Building the Presentation-Tier of Rich Web Applications with Hierarchical Components

Reda Kadri[1,2], Chouki Tibermacine[3] and Vincent Le Gloahec[1]

[1] Alkante SAS, Cesson-Sévigné, France
[2] VALORIA, University of South Brittany, Vannes, France
[3] LIRMM, University of Montpellier II, Montpellier, France
r.kadri@alkante.com, tibermacin@lirmm.fr and v.legloahec@alkante.com

Abstract. Nowadays information systems are increasingly distributed and deployed within the Internet platform. Without any doubt, the World Wide Web represents the *de facto* standard platform for hosting such distributed systems. The use of a multi-tiered architecture to develop such systems is often the best design decision to reach scalability, maintainability and reliability quality goals. Software in the presentation-tier of this architecture needs in practice to be designed with structured and reusable library modules. In this paper, we present a hierarchical component model which allows developers to build (model, generate code and then reuse) this software level of rich Web applications. In this model, components can be connected via their interfaces to build more complex components. These architecture design models can be reused together with their corresponding code using an association mechanism. As shown in this paper this is a valuable feature in assisting developers to position their developed documents within the overall software design and thus enable maintaining the consistency between artifacts of these two stages of the development process.

1 Introduction & Background

There are already a few years that the debate exists about rich Web applications. However it is the article published by James Garett[4], co-founder of *Active Path*, on his blog on February 2005, which seems to have started the awakening of developers. From Google (Gmail and Google Sugest) to Yahoo (Flickr), a dozen of general public Web sites already adopted *Ajax* [1] which becomes a frightening competitor for existing rich client technologies like Flash or those emerging such as XUL and Eclipse RCP. Indeed Ajax provides the same advantages as its competitors (advanced ergonomy, etc.) but it does not impose the installation of a plugin in the Web browser.

In our research, we are interested in the development of such rich Web applications in a context of multi-tiered architectures of Web information systems. Our work aims at introducing new high-level languages, methods and tools in

---

[4] http://www.adaptivepath.com/publications/essays/archives/000385.php

this field. In that context we propose in this paper a hierarchical reusable component model to design the architecture of these applications. The proposed model is hierarchical because rich Web applications are by definition pieces of software based on elements that are organized hierarchically. A simple HTML page can include a form, which can be composed of many `input`, `select` or `text area` components. All of these components are reusable assets that can be used in order to compose other client applications. For instance, an existing component representing an e-mail user interface can be reused, customized and composed with other components to build a more complex client application, such as a user agenda. In the following section, we present the proposed component model. An example defined using this model is then illustrated in section 3. The components that we deal with vary from simple HTML elements (forms, frames, hypertext links, etc) to more complex components such as authentication, e-mail, editorial management components. We make use of reusable software modules because we argue that functional requirements evolution of these components is rare, as discussed in [9]. Starting from applications designed with this component model, we can then generate PHP and AJAX code.

Existing composition techniques of Web-based components address only the static and structural aspect to generate interfaces of applications, and much work remains to do for developers in order to implement collaborations between components. The challenge that we raise in our work is to allow developers to define compositions of interactive components in order to build straightforwardly and hierarchically rich Web applications. In section 4, we present the tools developed for implementing our proposals. Before concluding this paper and presenting the future work, we provide a comparison between the proposed work and the related one.

2 Component-based Web Application Architecture Model

Instead of proposing a new component model and introducing another encumbering architecture modeling language, we chose to reuse an existing well-known standard which is the UML notation. Indeed, UML is a modeling language which has been adopted by many software development teams in industry and academia. Proposing UML extensions, even standard ones (like UML profiles), did not appear as a good solution, because we found in the version 2.0 of the UML specification all abstractions needed in modeling rich Web applications with hierarchical entities.

2.1 Architectural Elements

The component model proposed here (referred to as AlCoWeb) introduces a set of architectural abstractions, such as components, interfaces, connectors and ports.

– **Components** Components represent Web elements at different levels of abstraction. They can be either atomic or hierarchical. Atomic components are black-boxes (which do not have an explicit internal structure) or basic components which do not have an internal structure at all. Hierarchical components, however have an explicit internal structure. They are described using component assemblies. Components or component assemblies are modelled using UML 2 components. Examples of components include HTML text fields, authentication forms, auto-complete text boxes and check box lists.

– **Interfaces** Interfaces represent public services defined by components. They are modelled using UML interfaces and can be of different kinds. **Synchronous Interfaces** are interfaces that contain traditional object-oriented operations. They are decomposed into two kinds: **Provided Interfaces** define provided services to other components. For instance, a component HTMLTextField can define an interface which provides services like getFormattedValue() which returns the formatted value of the text field, or getPage() which returns the page that contains the text field. **Required Interfaces** declare the dependencies of the component in terms of required services, which should be provided by other components. A component CheckBox can for example define a required interface that declares services like setExternalValue(). This service allows a given component to set the value of another component attribute (value of a text field component, for example). **Event-based Interfaces** represent asynchronous operations which are based on events. Each service is executed only if a particular event occurs. The implementation of such operations is mainly defined using a client-side scripting language such as JavaScript. Examples of these services include HTML button onClick, text onSelect, HTML form object onFocus, or mouseOver operations. **Checking Interfaces** group some operations which are invoked to validate contents of components. As for the previous interfaces, the implementation of these operations is frequently defined using client-side scripting languages. For instance, in an HTML form component, we could perform some checking to see whether all mandatory items in the form are completed. We could also add a checking interface to a TextField component so that we can make some format checking (well-formed dates, valid URLs by querying a DNS server component, etc). In AlCoWeb, all kinds of interfaces are modelled using UML interfaces. No extensions are needed for this purpose. We argue this is sufficient to build components without ambiguity and reuse them afterwards.

– **Ports** Ports represent a set of interfaces of the same kind and related to the same functionality. They can represent provided, required, checking or event interfaces. They are modelled with the traditional UML ports.

– **Connectors** Connectors are interaction-oriented architectural elements. They link interfaces of different components and encapsulate interaction protocols. Examples are provided in section 3. These connectors are modelled using UML connector abstractions and can be of different kinds. **Hierarchical Connectors** are bindings between a hierarchical component and its sub-

components. **Assembly Connectors** are bindings between components at the same level of hierarchy.

2.2 Assembling Architectural Elements

Configurations of the elements introduced above can be described using component assemblies. These assemblies allow developers to build applications starting from components by linking them through connectors. Additional architecture constraints can be described to formalize design decisions.

Component Assemblies Assemblies represent configurations of whole applications. Components can be modelled from scratch or reused and customized after checking them out from repositories. These components can then be bound together, using newly modelled connectors. An illustrative example is presented in section 3.

Architecture Constraints In order to describe architecture constraints, we introduced a constraint language which accompanies this component model. This language is an ACL profile for Web development. As introduced in [13], ACL is a multi-level language. It separates predicate-level from architecture-level expression. Predicate-level concepts, like quantifiers and set operations, are described using an OCL (Object Constraint Language) [12] dialect, and the architecture-level expression concepts are encapsulated in MOF [11] metamodels. The metamodel defined for this ACL profile summarizes the concepts introduced in the previous sections.

For instance, we may need to define a constraint which states that the component of name `TextField` should not be connected to more than two different components. This constraint could be defined using the ACL profile as follows:

```
context  TextField:Component  inv :
TextField . interface . connectorEnd . connector . connectorEnd
. interface . component−>asSet()−>size ()  <= 3
```

This constraint navigates to all connectors to which are attached the interfaces of TextField. It then gets all components whose interfaces are linked to the connector ends of all obtained connectors. The resulting collection (Bag) is then transformed into a set to remove duplicates. The obtained set contains even the component TextField, this is the reason why its size should be less than or equals 3 (instead of 2, as stated in the constraint of the previous paragraph).

2.3 Component Deployment

Once the development of an application is finished, we can proceed to its deployment. An application is characterized by the description file of the component assembly. There are three possible kinds of deployment. The first is called *Evaluation Deployment*, whose purpose is the internal deployment of the application

within the development team. It serves for component testing. The second kind of deployment is called *Remote Qualification Deployment*, which aims at deploying the application in the customer environment. This is performed in order to test the application by the customers before validation. The last kind of deployment is said *Production Deployment*, which corresponds to the final product delivery.

2.4 Component Evolution and Reuse

The development process which is used with AlCoWeb introduces two professions. On the one hand, *component developers* model and code components, and put them into the repository. On the other hand, *component assemblers* checkout existing components from the repository in order to build larger applications. The two professions work on separate environments and the repository constitutes the bridge between the two professions. When component assemblers need new components in order to satisfy a particular requirement, they ask component developers to develop them and put them in the repository, or enhance existing ones and add them as new component versions.

2.5 Association of Design Artifacts to Code

Every entity in the Web application model is associated to some implementation elements in the code. When we navigate hierarchically in the model, we traverse in the same manner the implementation code. We thus introduce an association link between design models and the code. These associations link a given entity in the model to the code, which is marked by the entity identifier as a comment. The Associations can also reference files or directories. We argue that this mechanism is a good practice in making relationships between views [2] of the Web application architecture (relationships between structural and physical views).

3 Illustrative Example

We developed using AlCoWeb a large set of components that implement a plethora of technologies. Examples of these components include directory access systems (Active Directory, OpenLdap, etc.), password-based cryptographic system (MD5, DES, etc.), database access systems (MySQL, PostgreSQL/PostGIS, etc.), geographical web service access (WFS, WMS, etc.), AJAX widgets based on Dojo, Scriptaculous, Rico, Google and Yahoo Ajax APIs. For reasons of brevity and space limitation, we prefer not to detail some of these components and present below a simple example.

The left side of Figure 1 depicts an example of a `TextField` component. This component provides and requires a number of interfaces and defines some event interfaces. Provided and required interfaces defined for this component are separated into PHP-specific (`PhpTxtfPrdInterface` and `PhpTxtfReqInterface`) and JSP-specific (`JspTxtfPrdInterface` and `JspTxtfReqInterface`). Event interfaces are decomposed into two kinds: events whose source is an application

user (`ClientTxtfSideEvents`) and events whose source is the server where the application is hosted (`ServerTxtfSideEvents`).

<<interface>>
JspTxtfPrdInterface

setFormattedValue()
setValue()
addError()
setVisible()
getPage()
getFormattedValue()
getValue()
getErrorsAsString()
getParent()
isVisible()
hasErrors()

<<interface>>
PhpTxtfPrdInterface
setValue()
getValue()

<<interface>>
PhpRqdInterface
setExternalValue()
getExternalValue()

<<interface>>
JspRqdInterface
setExternalFormattedValue()
setExternalValue()
addExternalError()
setExternalVisible()
getExternalPage()
getExternalFormattedValue()
getExternalValue()
getExternalErrorsAsString()
getExternalParent()
isExternalVisible()
hasExternalErrors()

<<interface>>
ClientTxtfSideEvents
onLoad()
onKeyPress()
onChange()

<<interface>>
ServerTxtfSideEvents
onValidate()
beforeShow()

<<component>>
TextField

PhpTxtfPrdInterface
JspTxtfPrdInterface
ServerTxtfSideEvents
ClientTxtfSideEvents

- name
- isVisible
- caption
- controlSourceType
- controlSource
- dataType
- format
- defaultValue
- required
- unique
- inputValidation
- errorControl
- validationRule
- validationText

PhpRqdInterface
JspRqdInterface

<<component>>
HtmlPage::Page1

<<component>>
HtmlForm::Form1

<<component>>
ListBox::Country

<<component>>
TextField::PostalCode

<<component>>
ListBox::State

<<component>>
ListBox::City

<<component>>
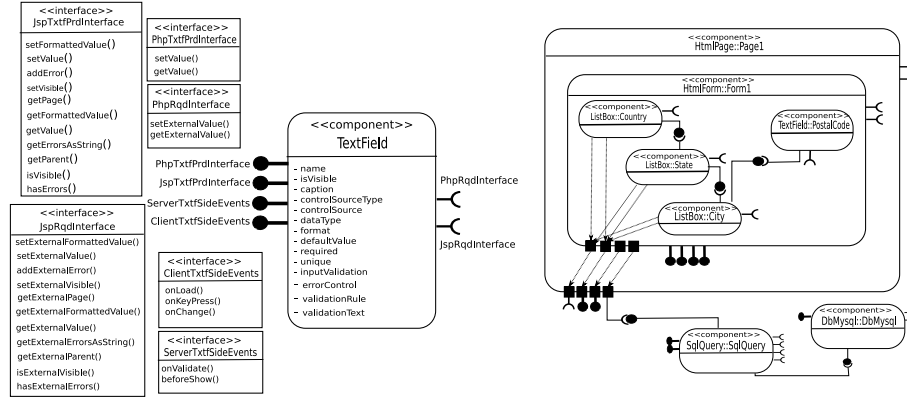SqlQuery::SqlQuery

<<component>>
DbMysql::DbMysql

Fig. 1. A simple example of a component assembly

The right side of the figure illustrates an HTML page of a Web application. This page is represented by a component (`HTMLPage`) which defines multiple required, provided and event-based interfaces. This component contains another component `HTMLForm`. Hierarchical connectors bind interfaces of the super-component to the sub-component. These are represented by dashed arrows at the bottom of the figure. The latter component contains four sub-components. The first sub-component (at the right) represents architecturally the `TextField` component introduced previously, and functionally its value represent the postal code of a city. The other sub-components are `ListBox` components, and represent lists of countries, states and cities. Suppose that all information about these geographic places is stored in a database, represented by the two components `SqlQuery` and `DbMysql`. Components are bound together through connectors defined by the developer by gluing interfaces (sockets and lollipops). The component `ListBox` representing countries connects to the database in order to get the list of available countries. As soon as this component receives a change event from the user, it executes the operation `onChange()` which updates the second `ListBox` component. The latter connects to the database using XMLHttpRequest in order to get the states of the chosen country. The same events occur for the last `ListBox` component. This Ajax-based functioning of the application is illustrated here using hierarchical components.

4 AlCoWeb-Builder: A tool for Component-based Web development

The environnment we developed rely on several frameworks offered by the Eclipse platform. AlCoWeb-Builder has been designed as a set of plug-ins that allows to separate the underlying component model from the graphical editor itself. The following section presents which frameworks have been used to develop this tool, and how they communicate to process from model design to code generation.
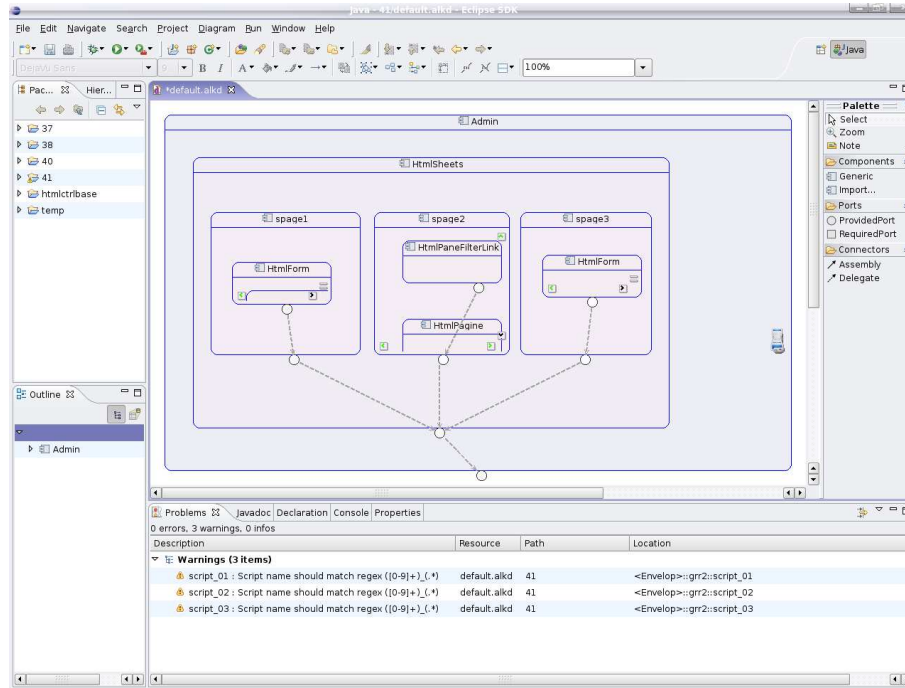

Fig. 2. Screenshot of AlCoWeb-Builder

4.1 Underlying Technologies

Eclipse is now a mature and productive platform. It is composed of various projects that provide frameworks for sofware development; such projects are for example Eclipse RCP (Rich Client Platform), BIRT (Business Intelligence & Reporting) or WST (Web Standard Tools), which provide support for EJB and Ajax development. Three main frameworks have been used to implement AlCoWeb-Builder:

- **GMF** (Graphical Modeling Framework [4]) offers a generative component and runtime infrastructure to produce a full-feature graphical editor based

on EMF (Eclipse Modeling Framework) and GEF (Graphical Editing Framework). On the one hand, the component model is designed using EMF functionalities and provides a set of Java classes which represents that model. On the other hand, GMF enriches GEF with purely graphical functionalities.

- **MDT** project (Model Development Tools [4]) provides two frameworks: UML2 and OCL. The UML2 framework is used as an implementation of the UML^{TM} 2.0 specification. In the same way, the OCL framework is the implemenation of the OMG's OCL standard. It defines an API for parsing and evaluating OCL contraints on EMF models.
- **JET** (Java Emitter Templates [4]) is part of the M2T [4] project (Model To Text). JET is used as the code-generator for models. JSP-like templates can be transformed in any kind of source artifacts (Java, PHP, Javascript . . .).
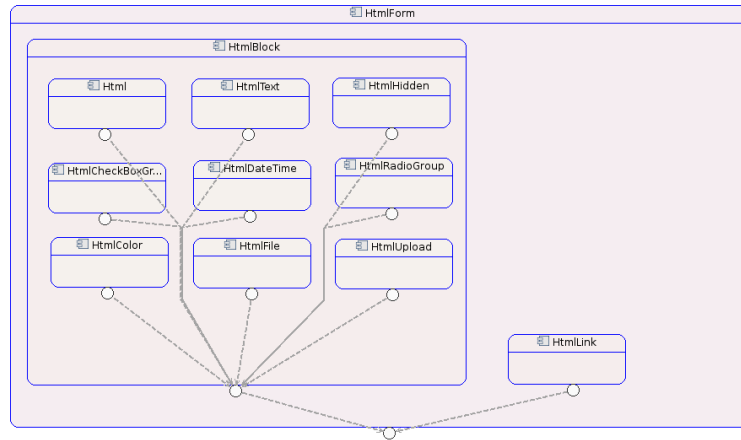

Fig. 3. Internal structure of the sub-component HTMLForm

4.2 Architecture and Functioning

The first prototype of AlCoWeb-Builder has been splitted into four main plug-ins : modeling, editing, constraints and transforming. The modeling and editing plug-ins represent the graphical editor itself, with all basic UML modeling features to design, edit and save modeling artifacts (as shown in Figure 2). It represents the generic part of the architecture which is mapped to the component model described before. Constraints plug-in allow the validation of modeling diagrams. For instance by ensuring that an `HtmlTextField` component cannot be added into an `HtmlForm` component. Code generation is represented also as a separate plug-in in order to allow multiple target languages (JSP, ASP, . . .).

In a first time, our component model has been designed using EMF facilities. This model is an instance of the EMF metamodel (Ecore) which is a Java
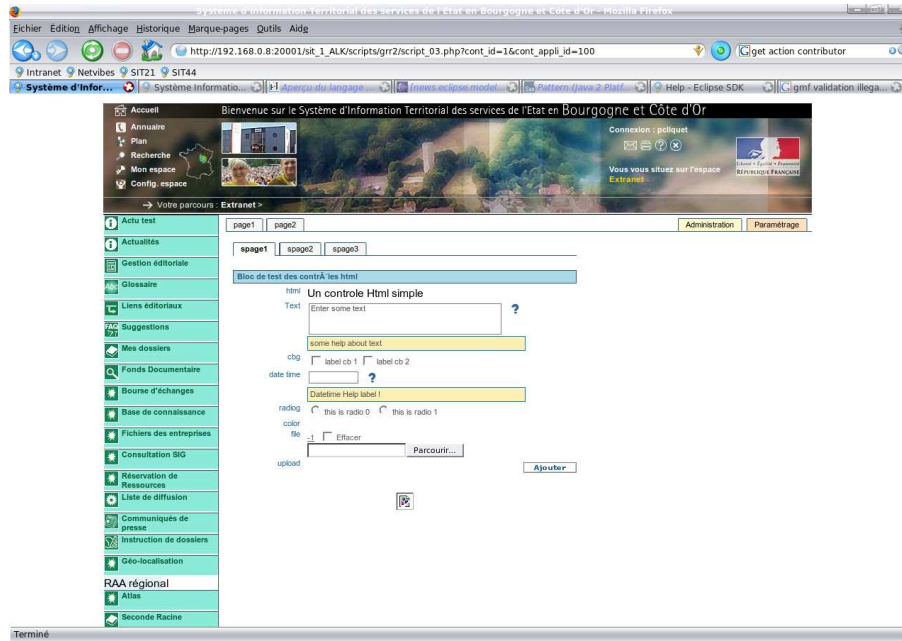
Fig. 4. Overview of the example application interface

implementation of a core subset of the OMG MOF (Meta Object Facility). In further releases, we plan to use directly the Java implementation of the UML2 meta-model, provided by the Eclipse MDT UML2 framework, as our component model.

From this model definition described in XMI (XML Meta-data interchange), EMF produces a set of Java classes. Thoses classes serve as a domain model in the GMF architecture, mainly build upon the MVC (Model View Controller) pattern. The GMF runtime offers a set of pre-integrated interesting features, such as diagram persistence, validation and OCL. In AlCoWeb-Builder, the persistance feature allows to save the diagram into two separated XMI resources : a domain file – the component model instance – and a diagram file – the notation model for graphical elements –.

EMF also adds support for constraint languages. In our tool, OCL constraints are used to validate the diagram and ensure model integrity. An OCL editor has been developed into a larger plug-in which implements evolution contracts and a quality oriented evolution assistance of the designed component diagrams [14]. This OCL editor is based on the OCL framework proposed by the Eclipse platform, but adds some auto-completion capabilities to make easier the edition of constraints for developers.

Finally, the JET2 framework provides the code generation facility. It consists of two sets of files : an input model and templates files. The input file, given in XML, is in our case a component model instance previously modelled

with the GMF editor. Templates use the XPath language to refer to nodes and attributes of the input model, and generate text of any kind. The first prototype of AlCoWeb-Builder uses those templates to generate PHP code.

Figure 3 shows the detailed internal structure of the component HTMLForm shown in the previous figure. Components are designed hierarchically and incrementally. A double click on a given component allows either to access to the architecture of this component if this one exists, or to define a new internal structure for this component. The interpretation of the generated PHP, HTML, JavaScript (AJAX) code for the whole application is depicted in Figure 4.

5 Related Work

Modeling software architectures using the UML language has already been discussed in multiple works, such as [10, 8]. In these works different types of extensions has been proposed to deal with the lack of expressiveness of UML in describing some aspects of software architectures. As in these approaches, in this paper we showed how we can use UML to model component-based architectural abstractions, but within a particular application domain which is Web software engineering. In addition, we make use of UML as it is in version 2.0 and not 1.5.

In the literature, many works contributed in the modeling of Web applications with UML. The most significant work in this area is Jim Conallen's one in [3]. The author presents an approach which makes a particular use of UML in modeling Web applications. Web Pages are represented by stereotyped classes, hyperlinks by stereotyped associations, page scripts by operations, etc. Additional semantics are defined for the UML modeling elements to distinguish between server-side and client-side aspects (page scripts, for instance). While Conallen's approach resembles to the approach presented here, it deals with traditional Web applications and not with rich Web applications which enable in some situations direct communication between presentation-tier components and data-tier components (as illustrated in section 3). The author proposes an extension to the UML language through stereotypes, tagged values and constraints, while what we propose in this work is simply the use of UML as it is. Indeed, the distinction between server-side and client-side elements is not of great interest in our case. Moreover, hierarchical representation of these elements is not considered in his work, while the presentation-tier of Web applications is by nature hierarchical.

In [6], Hennicker and Koch present a UML profile for Web Application Development. The work presented by these authors is more process-oriented than product-centric. Indeed, they propose a development method which starts by defining use cases as requirement specifications and then, following many steps, deduce presentation models. These models are represented by stereotyped UML classes and composite objects, and contain among others fine-grained HTML elements. As stated above, the work of Hennicker and Koch is process-oriented. It shows how we can obtain a Web application starting from requirement specifica-

tions. In our work we focus on the modeling of presentation models they discuss. We do not deal with the method used to obtain these models. In addition, we consider hierarchical elements in Web applications like their hierarchical presentation elements represented by stereotyped composite objects. However, presentation elements that we deal with in our work are interactive and collaborative (like forms and form objects); their presentation elements are navigation-specific (HTML pages that contain images and texts).

In [5], the authors present an approach for end users to develop Web applications using components. The proposed approach focuses on the building of high level generic components by developers, which can be reused by end users to assemble, deploy and run their applications. Components in this approach vary from form generators to database query engines. The concern is thus with whole Web applications, all the tiers are targeted here and their running environment. In the work we presented in this paper, components are presentation-tier specific and focuses on the modelling of architectures of applications based on collaborating library components. As stated previously, our approach is more product-centric than organisational-oriented like in [5].

6 Conclusion & Future Work

Alkante develops cartography-oriented rich Web applications for regional and local communities in Brittany (France)[5]. The work presented in this paper is the continuation of a former work [7] realized in this company which targeted the business-tier of these applications, and provided a new development process based on building and reusing software components. In Alkante's development team, reusing hierarchical modules is a recurrent aspect when producing geographical web information systems. In order to make easy this task we developed a component model which enables architecture modeling of rich Web applications and reusing at the same time these design models and their corresponding code. In addition, we defined a version management mechanism for AlCoWeb components. This is based on a repository which uses, as in traditional version systems, like CVS, branches and tags for their organization.

In the near future, we plan to make available AlCoWeb-Builder and our component repository in order to be enriched by the community (in an open-source development perspective) and to serve as a testbed database for the component-based software engineering community. AlCoWeb offers to the Web information systems community a lightweight language and an environment which is convivial (allowing hierarchical, incremental and reuse-based development of Web information systems) and transparent (with simple navigations between models and code).

---

[5] Alkante Website: www.alkante.com

References

1. O. A. Alliance. Open ajax alliance web site: http://www.openajax.org/, Last access: February 2007.
2. P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures, Views and Beyond.* Addison-Wesley, 2003.
3. J. Conallen. *Modeling Web Applications with UML, 2nd Edition.* Addison-Wesley Professional, October 2002.
4. Eclipse. Eclipse web site. http://www.eclipse.org/, Last access: June 2007.
5. J. A. Ginige, B. De Silva, and A. Ginige. Towards end user development of web applications for smes: A component based approach. In *In proceedings of the 5th International Conference on Web Engineering (ICWE'05)*, pages 489–499, Sydney, Australia, July 2005. LNCS 3579, Springer-Verlag.
6. R. Hennicker and N. Koch. Systematic design of web applications with uml. In *Unified Modeling Language: Systems Analysis, Design and Development Issues*, pages 1–20. Idea Group Publishing, Hershey, PA, USA, 2001.
7. R. Kadri, F. Merciol, and S. Sadou. Cbse in small and medium-sized enterprise: Experience report. In *Proceedings of the 9th ACM SIGSOFT International Symposium on Component-Based Software Engineering (CBSE'06)*, Vasteras, Sweden, June 2006. Springer LNCS.
8. M. M. Kandé and A. Strohmeier. Towards a uml profile for software architecture descriptions. In *Proceedings of UML'2000 - The Third International Conference on the Unified Modeling Language: Advancing the Standard -*, York, United Kingdom, October 2000.
9. M. Larsson. *Predicting Quality Attributes in Component-based Software Systems.* PhD thesis, Mälardalen University, Sweden, 2004.
10. N. Medvidovic, D. S. Rosenblum, D. F. Redmiles, and J. E. Robbins. Modeling software architectures in the unified modeling language. *ACM Transactions On Software Engineering and Methodology*, 11(1):2–57, 2002.
11. OMG. Meta object facility (mof) 2.0 core specification, document ptc/04-10-15. Object Management Group Web Site: http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-15.pdf, 2004.
12. OMG. Object constraint language specification, version 2.0, document formal/2006-05-01. Object Management Group Web Site: http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf, 2006.
13. C. Tibermacine, R. Fleurquin, and S. Sadou. Preserving architectural choices throughout the component-based software development process. In *Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA'05)*, pages 121–130, Pittsburgh, Pennsylvania, USA, November 2005. IEEE Computer Society Press.
14. C. Tibermacine, R. Fleurquin, and S. Sadou. On-demand quality-oriented assistance in component-based software evolution. In *Proceedings of the 9th ACM SIGSOFT International Symposium on Component-Based Software Engineering (CBSE'06)*, Vasteras, Sweden, June 2006. Springer LNCS.