The WebComfort Framework: an Extensible Platform for the Development of Web Applications

João de Sousa Saraiva, Alberto Rodrigues da Silva
INESC-ID & SIQuant,
Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
joao.saraiva@inesc-id.pt, alberto.silva@acm.org

Abstract

Content Management Systems (CMSs) are critical software platforms for the success of organizational web sites and intranets. Although most current CMS systems allow their extension through the addition of modules/components, such modules are usually relatively static, allowing only the configuration of certain parameters that constrain some aspects of their presentation.

This paper presents the architecture of WebComfort, a dynamic component-based CMS platform which allows users to manage and operate complex web applications in a dynamic and integrated fashion. The major technical details of this system are described in this paper, such as modules, toolkits, the data repository access, and the WebComfort API.

Keywords: *WebComfort, CMS, web-application, architecture, component-based, extensibility.*

1 Introduction

The worldwide expansion of the Internet in the last few years has led to the appearance of many web-oriented CMS (Content Management Systems) [5, 18, 22, 3, 23, 17] and ECM (Enterprise Content Management) [15, 2, 19, 12, 13, 16] platforms with the objective of facilitating the management and publication of digital contents.

CMS systems can be used as support platforms for web applications to be used in the dynamic management of websites and their contents. These systems present several common aspects, such as: extensibility and modularity, independence between content and its presentation, support for several types of contents, and support for access management and user control. However, there are other functionalities or technical requirements that are addressed by few CMS systems, such as true multi-language support, dynamic management of layout and visual appearance, content export, or support for workflow definition and execution. On the other hand, ECM systems are typically oriented towards using Internet-based technologies to capture, manage, store, preserve, and deliver content and documents in the context of organizational processes [2]. Nevertheless, these two content-management areas are not disjoint [15]; in fact, it is not unusual to find a CMS system acting as a repository for an enterprise's documents and contents.

WebComfort [27] is a Web Content and Application management framework that allows the management and operation of web applications, as well as the visualization of contents, through an intuitive web-based front-end. Although WebComfort is currently classified as a CMS system, our current research goal is to make it evolve to a stage where it can effectively bridge the gap between CMS and ECM systems, as well as provide support for complex real-world web applications. Based on external research in this area (such as [15, 16]) and on our own previous experiences, we have identified some issues that we believe are essential to addressing this goal, such as: (1) component-based architecture; (2) workflow specification; (3) navigation and UI specification; and (4) deployment. In this paper, we present our proposal for addressing the first issue.

This paper is structured in six main sections. Section 1 introduces the context of CMS and ECM systems, and presents the structure of the paper. Section 2 introduces the WebComfort platform's high-level architecture. Section 3 presents WebComfort's extensibility features. Section 4 discusses some architectural decisions taken during the development of this project. Section 5 presents related work that we consider relevant to this project. Section 6 concludes this paper, and presents some future work.

2 WebComfort platform

WebComfort [4, 5, 27] is a Web Content and Application management framework, promoted by SIQuant [21] and

implemented using Microsoft's ASP.NET 2.0 technology [24], that allows, in a dynamic and integrated fashion, the management and operation of web applications. WebComfort provides mechanisms for content management (structured or not) through generic Web clients (e.g., Internet Explorer, Mozilla Firefox). It also allows access from mobile devices (e.g., mobile phone or PDA), albeit in a more limited fashion. Figure 1 shows a screenshot of a WebComfort portal.

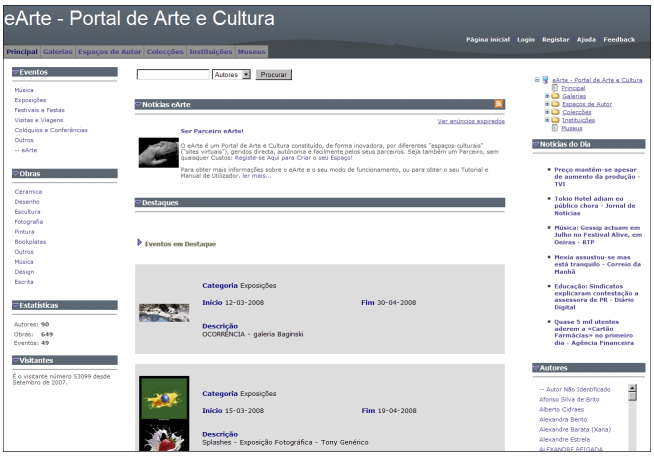

Figure 1. An example of a WebComfort portal (extracted from [8]).

The first and foremost WebComfort concept is the *WebComfort Application*, which can be defined for multiple languages and contains a variable number of *Tab*s (also called Dynamic Pages).

Tabs are ASP.NET pages that follow a predetermined structure through Master Pages [1]. Tabs can be organized in a hierarchical manner, and are dynamic because they contain *Module*s, which can be considered "component-based information presentation units" that can present several types of contents (such as HTML, images, or documents); modules are implemented as typical ASP.NET user controls, although they must subclass one of the classes provided by WebComfort. Currently, WebComfort provides several modules to address typical web site functionalities, such as Announcements, Events, Contacts, Links, Hit Counter, Image, Documents, HTML Document, XML Document, or Site Tree/Menu. Specific modules for electronic commerce, project management, multimedia, and GIS (Geographic Information Systems) are also available.

A WebComfort Application can also be associated with multiple *Role*s, because of WebComfort's flexible role-based mechanism for implementing its authorization and security policy. This mechanism has two main types of role, applicable to both tabs and modules: (1) viewing access, which determines whether the tab/module can be viewed by anyone (public tab/module) or only be specific roles; and (2) management access, which determines which roles can edit the properties and contents of the tab/module.

A WebComfort Application also contains a number of *Visual Theme*s, which can be considered as "augmented ASP.NET themes" [11] and consist of: (1) one ASP.NET Master Page, which determines the graphical layout of a tab – including Module placement – by using Content PlaceHolders; and (2) a collection of files (such as Skin and CSS files) that determine the graphical look of a tab. Each tab can have a specific theme applied (depending on what the tab's owner configures), which allows the use of different looks and layouts within the same application.

An important characteristic of WebComfort is the separation between content and presentation. Although each module has a single data model, it can provide multiple ways of presenting (and interacting with) the contents of its data model, through its flexible *Module Layout* mechanism. Besides containing a variable number of module layouts, a module can also contain a variable number of *Module Support Page*s, which can be used for the various tasks required by the module (e.g., a page to edit a certain type of content).

Finally, tabs and modules can also be associated with *workflow*s, which allow the specification of behavior for the various life-cycle stages of the managed contents, such as creation, editing, validation, publication, and eventual elimination.

Figure 2 presents an overview of these basic concepts of the WebComfort platform.

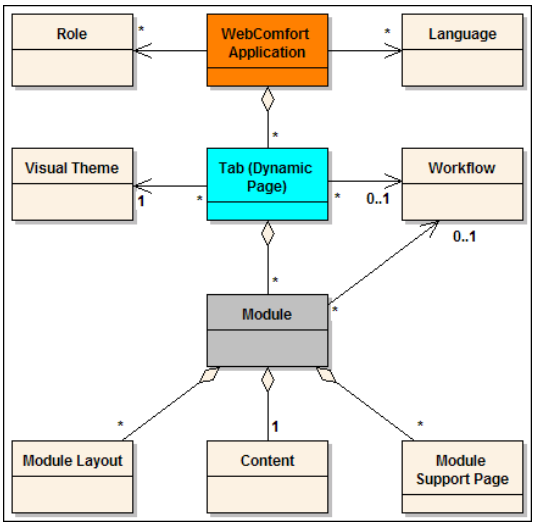

Figure 2. The basic concepts of WebComfort.

The WebComfort platform is considered a *framework* because of the features it offers for extension, such as: (1) allowing the easy installation of new module types to man-

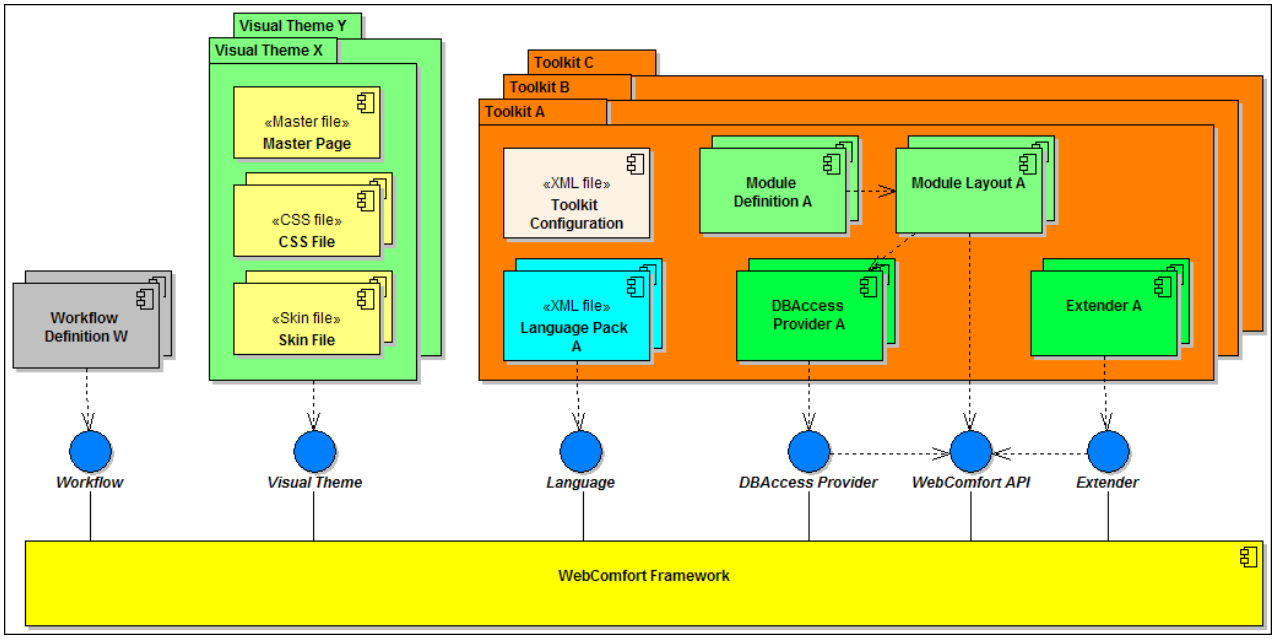

Figure 3. An overview of WebComfort's architecture features.

age and display information; (2) allowing the configuration of various behavioral aspects of the platform itself; and (3) providing an extensive API that allows module developers to create module logic that can be tightly integrated with the platform, and can even react to events that occur in the platform (e.g., module creation or installation). In addition to this API, WebComfort also introduces and supports the concept of *toolkit*, allowing developers to create "installable module packages" that can be installed by the platform itself, without requiring particular intervention from the user.

Besides these mechanisms, WebComfort also provides other features, such as: (1) Language Packs [4]; and (2) Workflow Definitions. Language Packs are XML files that provide text to use in WebComfort's user-interface; although we could use .NET's Resource Localization [24] mechanism to achieve this, these Language Packs have the advantage of not requiring any recompilation of the satellite assemblies whenever a change to a string is required. Finally, Workflow Definitions allow the application's manager to specify alternative behavior for a module; for example, although the HTML module does not support the traditional "submit–review–approve/reject–submit revision–approve–publish" workflow, the module's manager could specify (through the WebComfort interface) that the module should follow this workflow (assuming that the workflow definition was configured in the application).

Figure 3 overviews these features in the context of WebComfort's component architecture, which are discussed in the following section.

3 WebComfort extensible features

The WebComfort platform features a component-based architecture. Although Modules are the most relevant and obvious example of this fact, developers and/or web-application administrators can use additional components (such as Extenders) to customize several aspects of WebComfort's behavior and visualization. This section presents some relevant components and aspects of the architecture, namely: (1) modules; (2) toolkits; (3) extenders; (4) data repository access; (5) module actions; and (6) the WebComfort API. For a more thorough description of other WebComfort features, such as Language Packs, we recommend the reading of [28].

3.1 Modules

A *module* provides mechanisms to manage and layout a certain kind of content. For example, the Standard Toolkit's modules allow the management and visualization of typical Web contents, such as images, text, links, or even structured information such as a list of contacts, a discussion forum, or an interactive chat.

A module is always an instance of a specific *Module Definition*; the WebComfort "module definition"/"module" are conceptually similar to the traditional "class"/"instance", respectively. A module definition defines the characteristics that are common to all modules of that type, such as the information necessary to determine the classes that handle content copy/import/export.

However, module definitions also contain other components, called *Module Layouts*: a module layout defines how the module presents its contents. Thus, a module is not a programmatic construct that presents a specific kind of content in a certain way, but rather a concrete entity (because it can be created, configured, and removed) that establishes a mapping between the contents themselves (obtained from a database, for example) and a certain way of presenting those contents (the module layout). Figure 4 presents the relationships between module definitions and modules.

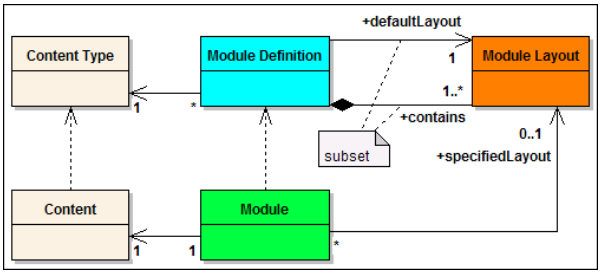

Figure 4. The relations between Module Definitions and Modules.

Each module can select one of the layouts that are contained within its module definition as its "selected layout". Similarly, each module definition must designate one of its module as the "default layout": this layout will be used when a module that instantiates that module definition does not specify the layout to be used.

Of course, from this discussion, when a developer creates a new user-control representing a module, the developer is not creating the module *per-se*, but rather a module layout. Nevertheless, for simplicity reasons, in this text we use the term "module" to refer to the module layout that is the current default of the corresponding module definition, unless explicitly stated otherwise.

WebComfort also provides a number of C#/.NET attributes that can be applied to the source code of module layouts (and other programmatic entities, such as helper classes and/or methods), specifying various aspects such as: (1) the "module definition unique identifier", a string (typically a GUID) that uniquely identifies the module definition; (2) methods that handle the copy/import/export of module contents; and (3) the module layouts already defined by the developer.

To quickly install a module definition, the portal administrator needs only to enter the "module definition unique identifier" in the module definition installation page, and all the information contained in these attributes will be automatically detected and used to configure the module definition. Of course, the administrator is free to alter any of the information that is automatically detected by the platform, or even reset the information to its defaults (the detected values) in the event of some mistake during editing. Listing 1 presents two examples of the usage of these attributes: a module layout, and a class and method that handle the export of that same module's contents.

3.2 Toolkits

From a simplistic perspective, a *Toolkit* can be considered as an integrated collection of modules that provide specific functionalities. However, from a more technical perspective, a toolkit can be better defined as "an installable set of modules (i.e., a set of module definitions and corresponding module layouts) and accompanying resources (e.g., images, Language Packs) that can be installed by the WebComfort platform without particular user-interaction", such as manual addition of module definitions or manual editing of the "web.config" configuration file [24].

Listing 1. Examples of the usage of the attributes defined by WebComfort.

```
1  [WebComfortModuleDefinition("A4CAB2D5-076B-42b1-9321-4
       E22FF5C0E3E")]
2  [ModuleLayout("Image", "Modules/StandardToolkit/Image/
       ImageModule.ascx", TargetPlatformSupport.Default,
       TargetPlatformSupport.No)]
3  public partial class ImageModule :
       WebComfortBaseModuleDesktopLayout {
4    protected override void OnLoad(EventArgs e) {
5      base.OnLoad(e); // Makes some platform-related
           security checks
6
7      // Take appropriate actions here
8    }
9  }
10
11 [WebComfortModuleDefinition("A4CAB2D5-076B-42b1-9321-4
       E22FF5C0E3E")]
12 public static class ImageExport {
13   [ModuleCopy]
14   public static void ModuleCopy(int sourceModuleId, int
         destinationModuleId) {
15     // Copy the contents of the source module into the
           destination module
16   }
17 }
```

Any toolkit must contain a configuration file that specifies how the platform should handle the various steps of toolkit installation and removal (e.g., the module definitions and module layouts to add, the resources to copy, the Language Packs to install). This configuration file also specifies the Extenders or Data Access providers (explained in the next subsections) to add to the platform.

3.3 Extenders

WebComfort *Extenders* allow to add functionality to the web application, without replacing the functionality that is already provided by the platform. An Extender consists

of a class which inherits from `WebComfortExtender` and overrides the methods corresponding to the events (e.g., "user registered", "toolkit installed") that the developer wishes to add functionality to.

Extenders are added to WebComfort by using the Provider pattern [10], allowing portal administrators to dynamically add or remove extenders by simply editing the "web.config" file, without requiring any changes to code.

An example of this mechanism is the WebComfort Sandbox [26], in which users can test the platform to see if it suits their needs. The Sandbox consists of a typical WebComfort installation with an extender (the Sandbox Extender) that addresses the "user registered" event: when a user registers with the Sandbox, the platform itself creates the user entry and triggers the "user registered" event on all the available extenders. The Sandbox Extender then creates a tab for the newly-registered user and some HTML modules that provide an initial explanation of how to manage WebComfort.

3.4 Data repository access

One of the main issues during WebComfort's development was access to its data repository: although WebComfort itself initially supported only SQL Server, there were also versions of WebComfort that had been migrated to Oracle. This easily led to a maintenance nightmare, because the code-bases for these versions were not the same, and so changes to one of the code-bases should also be inserted into the other code-base, or we would have some versions in which some already-solved bugs would still be present. However, the access to WebComfort data could not be done in a single code-base; even if we used ADO.NET technology to abstract ourselves from data-source details (such as the different SQL flavors used in today's available databases), there are still data-sources that are not supported by this technology.

So, instead of having several code-bases for WebComfort (one for each supported data repository type), we have separated the application logic and the data access logic into separate layers, so that data repository access would follow a component-based approach, still based on the Provider pattern [10]. The WebComfort platform only defines an abstract class that must be inherited by all WebComfort data access providers; this abstract class itself only provides references for a number of interfaces (e.g., `ITabsDB`, `IModulesDB`), which must also be implemented by the data access provider and provide a number of methods that are invoked by the platform. This approach allows us to add WebComfort support to other data repository types, such as a file-based repository or a relational database (e.g., Oracle, MySQL, MS SQL Server, PostgreSQL), with relatively small effort. Figure 5 illustrates WebComfort's data access architecture, with an example of a concrete implementation for SQL Server (methods are not represented for figure simplicity).

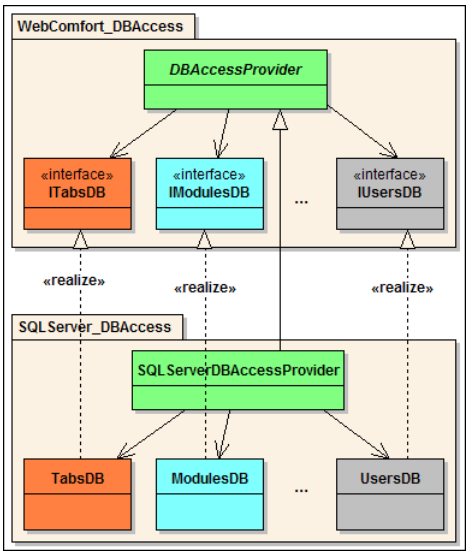

Figure 5. The WebComfort Data Access architecture.

Additionally, WebComfort allows the usage of this data access provider mechanism by toolkits: all data access providers must specify both the data-source type that is targeted (e.g., SQL Server, PostgreSQL) and the "function" that they address (e.g., Platform, Standard Toolkit). Thus, any toolkit can use this mechanism as a framework to provide functionality that is available for a wide number of data-sources.

3.5 Module actions

*Module action*s allow developers to specify a set of operations that are allowed in the context of a certain module. These Actions are simple classes that inherit from `WebComfortBaseModuleDesktopLayout.Action` and override some of the available methods. These methods control all aspects of the action, such as the conditions necessary to display or enable the action, what to do when the user invokes the action (client-side, through Javascript, or server-side), or even how the action should be rendered in the browser (e.g., as a simple text string, or as an image button).

The example presented in Listing 2 presents a simplified example of such an Action (in this case, an action to redirect to a printer-friendly page defined by the module), specifying both a condition that must be verified in order for the action to be presented to the user (in the `IsAvailable` property) and the instructions to execute on the server when the action is invoked by the user (in the `ProcessClick` method).

Listing 2. Example of a WebComfort module Action.

```
public class PrintAction : WebComfort.API.
    WebComfortBaseModuleDesktopLayout.Action {
  public PrintAction(string description, string
      actionUrl, WebComfortBaseModuleDesktopLayout
      control)
    : base("Print", description, "Print", "~/images/
        print.gif", actionUrl, control) {}

  public override bool IsActionAvailable {
    get { return base.IsActionAvailable && Control.
        ModuleConfiguration.AllowPrinting; }
  }

  public override void ProcessClick() {
    Control.SaveAndRedirect(string.Format("{0}?ModuleID
        ={2}", ActionUrl, Control.ModuleId));
  }
}
```

Actions are then added to the module itself by overriding the module's `GetModuleHeaderActionList` and/or `GetModuleFooterActionList`, which return a list of actions that should be displayed in the module's header or footer, respectively. Note that it is possible for different module layouts to present different sets of actions. Figure 6 presents a screenshot of some actions that can be displayed in the header area of an HTML module.

Figure 6. Screenshot of some actions in the module header.

3.6 The WebComfort API

Finally, WebComfort makes available an API, which includes the Extenders and data access providers already presented, as well as a large number of classes and methods that allow modules and toolkits to interact with the platform and take advantage of its concepts.

This API exists because, from our previous experiences, we concluded that creating a module/application on top of a framework that does not provide a rich-enough interface with which applications can interact, often leads to the existence of two applications: the base platform and the developed application itself. This situation carries some disadvantages: (1) the developed application is usually required to have code which is already present in the base platform; (2) to access the information defined by the framework's concepts (e.g., user information, list of existing tabs), the application must often use "hacks", such as directly accessing a specific database; and (3) the need to enter data in the application which has already been inserted in the framework (e.g., a user's address).

The simple example presented in Listing 3 illustrates the usage of the API in the context of a WebComfort module (more specifically, the module accesses the "Languages" and "Module Settings" facilities of the platform, through the `LanguagesAPI` and `ModuleSettingsAPI` classes respectively).

Listing 3. Examples of the usage of the WebComfort API.

```
protected override void OnLoad(EventArgs e) {
  base.OnLoad(e);

  int languageId = LanguagesAPI.GetModuleLanguageType(
      ReferenceModuleId);
  if(languageId != LanguagesAPI.TYPE_SINGLE_LANGUAGE) {
    languageId = LanguagesAPI.GetCurrentLanguage();
  }

  Dictionary<string, string> settings =
      ModuleSettingsAPI.GetModuleSettings(
      ReferenceModuleId, languageId);

  // Do something with the settings
}
```

4 Discussion

Most of the aspects presented have the objective of promoting the decoupling of WebComfort's various functionalities, in order to make WebComfort a true component-based platform with as few dependencies between components as possible. Examples of this decoupling are the "module definition unique identifier", the module layout mechanism, and the data access provider. Paramount to some of these aspects was the usage of the Provider pattern [10], which is a mix of the Abstract Factory, Strategy and Singleton patterns [9]; this pattern allows the runtime configuration of an ASP.NET web application by simply changing the "web.config" file [24] to specify which component (i.e., a provider) should be used to provide the application's behavior.

The usage of the "module definition unique identifier" and the attributes provided by WebComfort brings the added advantage of removing the dependencies between the code and the module definition itself. A good example of this fact is the connection between the module definition and the classes and methods that handle the copy/import/-export of a module's content: the module definition does not need to specify the namespaces and names of the classes and methods that handle these operations, because these are obtained at runtime (by using .NET's reflection mechanism) through the attributes provided by WebComfort (as shown, for example, in Listing 1).

As for module layouts, the greatest advantage of this mechanism is that it allows modules to truly separate the

content itself from the way it is presented: if an administrator decides that a module's contents should be displayed in a different way, all the administrator has to do is select a different module layout for that module, without requiring the instantiation of a different module and subsequent migration of contents between modules.

WebComfort's greatest advantage over previous versions (and even most other CMS systems available) is perhaps its new data access layer. Because of WebComfort's usage of data access providers to decouple the application logic from database-specific details (e.g., SQL-flavors, the usage of ADO.NET or a specific database adapter), WebComfort is now available in SQL Server as well as PostgreSQL, and we are satisfied with the results, as the effort to support PostgreSQL took only two days to complete.

These aspects have been validated in some real-world applications. Of these, we highlight "Portal e-Arte" [8], which is a portal designed to support the Portuguese artist community (see Figure 1); this portal is built as a web application based on the WebComfort platform, and makes use of most of the aspects presented in the previous section.

Finally, and from our experiences in developing web applications over WebComfort, we believe that its current status makes it adequate for supporting relatively-complex web applications. WebComfort features a component-based architecture, and its API facilitates the task of deploying new modules and/or toolkits onto the platform. Nevertheless, development of WebComfort modules is still a manual task, and we plan to improve this issue through an MDE-based approach [20].

5 Related Work

Although WebComfort component-based architecture provides some degree of flexibility that cannot be easily found in other CMS systems, there are a few CMS systems that address some of the issues presented in this paper. In this section, we present the CMS systems that we consider most relevant in this area.

DotNetNuke [6] is an open-source CMS system written in Visual Basic.NET, powered by ASP.NET 2.0 and Microsoft SQL Server. DotNetNuke also uses the Provider pattern in various areas, such as authentication and database access, and it has a wide variety of modules for various purposes. However, it does not provide concepts such as toolkits (although modules can also be installed through the web interface) or extenders (which means that modules cannot take appropriate action when platform events take place). It also does not provide the concept of module layout.

Drupal [7] is an open-source CMS system built in PHP, supporting Apache, IIS, MySQL and PostgreSQL technologies. Like other CMS systems of its kind, it features a wide variety of Modules that provide several different functionalities, as well as a role-based permission system. However, Drupal is more oriented towards the management and visualization of contents themselves (through modules) than towards customization of the platform itself: Drupal does not provide ways for the developer or administrator to override/extend certain aspects of the platform functionality (without changing its source code). Drupal also provides the concept of action, in a fashion similar to WebComfort's own module actions.

Joomla! [14] is another open-source CMS system built over PHP and MySQL. It also features a wide variety of Modules that provide several different functionalities, as well as a role-based permission system. However, unlike Drupal, Joomla! was built to also support extensions at the core level, allowing administrators to add base functionalities (such as multi-language or indexing support) by adding plugins to Joomla!, without requiring recompilation of the platform.

Finally, Typo3 [25] is another open-source CMS system built in PHP, supporting Apache, IIS, MySQL, PostgreSQL, Oracle, and Microsoft SQL Server technologies. Management is performed via a back-end interface, instead of a front-end, although both interfaces are web-based. Typo3 was built to easily support extensions at its core level (through the Typo3 Extension API), allowing administrators to add functionalities (such as modules, application logic, or third-party applications) to Typo3 installations.

6 Conclusions

The recent expansion of the Internet has originated many CMS and ECM systems that aim to facilitate the management and publication of digital contents. These platforms, which tend to be modular, extensible and versatile, can be used as support web applications for the dynamic management of websites and respective contents. Nevertheless, there are still functionalities that are only addressed by a small number of these systems, such as workflows or true multi-language support.

WebComfort is a Web Content and Application framework that allows the management and operation of web applications in a dynamic and integrated fashion. It provides mechanisms for content management, through generic Web clients such as Internet Explorer or Mozilla Firefox.

In addition to content management, WebComfort also allows developers and administrators to customize several aspects (such as presentation, layout or behavior) of platform. This paper presented and discussed some features of WebComfort's component-based architecture, namely: (1) modules; (2) toolkits; (3) extenders; (4) data repository access; (5) module actions; and (6) the WebComfort API. These features have already been used and validated in some toolkits and applications, and there are currently sev-

eral toolkits being produced to address various areas, such as social network analysis, geographic information systems, eCommerce, or document management.

As for future work, we plan to create an MDE-based approach that allows developers to create complex web applications in a quick and efficient manner; this approach will also be able to take advantage of WebComfort's current extensibility mechanisms.

Also, we intend to add more extensibility and configuration points to the platform. One of our priorities is to add a configurable authentication mechanism (based on the Provider pattern) that allows us to obtain users and roles from a variety of sources (e.g., an LDAP server), similar to the mechanism that is already present in DotNetNuke. Another aspect that we would also like to address in the future is the specification of role-based permissions for use-cases, in both the WebComfort platform and any modules/toolkits; the current mechanism only allows the specification of viewing and editing permissions which is sometimes not enough for modules of a more complex nature. We also plan to introduce the concept of "toolkit version": although WebComfort supports dependencies between toolkits (i.e., if toolkit A depends on toolkit B, then toolkit A cannot be installed if toolkit B is not installed, and toolkit B cannot be uninstalled if toolkit A is installed), toolkits are expected to evolve over time, and WebComfort should be able to support that evolution.

References

[1] ASP.NET Master Pages Overview. Retrieved Monday 17th March, 2008 from http://msdn2.microsoft.com/en-us/library/wtxbf3hh.aspx.

[2] Association for Information and Image Management. Retrieved Thursday 20th March, 2008 from http://www.aiim.org.

[3] B. Boiko. *Content Management Bible*. Wiley, December 2001.

[4] J. L. Carmo and A. R. d. Silva. The WebComfort Project. In *Proceedings of the Second International Conference of Innovative Views of .NET Technologies (IVNET'06)*. Sociedade Brasileira de Computação and Microsoft, October 2006. Retrieved Monday 17th March, 2008 from http://isg.inesc-id.pt/alb/static/papers/2006/jc-ivnet2006-webcomfort.pdf.

[5] J. L. V. d. Carmo. Web Content Management Systems: Experiences and Evaluations with the WebComfort Framework. Master's thesis, Instituto Superior Técnico, Portugal, December 2006.

[6] DotNetNuke. Retrieved Tuesday 18th March, 2008 from http://www.dotnetnuke.com/.

[7] Drupal CMS. Retrieved Wednesday 19th March, 2008 from http://drupal.org.

[8] eArte - Portal de Arte e Cultura. Retrieved Tuesday 18th March, 2008 from http://www.portal-earte.com.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] R. Howard. Provider Model Design Pattern and Specification, March 2004. Retrieved Tuesday 18th March, 2008 from http://msdn2.microsoft.com/en-us/library/ms972319.aspx.

[11] Introducing Themes and Skins in ASP.NET 2.0. Retrieved Monday 17th March, 2008 from http://www.ondotnet.com/pub/a/dotnet/2004/08/30/themesandskins.html.

[12] T. Jenkins. *Enterprise Content Management Technology: What You Need to Know*. Open Text Corporation, October 2004.

[13] T. Jenkins. *Enterprise Content Management Solutions: What You Need to Know*. Open Text Corporation, April 2005.

[14] Joomla! CMS. Retrieved Wednesday 19th March, 2008 from http://www.joomla.org.

[15] U. Kampffmeyer. ECM – Enterprise Content Management, 2006. Retrieved Thursday 20th March, 2008 from http://www.project-consult.net/Files/ECM_WhitePaper_kff_2006.pdf.

[16] Microsoft. Enterprise Content Management: Breaking the Barriers to Broad User Adoption. Retrieved Tuesday 3rd June, 2008 from http://www.microsoftio.com/content/bpio/prospect_and_demand/ecm_wp2.pdf, June 2006.

[17] OpenSourceCMS. Retrieved Thursday 20th March, 2008 from http://www.opensourcecms.com.

[18] J. Robertson. So, what is a content management system?, June 2003. Retrieved Thursday 20th March, 2008 from http://www.steptwo.com.au/papers/kmc_what/index.html.

[19] A. Rockley. *Managing Enterprise Content: A Unified Content Strategy (VOICES)*. New Riders Press, October 2002.

[20] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, February 2006.

[21] SIQuant – Engenharia do Território e Sistemas de Informação. Retrieved Monday 17th March, 2008 from http://www.siquant.pt.

[22] P. Suh, D. Addey, D. Thiemecke, and J. Ellis. *Content Management Systems (Tools of the Trade)*. Glasshaus, October 2003.

[23] The CMS Matrix. Retrieved Thursday 20th March, 2008 from http://www.cmsmatrix.org.

[24] The Official Microsoft ASP.NET Site. Retrieved Monday 17th March, 2008 from http://www.asp.net.

[25] Typo3 CMS. Retrieved Thursday 20th March, 2008 from http://www.typo3.org.

[26] WebComfort SandBox. Retrieved Wednesday 19th March, 2008 from http://www.webcomfort.org/sandbox.

[27] WebComfort.org. Retrieved Monday 17th March, 2008 from http://www.webcomfort.org.

[28] WebComfort.org – Documents. Retrieved Thursday 20th March, 2008 from http://www.webcomfort.org/Documentos.