities that interact with the system under study. The UseCases view specifies the operations that the actors can perform over the business entities, when interacting with the system. Third, the User-Interfaces view comprises the NavigationSpace and InteractionSpace views. The NavigationSpace view defines the navigation flow between the several screens of the system (designated by interaction spaces) with which the user interacts. In turn, the InteractionSpace view represents the elements of the graphical interface contained in each interaction space and can also specify the access control of the actors to these elements.

XIS also proposes two modeling approaches: the smart approach and the dummy approach [10]. When using the smart approach, the designer only needs to define the Domain, BusinessEntities, Actors and UseCases views. Then, the User-Interfaces views are automatically generated through M2M transformations (based on the models defined in the Domain and UseCases views). After that, it is possible to refine these generated UI models through direct authoring or design. On the other hand, in the dummy approach, the designer needs to define manually the entire Domain, Actors, NavigationSpace and InteractionSpace views. XIS was originally defined to develop desktop or web interactive applications and to generate code for software frameworks, such as Windows Forms.NET and Microsoft ASP.NET, while XIS-CMS is specifically focused on CMS-based applications, particularly in CMS module development.

More recently an extension of XIS, known as XIS-Mobile [11], has been defined with the focus on developing cross-platform mobile applications. XIS-Mobile is also a DSL, defined as a UML profile, which reuses some of the best concepts proposed on XIS, namely its multi-view organization and design approaches. It also introduces new concepts (e.g. new types of widgets, internet connection, localization and gesture support) in order to be more appropriate to design mobile applications scenarios. Thus, the XIS-Mobile language is organized in six views: Domain, BusinessEntities, UseCases, InteractionSpace, NavigationSpace and Architectural. XIS-Mobile is supported by a framework that allows designing and validating models described in the XIS-Mobile language, generating other models from them (through M2M transformations) and ultimately generating native source code for multiple mobile platforms (Android, iOS and Windows Phone), through M2T transformations [11].

XIS-Mobile, being a successful implementation of the XIS profile, offers a canvas to the framework methodology and supporting technologies to the implementation of the XIS-CMS Language and framework.
3. XIS-CMS LANGUAGE

There are few languages that attempt to design the whole structure and management of a web application (i.e., including menus, pages, modules and page deployment), such as CMS-ML [12] and JOOMDD [13]. Unlike these languages, the XIS-CMS language is focused on the design and development of toolkits of CMS modules, with the respective features for data and modules management. The XIS-CMS language adopts some concepts defined in the XIS language, but introduces new views and an appropriate multi-view organization. As depicted in Fig. 1, 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.

XIS-CMS proposes two modeling approaches: the smart approach and the dummy approach. While in the dummy approach all views must be manually defined, in the smart approach, the NavigationSpace and the InteractionSpace views are automatically generated through M2M transformations from the Domain, BusinessEntities, Roles and UseCases views.

For the sake of better understanding the concepts of XIS-CMS language, we introduce a simple case study, the MySuppliers toolkit, as described below:

The MySuppliers web application allows managing the products and the suppliers of a generic store. A supplier is defined by his personal information (name, registration date, address and city) and can supply several products until a configured maximum value (defined by the module setting). Each product has a name, width and height. The products are managed by a Product Manager, while the suppliers are managed by a Supplier Manager. The Supplier Manager can also view, but not edit, the products.

For this purpose, it is necessary to define a toolkit with two modules: one for the product management and another for the supplier management. Note that this separation is important due to the different permissions each role has on each module.

This toolkit can be deployed on multiple CMS platforms.

3.1 Toolkit View

The Toolkit view includes all the other views and represents the logical aggregation of shared business concepts and modules. It also provides configuration through the use of tagged values. (Tagged values are a set of properties attached to the stereotypes that allow adding additional information to a given UML element.) 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.

3.2 Entities View

The Entities view contains the Domain view and the BusinessEntities view, and describes the domain and business entities and their relationships at different abstraction levels.
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. 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.

Regarding this simple case study, the MySuppliers application only manages two domain entities, the Product and the Supplier, as shown in Fig. 2. The Supplier entity has a n-ary association with the Product entity.

Business Entities View. The BusinessEntities view defines higher-level entities, called business entities. Each business entity is defined as a XisBusinessEntity that aggregates XisEntities. The goal of these business entities is to provide context to the use cases and interaction spaces, which will be useful during the model transformation stages. A XisBusinessEntity defines a master entity (from the Domain View) through a XisMasterAssociation and can also define detail and reference entities through XisDetailAssociations and XisReferenceAssociations, respectively. Both association types 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 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.

As shown by Fig. 3, In the MySuppliers application there are two business entities: ProductBE and SupplierBE. The ProductBE defines the Product as master entity, whereas the SupplierBE defines the Supplier as master entity and the Product as a reference entity.

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

3.4 Modules 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.
3.4.1 UseCases View

The UseCases view describes the operations that the roles (defined in the Roles View) are allowed to perform regarding a specific business entity. A CMS module is represented as a XisModule, defined as a Class, which may contain XisModuleConfigurations, represented as attributes and operations. Permissions to the operations are defined by the XisRole-ModuleAssociation established between a XisRole and a XisModule. Fig. 4 shows these stereotypes and their relationships.

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

The MySuppliers application is composed of two modules: Product and Supplier. Fig. 5 illustrates the UseCases view only for the Product module, where the Product Manager role can perform all available operations (i.e., view, create, edit and delete) and the Supplier Manager role can only view information.

3.4.2 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.

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 [14], 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. Each screen (or “interaction space”) is defined as a XisInteractionSpace, with the association to a XisModule through a XisIS-ModuleAssociation and aggregates one or more XisWidgets. 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, a label, or an input field. In turn, a XisCompositeWidget is a control that groups other XisWidgets, such as a grid or a panel. XisWidgets can have actions associated with them that trigger events or navigation flows.
Because this view is the more labor-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 fulfill the desired requirements.

Considering the Supplier module of the MySuppliers application, the SupplierIndex is the entry screen of the module. It contains a XisGrid element that describes a grid with the columns of the supplier attributes: name, registrationDate, address and city. The XisGrid has a context menu associated with the actions “View” and “Edit”. The “View” action navigates to the SupplierView screen and the “Edit” action navigates to the SupplierEdit screen in the edit mode. The SupplierIndex also contains a XisButton with the “Create” action that navigates to the SupplierEdit screen in the create mode.

**NavigationSpace View.** The NavigationSpace View defines the navigation flow between several screens that are accessible to the end-user when interacting with a module. Each screen is represented by a XisInteractionSpace and the navigation flows between them as defined by XisInteractionSpaceAssociations. The XisInteractionSpace Association has the “actionName” tagged value to define the action that triggers the navigation between the screens. This view provides an overview of the structure of a module and its interactions, which is useful both to technical and non-technical stakeholders.

Regarding the MySuppliers application, the Supplier module’s NavigationSpace View is composed of eight XisInteractionSpaceAssociations and four XisInteractionSpaces (see Fig. 6): (1) SupplierIndex shows all the suppliers; (2) SupplierView shows a read-only screen with the information of the selected supplier; (3) SupplierEdit allows the edition of the selected supplier or the creation of a new supplier; and (4) SupplierSettings represents a custom settings screen to configure the maximum number of products.

Table 1 summarizes the stereotypes defined in each view of the XIS-CMS language and the stereotypes used from other views. For example, the UseCases view uses stereotypes defined in the BusinessEntities and the Roles views in order to define which entities are managed and the corresponding permissions.

**Table 1:** XIS-CMS views and stereotypes.

<table>
<thead>
<tr>
<th>View</th>
<th>Stereotypes</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>XisDomain</td>
<td>XisDomain</td>
</tr>
<tr>
<td>BusinessEntities</td>
<td>XisBusinessEntity</td>
<td>XisBusinessEntity</td>
</tr>
<tr>
<td>XisMasterAssociation</td>
<td>XisDetailAssociation</td>
<td>XisReferenceAssociation</td>
</tr>
<tr>
<td>Roles</td>
<td>XisRole</td>
<td>XisRole</td>
</tr>
<tr>
<td>UseCases</td>
<td>XisModule</td>
<td>XisModule</td>
</tr>
<tr>
<td>XisModuleConfigurations</td>
<td>XisModule</td>
<td>XisModule-ModuleAssociation</td>
</tr>
<tr>
<td>XisActor-ModuleAssociation</td>
<td>XisActor-ModuleAssociation</td>
<td></td>
</tr>
<tr>
<td>InteractionSpace</td>
<td>XisInteractionSpace</td>
<td>XisInteractionSpace</td>
</tr>
<tr>
<td>XisWidgets</td>
<td>XisSimpleWidget</td>
<td>XisCompositeWidget</td>
</tr>
<tr>
<td>XisField</td>
<td>XisField</td>
<td>XisField</td>
</tr>
<tr>
<td>NavigationSpace</td>
<td>XisInteractionSpaceAssociation</td>
<td>XisInteractionSpace</td>
</tr>
</tbody>
</table>
4. XIS-CMS FRAMEWORK

The XIS-CMS language only becomes relevant for the development of CMS-based web applications together with its supporting MDD-based framework. The XIS-CMS framework is implemented using the Model Driven Generation (MDG) Technologies provided by Sparx Systems EA along with the EMF (Eclipse Modeling Framework). Fig. 7 illustrates the suggested development process of CMS-based web applications using the XIS-CMS framework. This process comprises the following steps: (1) the definition of the required XIS-CMS views using the Visual Editor; (2) the validation of those views through the Model Validator; (3) the generation of the User-Interfaces views using the Model Generator; and (4) the generation of the application’s source code for the target CMS platform through the Code Generator. If the models generated after step (3) do not fulfill all the designer’s requirements, the process should return to step (1) for a new iteration where the designer performs his corrections. With the exception of step (1), which is manual, the other three steps are automatic, only requiring the designer to trigger their execution.

![Figure 7: Development process with the XIS-CMS framework.](image)

XIS-CMS framework is composed of four modules: Visual Editor, Model Validator, Model Generator, and Code Generator. First, the Visual Editor is implemented on top of EA through the development of an MDG Technology plugin. This plugin allows the definition of the XIS-CMS language as a UML profile fully compliant with the OMG specification for UML2. Additionally, it also allowed the creation of toolboxes, diagram types and diagram templates customized to the XIS-CMS language. Second, the Model Validator is implemented as a plugin leveraging EA’s Model Validation API. This solution allowed to programmatically define 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 [15]. 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, such as missing associations or tagged values unfilled, improving the quality of the models and, consequently, enhancing the quality of the generated models and code. Third, the Model Generator is also implemented using EA’s environment, namely as a plugin using EA’s Automation Interface. It allows accessing the repository containing the created models, as well as creating new ones. The Model Generator is the responsible for performing the M2M transformations, using the information of the Domain, BusinessEntities, Roles and UseCases views. Fourth, the Code Generator is based on Acceleo, a template-based code generator framework available as an Eclipse plugin. Acceleo implements the MOF MTL (Model to Text Language) standard [16] and allows defining code templates for any kind of model compatible with EMF. The code templates are composed of regular text (static part of the template) and annotations (dynamic part of the template) that are replaced by values of the model during generation time. For now, the XIS-CMS framework supports the generation of CMS-based web applications for DNN. It is important to emphasize that whenever the designer requires the support of other CMS platforms, he only needs to define the corresponding code templates using Acceleo.

Fig. 8 shows the generated toolkit modules of the MySuppliers application and a sample of the product management module. Fig. 9 shows an excerpt of the code generated by XIS-CMS through the M2T transformations.
5. EVALUATION AND RELATED WORK

During the development of the XIS-CMS approach, the iterative Action Research Methodology [19] was used. Section 5.1 describes the evaluations performed during the iterations. Section 5.2 discusses and compares the XIS-CMS approaches with other related works.

5.1 Evaluation

This work had a two-fold evaluation: the first phase was during the development of the XIS-CMS approach through the use and analysis of case studies and comparison with manually programmed modules; the second phase was the realization of user test session to evaluate the XIS-CMS approach from the perspective of user that were not directly involved in this work.

The first evaluation was crucial to the evolutive process of define and implementing the XIS-CMS. Concepts such as the Toolkit View and the need of permission configured in the UseCases view as examples of functionalities that were detected and evolved from this iterations.

In the second evaluation phase, the user test sessions had an introduction presentation to explain the concepts of XIS-CMS and a demonstration of a simple module from design to deploy on the DotNetNuke platform. After that, they had to implement a case study with help of a step-by-step guide, and at the end was requested to fill an online questionnaire. The questionnaire had eleven question divided in three sections. The results were collected from 10 responses from users, with age between 22 and 48 years old with at least a Bachelor degree.

Table 2 shows the average scores of each section. The results were overall positive, although 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, especially the InteractionSpace View. The XIS-CMS Framework had a good acceptance with positive comments due to the generation of source code is ready to compile and do not need additional programming to obtain basic functionalities.
5.2 Related Work

Although the idea of using MDD approaches to develop web applications is not new, it is not widely applied to the domain of CMS and therefore is still not fully explored and challenged. Table 3 summarizes and compares other attempts to apply a MDD approach, but they are more CMS platforms specific which apply a reverse-engineering method to obtain the model, such as JOOMDD for the Joomla CMS platform [13]. JOOMDD has two components: one for developing extensions and another for managing the web application structure. Despite it achieved some success [17], it is a platform-specific approach not satisfying our goal of platform independence.

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 environment. 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 [18].

Similarly to XIS-CMS, CMS-ML [12] is a graphical language and is organized in multiple views. However, CMS-ML has a wider scope than XIS-CMS in the sense that it allows specifying website templates and how toolkits will be deployed in a given website. In turn, XIS-CMS targets the development on a toolkit level assuming that the user has to manually load each toolkit and manage its configurations. Extending XIS-CMS to support the definition of website templates and multi-tenancy is an interesting future work that is intended to further research. XIS-CMS shares some 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. Additionally, comparing to CMS-IL, XIS-CMS is in a higher level of abstraction and does not provide concepts that allow specifying so technical features. Also, XIS-CMS have a technological supporting framework, while CMS-ML and CMS-IL does not.

XIS-CMS does not aim to replace the role of the software developer, but creates the structure of the modules and accelerates the production of common and boilerplate code, which is a considerable portion of the final application’s source code. Then, the developer can customize the source code to better refine and implement the application requirements.

Table 3: CMS Model-Driven development approaches.

<table>
<thead>
<tr>
<th>Features</th>
<th>Modeling Language</th>
<th>Abstraction</th>
<th>Tool Support Design</th>
<th>M2M</th>
<th>M2T</th>
<th>Structure</th>
<th>Tool Support</th>
<th>Toolkit, Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>XIS-CMS</td>
<td>DSL, UML, PIM</td>
<td>PIM</td>
<td>EA</td>
<td>EA</td>
<td>Acceleo</td>
<td>Any (CMS)</td>
<td>×</td>
<td>✓  ✓   ✓</td>
</tr>
<tr>
<td>JOOMDD</td>
<td>No</td>
<td>PSM</td>
<td>Eclipse</td>
<td>Eclipse</td>
<td>Xtend</td>
<td>Joomla</td>
<td>✓</td>
<td>✓  ✓   ✓</td>
</tr>
<tr>
<td>CM (IBM)</td>
<td>UML, EMF</td>
<td>PSM</td>
<td>CM Data Architect</td>
<td>Web Application Architect</td>
<td>Eclipse</td>
<td>Content Management</td>
<td>✓</td>
<td>×     ✓</td>
</tr>
<tr>
<td>WSL</td>
<td>BNF, EMF</td>
<td>PIM</td>
<td>xText</td>
<td>xText</td>
<td>Xpand, Extend</td>
<td>Any (Plone)</td>
<td>×</td>
<td>×    ✓</td>
</tr>
<tr>
<td>CMS-ML</td>
<td>Yes</td>
<td>PIM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓  ✓   ✓</td>
</tr>
</tbody>
</table>

Table 2: Average score per section of XIS-CMS approach.

<table>
<thead>
<tr>
<th>XIS-CMS</th>
<th>Language</th>
<th>Framework</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>3.87</td>
<td>4.55</td>
<td>4.20</td>
</tr>
</tbody>
</table>
6. CONCLUSION

In this paper we presented the XIS-CMS approach, which includes both a language and a framework, as a solution to model and develop cross-CMS platforms modules. The XIS-CMS approach intends to allow both technical and non-technical stakeholders to understand and eventually change these modules and respective applications. XIS-CMS follows a MDD approach, using M2M and M2T transformations, in order to increase the productivity by automatically generating models but also other artifacts, such as source code, simplifying the development and maintenance of applications. XIS-CMS also promotes a platform-independent solution through its PIM XIS-CMS language and so, increase the portability of its models regardless of the CMS platforms used. XIS-CMS language applies the “separation of concerns” principle by defining multiple views to represent each aspect of the CMS modules. Namely, it comprises the Entities (including the sub-views Domain and BusinessEntities), Roles and Modules (including the sub-views Use-Cases and User Interfaces) views. Together with the XIS-CMS language, we also introduced its companion framework 

As future work, we intend to apply XIS-CMS in more complex and real-world domains, which represent opportunities to better employ and evaluate all the concepts that XIS-CMS provides and further explore its requirements. Also, research has to be developed in order to extend XIS-CMS so that it would support the modeling and implementation of a complete CMS web application structure including, for example, the deployment and management of its menus, pages and containers.

REFERENCES

14. E. Gamma, R. Helm, R. Johnson and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.