# CMS-CL : A DSL to simplify the configuration of Content Management Systems

Matthieu Allon, Jordi Cabot

AtlanMod team (Inria, Mines Nantes, LINA), Nantes, France
{matthieu.allon,jordi.cabot}@inria.fr

**Abstract.** Most company websites are implemented on top of a Content Management System (CMS). Even though CMSs are simpler to use than pure static HTML sites, their configuration still require technical knowledge (database configuration, installation of plug-ins,...) that no all end-users have.
In this sense, we propose a Domain Specific Language (DSL) that enables to non-expert users to easily indicate their configuration preferences (in terms of functionalities, appearance, default content, ...). As notation for our DSL we adopt the mind-mapping syntax which is easy to learn and familiar for many users. Our approach has been implemented for WordPress, and an experimentation has been carried out with end-users to measure its usefulness.

**Keywords:** Mind mapping, DSL, CMS, Configuration, End users

## 1 Introduction

Nowadays, many tools are available on the market to simplify and speed-up the development of websites. Such tools can provide support for static (e.g. the DreamWeaver, WebExpert, ... editors) or dynamic (e.g. the Drupal, Joomla, ... CMS) websites. The most widely used tools are CMS (Content Management System - about 60% of the current websites use a CMS [1]): it brings clear adavantages [2], with respect to other methods and tools.

Unfortunately, CMS have drawbacks for end-users: the technical knowledge (e.g. plugins, theme, ...) to use it, the lack of default functionalities (e.g. for Search Engine Optimization, ECommerce, Social Networks ...), the concepts scattering and the lack of participation of end-users in website creation.

Hence, we have created a DSL [3] (named 'CMS-CL': CMS Configuration Language) which has all concepts present in WordPress, with some default functionalites (e.g. Search Engine Optimization, anti-spam, multi-language, ...). With this DSL, users focus only on the conceptual side (e.g. relations between a post and a page, ...).

We have decided to use a specific representation with our DSL: the mind-mapping. It is a way to represent concepts: each one is symbolized as a node (the root/central node being the main concept), and the various relations between these concepts are represented through edges. The mind-mapping allows to easily manipulate the different DSL concepts, to have all of it in one view and to enable end-users to be less partisan of a wait-and-see by participating more in the website creation, and therefore exchanging more with web designers and other members of a website project.

We have implemented a prototype using our DSL with a mind-mapping based notation. Then, we made an experiment of this tool with end users, to validate its interest.

This paper presents the design and how to use the DSL through the following sections : the approach overwiew (Section 2), a presentation of our DSL and its syntax (Section 3), a presentation of the implementation (Section 4), the CMS-CL prototype validation (Section 5), CMS-CL and the web designers (Section 6), the related work (Section 7) and a conclusion about this approach.

## 2 Approach overview

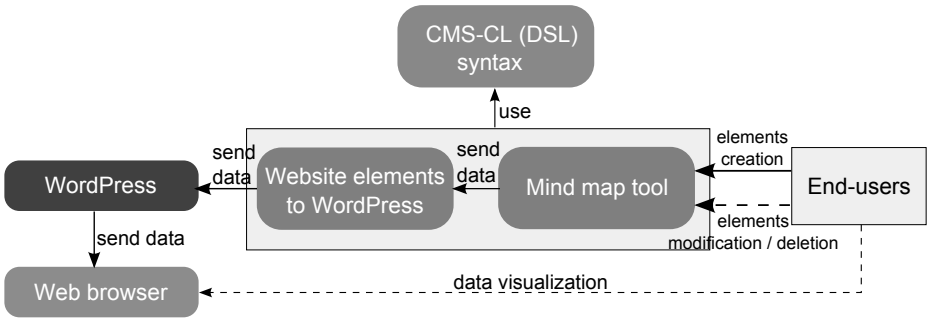


**Fig. 1.** CMS-CL and mind mapping use

As it is presented in Fig.1, the use of our defined DSL ('CMS-CL') with the mind mapping is the following :

**Elements creation.** Using a GUI [1], end users are able to create some website elements.

**Mind-mapping metamodel reduction.** The mind mapping metamodel is reducted to the elements of the CMS-CL metamodel (paragraph 3.1). Then, data (elements) are injected into the WordPress platform.

**Data visualization.** After the data injection, it is possible to see the changes on the web site, via a web browser.

**Elements modification and/or deletion.**

*Deletion:* Users just have to put a delete icon (✗) as described in the section 3, paragraph 3.2.

*Modification:* Users must have to delete first the corresponding element, and recreate a new one with the modification. The only exception concerns the 'Website' root node, because it is mandatory : its attribute can be directly modified.

## 3 CMS-CL (CMS Configuration Language)

'CMS-CL' is a DSL. A DSL [3] is a computer language for a specific domain (e.g. SQL, HTML, OCL, ...) : its syntax are designed to work in the targeted domain as good

[1] Graphical User Interface

as possible. This syntax is composed of three parts : the *abstract syntax* is the unvariable part (i.e. the structure of the language) without any particular representation. The *semantics part* is about the meaning of the elements, and the *concrete syntax* concerns the variable part, which describes the representation of the language. In the following sections, the CMS-CL syntaxs and semantics are detailed.

### 3.1 CMS-CL Abstract Syntax and Semantics

The DSL created for the website management contains several features allowing configuration modifications and creations by users. These elements have been represented in a metamodel [4]: for the sake of conciseness, we do not show the entire metamodel[2], but a simplified one in Fig.2.

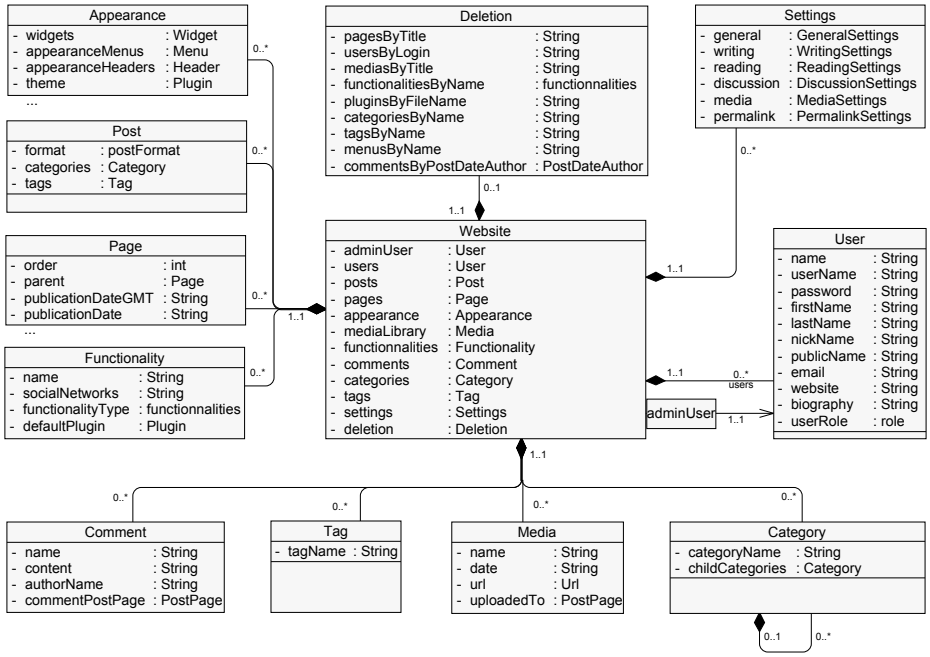


**Fig. 2.** CMS-CL simplified metamodel (abstract syntax)

**User.** With this concept, it is possible to create users with their mandatory respective roles (granting rights to the various elements of the site) : *'Administrator', 'Editor', 'Author', 'Contributor' and 'Suscriber'*. It is important to notice that users modifying the website configuration must have an administrator role, which is represented by the element *'AdminUser'* (by default the users are connected with the administrator account automatically created during the WordPress server installation).

**Appearance.**End users will be able to change the appearance of the website, using the elements of 'Appearance'. The *'apperanceHeader'* and *'theme'* elements will respec-

[2] CMS-CL complete metamodel: `https://raw.github.com/mallon/WordPress-MindMapping/master/Documentation/DetailedFigures/WdpDslCompleteMM.pdf`

tively change the header of the site's main page and select a new theme. Its modification will influence how other features appear (e.g. the placement of the page on the website). For the 'Theme' feature, there are six default themes, choosen accordingly to the list of the most popular ones in 2013 [5] : *Responsive, Magazine, Business, Galleries, Artistic and SEO*. The item *'Menus'* allows changing the default content of the navigation elements. It is possible to change the default access to the pages and posts, by creating a new item *'Menu'* with external links (*'Links'*), pages/posts categories (*'Categories'*) or pages (*'Accessed pages'*). Furthermore, end users can add functionalities to the site (and are displayed regardless of the page/post) with the twelve default *'Widgets'*.

**Deletion.** This entity contains all the names of the various items that the user wish to delete.

**Comment.** It represents the comments write by users on a page/post.

**Category.** Each categories can be composed in sub-categories. For instance, it can exists a category 'Music' containing two sub-categories 'Group' and 'Concert'.

**Media.** This feature corresponds to images and videos which can be inserted with the content.

**Tag.** It allows to find various posts concerning a same subject. Navigation within a website can be easy through the use of tags in order to find a particular information : for this reason, a tag can be added to a post.

**Functionality.** Users usually wish to customize their web site according to their needs. Hence, a set of default functionalities is provided: *eCommerce, spam, indexing, multilanguage, seo, pictures, and social Networks*. The users may select a functionality among them , or install another one ('plugin' attribute).

**Page and Post.** Users (which have the correct rights) can add posts. Pages can be also added, and for this two elements, the author correspond to the administrator account of the user (see 'User' details 3.1).

**Settings.** It contains the various types of configuration : *General, Writing, Reading, Media* and *Permalinks*. In the *GeneralSettings* class, *login* and *administrator* password attributes are required to enable the connection (to the platform site creation) which is realized with the platform *address* attribute. But the settings require more technical knowledges, therefore the end-users will not have the possibility to modify it : only the web-designers will be able to do this (section 6).

### 3.2 CMS-CL Concrete Syntax

In the next two paragraphs, we present the tool used to represent a mind-map and the CMS-CL concrete syntax, which is the visual representation. A mapping between the metamodel concepts (the abstract syntax) and this visual representation (the mind-mapping) was necessary.

**Representation Tool used.** There are a lot of different softwares for mind mapping, and FreeMind was during a long time of those who have different interesting features : popularity, soundness, interactiveness, open source, extensibility, export facilities and scripting. A fork of FreeMind was created: Freeplane. It provides and improves some of the points mentioned above: a regularly updating and an improved extension system [6].

**CMS-CL and mind-mapping.** A mind-map is a diagram used to present concepts (or other items like tasks). There is one central governing concept, which is in the center of this diagram, and each other concepts are related to it. Each idea is symbolized as a child node of the central node (i.e. the central concept), and the various relations between these concepts are represented through edges. The software presented above adds the possiblity to create attributes for a node : an attribute is a node property (e.g. an address for a website).

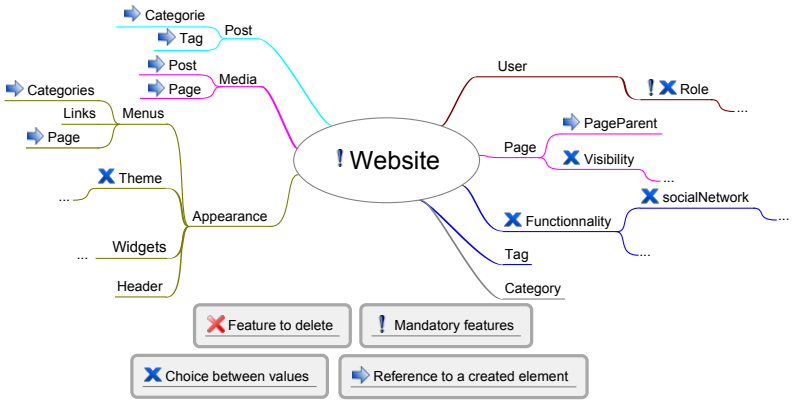


**Fig. 3.** CMS-CL on top of the FreePlane metamodel (simplified representation)

The various concepts, their parameters and relationships are respectively represented by nodes, attributes, and edges. CMS-CL has a visual representation via the mind mapping and it is on top of the Freeplane metamodel. To adapt our DSL to this metamodel[3] , it was necessary to make a 'CMS-CL-to-FreePlane' mapping[4] [5].

**Website.** It is the root node, which represents the web site to create, with its title, tagline, login and administrator password attributes.

**Appearance.**Directly connected to the root node, it contains four child nodes : *Widget*, *Menu*, *Header* and *Theme*. Each of this fourth child nodes have also child nodes.

**User.** Child node of the 'Website' root node. It has a child node *Role*.

**Page.** Child node of the 'Website' root node. Its title and content are defined by its first two attributes: 'title' and 'content'. His child node *'PageParent'* represents the page to be displayed with it (in the site menu bar by default). The definition of the attribute 'order' within the parent node *'Page'* allows you to change the order. Because a page can be public or private, a second child node was added to the node *'Page'* : *'Visibility'*.

**Post.** Child of the root node, it is an article posted by a user. It contains two child nodes referencing existing other nodes : *Category* and *Tag*.

[3] FreePlane metamodel: https://raw.github.com/mallon/WordPress-MindMapping/master/Documentation/DetailedFigures/FreeplaneMM.pdf

[4] 'CMS-CL-to-FreePlane' mapping: https://raw.github.com/mallon/Word-PressMindMapping/master/Documentation/DetailedFigures/CMS-CL2FreePlane.pdf

[5] Schematique representation: https://raw.github.com/mallon/Word-PressMindMapping/master/Documentation/DetailedFigures/CMS-CLFeaturesComplete.pdf

**Tag and Category.** Child of the root node, with their attributes 'name' and 'description'.

**Functionality.** Child of the root node, it has seven child nodes. It is possible to choose only one of them. For the functionalities concerning social networks, two children nodes of the *'SocialNetwork '* node can be added (*'facebook'* or *'twitter'*).

**Media.** Child of the root node, it has *Page* and *Post* nodes as child. Each of them has an *arrow* icon, to be related to others existing *Post* or *Page* nodes.

**Settings.** Because the targeted user for the mind-mapping is users with a low technical knowledge, the 'Settings' are not represented in this visual concrete syntax, but in the textual one (6), concerning web-designers.

**Constraints.** There are some constraints on the FreePlane mind map, because a CMS-CL expression is conform to a FreePlane model, but not the opposite. This conformity is determined by the CMS-CL abstract syntax, and we can see the various constraints on the figure 3, with the four free nodes, used as keys. To summarize, the constraints are the following :

*Deletion:* To delete a website element, it is necessary to add the delete icon (✘) on the targeted elements. More precisely, the elements which support this delete icon are those with attributes (excepted the 'Website' root node, which is mandatory), or with a choice concerning their child nodes (so, the nodes with a cross icon).

*Mandatory features:* Some elements are mandatory, as the child node *Role* of *User*, or the three attributes of the *Website* root node : *address, adminPassword and adminLogin*. The icon used is the *Exclamation mark* one (❗).

*Choice:* Several nodes have several child nodes, but it is possible to choose only one of them. It is the case for the functionalties or the themes. The icon used is the *Cross* one (✘)

*Reference:* Some nodes have child nodes referencing other existing nodes. For instance, the *Post* node can referenced an existing *Category* node. It would have been possible to use *arrow links*, but it is more readable with an *arrow* icon (➧), because it avoids to have a lot of links.

## 4 Implementation

There are two categories of components : the client and the server, for which the followings parts presents various functionnalities.

### 4.1 Client part

**Mind mapping (1).** To manipulate concepts with the mind mapping, the end-users use the Freeplane GUI and one of its output format which is the XML. With Freeplane, the users create a mind map (see section 3, paragraph 3.2) of the different elements they want to manage for their web site. Then, via the export menu, they get a file representing this mind map, in an XML format[6] [7]. There are other formats proposed

[6] Extensbile Markup Language

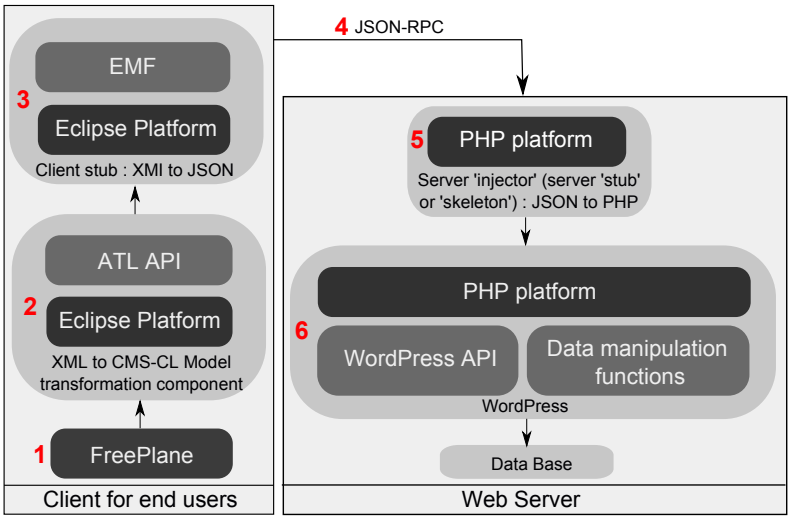


**Fig. 4.** Client-server complete process

(svg, mwiki, html, jpg, odt), but XML has some advantages that the other ones do not have : information structure is easily discerned by both humans and computers and it is application-independent.

**Transformation to CMS-CL model (2).** The XML representation must be now transformed in a CMS-CL model. For this, ATL[7] is used : it is a model transformation language an Eclipse plugin. The output CMS-CL model is used by a third client component whose the description is done bellow.

**Client 'stub' (3).** This component is called a 'stub' because it allows to have the same data format between the client and the server for the data interchange : the client parse (it is called 'marshalling') the data in JSON. So, the CMS-CL model is transformed to this common data interchange format by using the EMF[8] [8] on Eclipse. The choosen format was JSON (see the listing1.1 example) because it is specialized for data interchange, which is the case between the client and the server. It would be possible to have a direct transformation to this format (XML-to-JSON), but there are four reasons we choose to do this in two steps ('XML-to-CMS-CL' and 'CMS-CL-to-JSON') : *Maintenability* (only the 'CMS-CL-to-JSON' transformation would be changed on a data interchange format modification), *Productivity* (if it is easier to maintain, it is also a gain of time), *Readability* (with a one step process, we would have more complex metamodels, which is less readable), and *Reusability* (each transformations could be usable by other components).

```
{"website" : { "adminUser": { "idUser":"userOne", ... } }, ... }
```

**Listing 1.1.** Web site elements in JSON

[7] AtlanMod Transformation Language
[8] Eclipse Modeling Framework

### 4.2 Communication between Server and Client (4)

The communication between the client and the server parts use the JSON-RPC [9] protocol : it uses the RPC protocol to call server methods, and the requests are in the JSON format.

### 4.3 Server part

**Injector (server 'stub' or 'skeleton') (5).** This injector -also called the server 'stub' or 'skeleton'- consists to do the inverse of the 'stub' on the client side, i.e. to unparse the parameters (also called 'unmarshalling') and call its local WordPress functions with the parameters.

**API and Data manipulations functions (6).** The functions called by the injector are divided in two types : those available in the WordPress API and those which are not remotely callable. But all of them send data to the data base, allowing a website update.

## 5 Validation

The aim of this step was to validate the utility of the tool with end users. To do this, we created a simple use case[9], and retrieve users impressions via a questionnaire[10].

### 5.1 Use case

**Mind-mapping process.** During this step, the user must create mind-map, inject the elements in the WordPress CMS, and then observe the changes of the website. The provided mind-map for this test consisted of four components, as shown in Fig.5: Post, Page, User and Theme. But the user was always able to add a element choosed among these four types.


**Fig. 5.** Mind-map sample

**WordPress use.** Users had to the same as in the previous step, but this time with WordPress, to be able to compare the two tools.

[9] Experimentation package: `https://drive.google.com/file/d/0B3kzMiMYv9r2ZXZ2c2VsTnA4WWM/edit?usp=sharing`

[10] Questionnaire: `https://docs.google.com/forms/d/1MUKfjnyXrpKdfKZ5nnhYtUxyO11aOrRDad5KT2DM-3g/viewform`

**Comparison between WordPress and the mind-mapping process.** Once both tools were used, users were able to give their opinion about the usefulness of a website configuration via a mind-mapping based DSL, by comparison to a tool such as WordPress.

### 5.2 Experimentation result

## 6 Textual representation and web designers

**Approach.** Finally, the use of our defined DSL ('CMS-CL') with the textutal tool is the

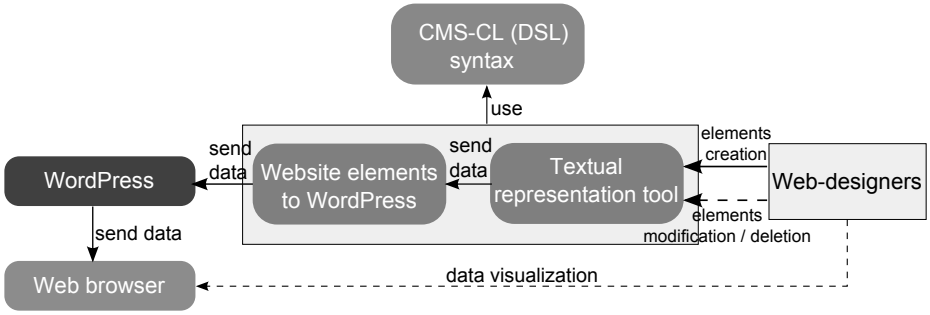


**Fig. 6.** CMS-CL and textual representation tool

same than the one presented for end users (section 2), with just one modification about the component for the textual representation.

**CMS-CL Syntax.** The syntax for the web designers tool is quite the same than the end users one : there is an identic abstract syntax, but the concrete one is different because we choose a representation which is textual[11].

A lot of tools allow textual representation, but we finally used XText [10], because it is a framework specialized on the domain specific languages development : it has all the functionnalities than parser, linker, compiler or interpreter have; but also because it is an Eclipse plugin and this IDE allows to have a reduce version of it (called an Eclipse product [11]) with just some basic functionnalities and those we wanted (in our case : CMS-CL textual representation and compilation). Another reason is the active community behind it, and its regular updates.

**Implementation** The technical part for the web designers changes just for the two first components on the client part, and uses the same stub than the end users technical part (see section 4). The server components does not change.

*The first component (1) :* Enables to write a definition of the different elements that a web designers wants to add to a web site, using the XText grammar (those we have defined) respecting CMS-CL (the output being an XText CMS-CL model).

*The second component (2) :* Transforms the specific file format of the XText CMS-CL model to a file format (XMI) usable with the stub.

[11] 'CMS-CL-to-XText' mapping: `https://raw.github.com/mallon/WordPress-MindMapping/master/Documentation/DetailedFigures/CMS-CL2XText.pdf`

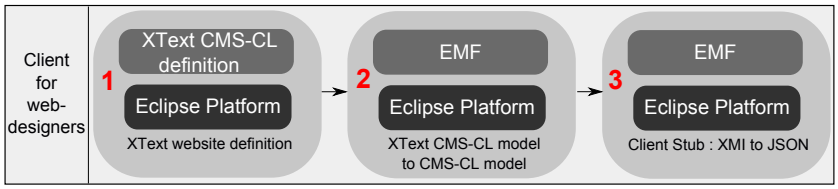


**Fig. 7.** Web designer (client) components

**Use case.** With the Eclipse product, web designers can defined their website structure, as following List.1.2 shows, textually equivalent to the web site in section 5.

```
<Website>
<adminUser elementContent={userOne}/>
<users listContent=[<User>
            <idUser content=userOne/>
            <userName content="admin"/>
            <password content="password"/>
            <userRole elementContent={administrator}/>
          </User>
]/>
          <Post>
            <idPost content=postOne/>
            <format elementCotnent={aside}/>
            <title content="'God Is An Astronaut' - Happy new year !"/>
            <content content="We would like to wish ..."/>
            <author elementContent={userOne}/>
          </Post>,
          <Post>...</Post>,
          <Post>...</Post>
...
</Website>
```

**Listing 1.2.** Web site textual representation

*Modification:* It is the same as presented in section 5.
*Deletion:* They have to use a *'deletion'* tag, as in List.1.3.

```
<Deletion>
  <usersByLogin listContent=["nameUserOne"]/>
<\Deletion>
```

**Listing 1.3.** Deletion of a user with the 'Deletion' tag

## 7 Related work

The use of a DSL to simplify the web creation for end users was inspired by the work of O.Diaz and G.Puente. In this paper, they present their own DSL - 'WSL' - for the creation of wiki via the mind mapping and the MediaWiki platform [12].

One other work [13] focus on the data conceptual modelisation and communication to end users, using respectively two kinds of specific representation : an XML Document type Definition (DTD) and an XML Document Object Model (DOM).

Concerning a simpler (graphical) development support for more advanced end users (or developers), some other paper proposed a web interface, where the programming

and the GUI creation processes are not separated [14]. Their work is based on the Lively Kernel [15] of the Sun Labs.

Other works [16, 17] focusing more on functionalities, and more precisely on the proposed web services : they have made DSLs to improve the web services management, their compositions, and their creations.

Another tool [18] enables to configure WordPress, but it is limited: indeed, it is technically only for the WordPress CMS (contrary to our tool, which could be easily applied to other CMS like Joomla or Drupal), and it uses a textual representation, which does not allow to have an equivalent quick and comprehensive overview of items added or changed.

Finally, a design tool using [19] a specific DSL was also proposed : with its specific DSL 'WebML', the tool allows designing complex Web and SOA[12] applications, by providing visual design facilities and code generation. Because they used for the modeling step a representation which is less comprehensive for end users than a mind map and also because the textual representation make easier the web designer work, our tool and concepts could be integrate in it.

## 8 Conclusion

In this paper we have presented an approach allowing a simpler web site configuration for users. *For this web site configuration*, we have used the *WordPress CMS*, and *our new DSL ('CMS-CL' : CMS Configuration Language)* to represent the WordPress configuration. Then, for the end users, we have used our DSL on top of the *mind-mapping* to allow them to manipulate the different elements and easily configure the web site. About the web designers, we have created an IDE (using an Eclipse product and the XText plugin) using the DSL to textually defined the web site configuration.

As further work, it would be interesting to provide the possibility for end users to *use our DSL in the same way, but collaboratively* : the DSL would be manuipulated online, and each authorized user could be able to manage web site elements, communicating with other users. In addition, *this collaboration* between users *could be apply on several sites*, i.e. a user could access to several online representations of different websites. The benefits given by this collaborative work platform would be the quality and speed improvement of the web sites creation. Another interesting thing, would be *to create a more generalized DSL to use it with other CMS*.

## References


1. W3Techs: Cms repartition, `http://w3techs.com/technologies/overview/content_management/all`.
2. TheWebShoppe: Pros and cons of a content management system (cms), `http://thewebshoppe.net/pros-and-cons-of-a-content-management-system-cms/`.
3. A.Van Deursen, P.Klint, J.Visser: Domain-specific languages : An annotated bibliography. Acm Sigplan Notices Journal, Volume 35, pp. 26-36. Amsterdam (2000)


[12] Service Oriented Architectures

4. G.Genova: Modeling and metamodeling in model driven development. In: Doctorate Seminar, Department of Electrical Engineering, Warsaw University of Technology, May 14-15 2009. Varsaw (2009)
5. Wordpress.org: Themes directory, http://wordpress.org/themes/browse/popular/.
6. J.Mueller, D.Polansky, P.Novak, C.Foltin, D.Polivaev, P.Cuklin, V.Boerchers, R.Wesley: Free mind mapping and knowledge management software, http://freeplane.sourceforge.net/wiki/index.php/Main_Page.
7. W3C: Extensible markup language(xml), http://www.w3.org/XML/.
8. Eclipse Foundation: Eclipse modeling framework project (emf), http://eclipse.org/modeling/emf/.
9. JSON-RPC Working Group: Json-rpc, http://json-rpc.org/.
10. Eclipse Foundation: Xtext, http://eclipse.org/Xtext/.
11. Eclipse Foundation: What is an eclipse product ?, http://wiki.eclipse.org/FAQ_What_is_an_Eclipse_product\%3F.
12. O.Diaz, G.Puente: Wiki scaffolding: Aligning wikis with the corporate strategy. In: Elsevier, Volume 37, Issue 8, December 2012, pp. 737-752. San Sebastian (2012)
13. O.De Troyer, T.Decruyenaere: Conceptual modelling of web sites for end-users. World Wide Web journal, Volume 3, Issue 1 , pp. 27-42. Springer, Belgium (2000)
14. J.Lincke, R.Krahn, D.Ingalls, R.Hirschfeld: Lively fabrik a web-based end-user programming environment. Creating, Connecting and Collaborating through Computing, 2009. C5 '09. Seventh International Conference on, 19-22 Jan. 2009, pp.11-19. IEEEXplore, Kyoto (2009)
15. Hasso Plattner Institut: Lively kernel, http://www.lively-kernel.org/.
16. V.Tosic, B.Pagurek, K.Patel, B.Esfandiari, W.Ma: Management applications of the web service offerings language (wsol). 15th International Conference, CAiSE 2003 Klagenfurt/Velden, Austria, June 1620, 2003 Proceedings, pp. 468-484. Springer, Klagenfurt (2003)
17. E.Maximilien, H.Wilkinson, N.Desai, S.Tai: A domain-specific language for web apis and services mashups. Fifth International Conference, September 17-20, 2007. Proceedings, pp. 13-26. Springer, Vienna (2007)
18. D.Bachhuber, J.Martinez, M.Haines-Young, R.Swain, U.Pogson: Dictator, http://github.com/danielbachhuber/dictator.
19. R.Acerbis, A.Bongio, M.Brambilla, S.Butti, S.Ceri, P.Fraternali: Web applications design and development with webml and webratio 5.0. In: 46th International Conference, TOOLS EUROPE 2008.Springer, Zurich (2008)