APPLICATIONS COMPONENT BASED WEB ENGINEERING
IN AN INTERNATIONAL ENTERPRISE

Philipp Obreiter
Shanghai Distributed Computing Technology Center, Shanghai Jiaotong University
1954 Huashan road, Shanghai 200030, China

Guntram Graef
Telecooperation Office (TecO), Universität Karlsruhe
Vincenz-Prießnitz Str. 1, 76131 Karlsruhe, Germany

ABSTRACT

From a software-engineering perspective the World Wide Web is a new application platform. The underlying implementation model of the Web complicates the development and the evolution of Web applications. The fact that the Web is a highly dynamic and innovative environment suggests that the advantages of component based software development can be very useful for the development and evolution of Web applications. As a case study of a completed Web application project within Siemens Ltd. China, this contribution is focused on sharing experiences of applying a new component based Web Engineering development method. This includes a customized development process and an architectural framework used in this project.

KEYWORDS

Web Engineering, Web Composition, Component Architecture, Case study

INTRODUCTION

Drawing its advantages from the ubiquitous access to information and applications, the World Wide Web (Web) has successfully established itself as the by far most frequently used platform for Hypermedia applications. However, the underlying technology and the requirements towards these applications change frequently as pointed out by (Cusumano/Yoffie 1999). Still, even nowadays the development of Web Applications is ad-hoc and doesn’t make wide use of Software Engineering concepts (Fairley 1985).

From a Software Engineering point of view the Web with its unique character and special properties is a new application domain (Gaedke 1998) and dedicated support for the development and modeling of Web Applications is needed. Therefore, a new discipline named Web Engineering (Gaede/Graef 2000) has emerged to deal with these demands. Web Engineering
is the application of Software Engineering concepts to the development and evolution of Web Applications in order to achieve both a cost reduction and an increase in quality.

Furthermore, the positive experiences with Component Based Software Development and its advantages (McClure 1997; Lim 1998; Tracz 1995; Szyperski 1997) make it desirable to be able to use a dedicated component technology for the development and evolution of Web Applications. Components encapsulate secrets and celebrate information hiding as introduced in (Parnas 1972) and therefore bolster reusability and maintainability. The concept of building Web Applications from components has been introduced as Component Based Web Engineering in (Gaede/Rehse 2000):

**COMPONENT BASED WEB ENGINEERING (CBWE)** – The production of Web Applications by composing existing components using a defined process that includes systematic reuse of components and of domain knowledge.

The goal of building Web Applications through CBWE is to rely on components as units of composition. This also requires a defined development process with dedicated support for reuse of prior solutions (existing components) and experience (domain knowledge).

To evaluate the usefulness of CBWE for successfully developing Web Applications, we applied its concepts during a project in a large international enterprise. In the following sections we will explain how we applied CBWE and what kind of insights we gathered throughout the project. We will first introduce a process model which allows us to set up a defined process for the production of Web Applications. Then we will take a short look at our project’s traits. These traits will lead to us to the decision to customize the Web Application’s development process. After that we’ll take a glance at the resulting architecture, implementation and deployment. We will emphasize the architectural framework as the core of our target application. At the end of this contribution we’ll summarize the experience we achieved throughout the project and the application of Component Based Web Engineering.

**WCML: DEVELOPMENT AND EVOLUTION OF WEB APPLICATIONS**

Component Based Web Engineering relies on an adequately defined development process. Processes proposed in the past, including the OOHDM (Schwabe/Rossi/Barbosa 1996) and JESSICA (Barta/Schranz 1998), lack reuse support and are often only suited for a certain kind of Web Application. Furthermore, these processes and methods only focus on object oriented paradigms and thus do not directly induce the use of components. To address these inconveniences, (Gaede/Graef 2000) introduce the Web Composition Process Model (WCPM). It does not define a specific development process, but it is rather a framework in which arbitrary development processes like OOHDM can be integrated. In addition to that, the WCPM directly supports reusability through its component reuse management. It is an open process model suitable for a large variety of applications. In the following, a brief introduction will be given.

Based on the spiral model (Boehm 1988), the Web Composition Process Model supports the evolution of WebApplication in short development cycles. Figure 1 shows that according to the WCPM, building and maintaining an application is an endless cycle of evolution analysis and planning, design and realization. However, an evolution cycle does not always consist of just one homogenous process. It is usually split up into several sub-processes by hierarchically decomposing it. The WCPM regards every component as to be developed within its own development process. Every sub process is the development process of a component. There is no
restriction of the type of process used as a sub process. In figure 2 for example, the left component is developed according to the waterfall model (Sommerville 1982), but any other kind of development process including the spiral model itself would be applicable, too. In the latter case, the WCPM is a fractal extension of the spiral model. It allows the most suitable development process to be used on the component level and therefore is an open process model. Hence, WCPM isn’t restrained to a certain class of Web Applications.

![Figure 1: WCPM’s notion of evolution](image1)

Due to the openness, a reuse process is applicable as a sub process, too. Nevertheless, a mechanism is needed to solve the problem how to synchronize the sub processes and how to manage artifacts like existing components. Such a mechanism is thus responsible for holding the whole overall process together and to leverage coordinated and explicit reuse. In (Gaedke/Rehse/Graef 1999) a component repository was introduced as a mechanism for reuse management. It manages the storage of components and offers a variety of methods for component access and retrieval.

![Figure 2: Example of the WCPM](image2)

Therefore, it is decided at the beginning of a component’s own process, how the component will be developed. It is first inquired through interaction with the reuse management, whether an already existing component can be used by component reuse and adjustment. Only if no
reuse seems to be possible, a suitable development process is chosen for the component’s development. Figure 2 visualizes this concept by showing two components with the first one being developed according to the waterfall model and the second one through reuse. Both processes collaborate with the reuse management for artifact exchange and thus can be coordinated and synchronized.

The problem of retrieving components for potential reuse within the component repository has to be resolved, too. Components’ characteristics and properties together form the metadata of a component. In case of reuse, the person seeking for reuse and looking for an existing component (component consumer) is only able to formulate a query about the metadata and not about the component itself. This is because she will know about conditions the desired component fulfills and not about the component itself. The reuse management has therefore to store both, the component and its metadata. The model of metadata however should not be tightly bound to the representation in order to allow queries about components to be performed within the appropriate model. Therefore, the representation of a component within the component store and its model of metadata should be decoupled. This kind of separation is a well known design pattern (Gamma et al. 1995), in case of applying it to objects it is called a bridge. The producer of a component is supposed to provide its representation within several metadata models in order to leverage the probability of the component consumer being successful in retrieving it. In case of the component producer wanting his components to be reused, he would have to perform this work anyway, so that there is no additional overhead if providing metadata together with the component.

TARGET SYSTEM AND FUNCTIONALITY SCOPE

A Web Application named “WebWizard2” had to be designed for Siemens Ltd. China. Its departments were looking for tool support for their internet and intranet site designers, in order to close the gap between the functionality they require and that available by WYSIWYG editors like Frontpage (Microsoft 1996). They especially requested a three tier change management system (Jones 1996) to be set up and time validated publishing to be supported. Furthermore, even though applying the corporate look-and-feel corporate guidelines, the customizable parts had still to be fully manageable and editable by the site designers who might lack detailed technical knowledge. That is why the “WebWizard2“ is a tool which supports the development and management of web sites that tend to have a complex navigational structure but do not heavily rely on complex program logic. The tool is not required to support the development of full blown Web Applications.

The stakeholders scheduled the Web Application to be medium sized (Roche 1994). To take advantage of the staff already trained and experienced with the Microsoft Internet Information Server and Active Server Pages, the application should be built upon that technology.

DEVELOPMENT PROCESS

At the beginning of the project, neither component artifacts nor a component repository were existing. The only artifact was a Web Application already implementing parts of the functionality that our application was requested to provide. We based our acquisition of domain knowledge on using that initial application as a prototype by extending and evaluating it and thus applying the Rapid Prototyping Development Model (Brooks 1988).

For the development process of our Web Application, we decided to employ a slightly modified version of the Web Composition Process Model (WCPM) described earlier. However we
didn’t opt to build a component repository and thus to clearly separate model and representation as suggested by WCPM. This was implied by the fact that the working staff was new to component based engineering and real reuse strategies hadn’t been adopted before. Prior to our project, the identification of reuse opportunities was fully spontaneous and depended on the developers’ knowledge about past solutions. Therefore, the introduction of a file server as component store was already a big step forward. A component store fulfills the same tasks as a component repository but is more focused on the storage of the component than on querying support for components, thus tightly binding their metamodel and representation.

In our application we had to deal with a large number of orthogonal components that we needed to evolve in short development cycles both individually and as a whole. The fractal spiral model used in the WCPM allowed us to do this in an elegant way. While the traditional spiral model (Boehm 1988) unlike other models such as the waterfall model (Sommerville 1982) supports multiple iterations needed for evolving a highly dynamic application, its fractal extension allowed us to break down the development process to components and subcomponents tracking the evolution of each with its own sub-spiral. The more the number of already built components rose the more the development process of one component was likely to be feasible as a reuse process. Necessary synchronization of component development processes was achieved via artifact management with the artifacts being stored within the component store. This proofed to be quite flexible and brought a minimum of overhead.

During the first development cycles we acquired enough domain knowledge through the prototype to be able to design a architectural framework capable of being robust yet flexible enough throughout later implementation and maintenance. We used UML to describe this framework and the application site’s navigational structure, most parts of the models being directly mappable to components. We were able to use a bottom-up approach in identifying and developing the components without “gold plating” (Boehm 1991) our solution by having tightly investigated the problem domain through the prototype.

ARCHITECTURE AND PATTERNS

The changes of requirements towards the application force the maintenance team to frequently change parts of the system. In order to prevent an architectural drift and to enable quick, effective and non error-prone maintenance work, we designed a robust yet extensible architectural framework. In addition, we documented where we used which design patterns to improve the communication between the designers and the maintainers of the system (Gamma et al. 1995, Prechelt/Unger 1998).

Our architectural framework thus ensures that parts logically not belonging together are provided by different components. That’s why we didn’t opt for an approach that makes heavy use of server side scripting, otherwise the above mentioned principle would have been violated e.g. by the mix of data retrieval information and design information, such as HTML code, within one page.

Apart of that, given the fact that many similar Web Applications were under development or were scheduled to be built, the reuse of parts of our system was very desirable. Therefore, the architectural framework had to ensure the reusability of its components too. Here again, a component based approach with components as the base of reuse (Gaedeke/Rehse 2000, Gaedke/Graef 2000) proved to be very effective.

All this led us to the decision to layer the application into a presentation, application and database layer, with every component providing exactly one sub-functionality of one of these lay-
ers. Orthogonal to the three layers we positioned data structure components ranging from the classical stack or tree to dictionary and collection.

In the following, the three layers will be described one by one by the design patterns they use and the design secrets of their sub-components:

The database layer hides details of the data sources and enriches their access methods. The data sources of the application are normally database or file system based, but only the components of this layer know about their concrete whereabouts. It is further hidden how the data sources are structured and how they can be accessed. The database layer is thus an abstraction layer (Gerlan/Shaw 1993) which enables a really transparent access to data sources without being troubled by the question "where is what and how is it stored?". Its components are mainly proxies and decorators. From the maintenance point of view, changing design decisions concerning data sources only comes along with local adjustment, that is, the adjustment of the component whose design secrets endorse the modified design decision.

The widening of the access methods’ functionality is targeted at leaving the building functionality of frequently used views of data sources within the database layer. This design decision is similar to the decision of sinking not basic but heavily used functionality to the Microkernel within operating system theory (Liedtke et al. 1991) targeted at better localization and performance. Ideas of data-centric programming models (Adams 1968) with their focus on how data triggers programs for their handling rather than on how programs process data, have also been applied by making up-calls to the user after the requested data has been retrieved. The component which "uses" (Gerlan/Shaw 1993) the database layer is thus able to subscribe to events fired by the database layer’s components and has to provide methods for the handling of these events. Such a mechanism is close to the combined use of the framework and visitor design pattern (Gamma et al. 1995).

The application layer provides the complete business logic. Its components deal with algorithms and object behaviour and their design secrets are how the logic is provided. Therefore, they are subject to the use of the strategy design pattern. With the database layer being data-centered, the application layer is program-centered and provides the whole Web Application functionality as service primitives.

We observed that the application layer’s business components tend to be extrinsic due to the fact that their life-cycle normally only stretches to one dialogue step.

Based on the service access point of the application layer, the presentation layer still has to tackle the problems inherent to Web Applications.

We modelled the user-machine-dialogue into three parts, namely the Process Before Output (PBO) for building up the next dialogue screen, the user’s interaction with the system and the Process After Input (PAI) for processing the data the user entered in the input masks (Buck-Emden/Galimow 1997). More specifically, in case of a Web Application a dialogue screen is a web page and the data is entered within forms or is implicitly given by the user choosing a link. Note, that there has to be a decision-maker that computes with which PBO to continue, referring to the PAI’s result (figure 3). This approach proved to be applicable to Web Applications even though the underlying web technologies induce the distinction of only two parts, namely the server and client side computation and interaction. In splitting the server side part further up into PAI, decision maker and PBO, a disciplined approach is set in place for structuring the Web Application’s dynamical aspects. Based on this concept, we were able to decompose the presentation layer into three parts.
In (Gaede et al. 1999) it is mentioned that most problems in evolving Web Applications are due to the disparity of the coarse granularity of the implementation model of the web with document resources like web pages, and the fine grained object-oriented modeling methods. To counter this resource granularity problem, we used Markup Language structure components ranging from common tables or images up to whole pages to allow fine grained modeling of pages and their sub-elements. They all have a presentation function that returns their markup language presentation, such as HTML in our case, but through adjusting those functions (without touching the rest of the code) any other markup language can be supported. These components are a by the book examples of the compositum design pattern. The structure and storage of markup language elements are their design secrets.

Factory/builder components are responsible for constructing pages or their sub-elements by initializing and combining respective Markup Language structure components. Each page of the Web Application has a corresponding factory component. However, the rest of the factory components is responsible for building up sub-page elements. This approach is close to the Web Composition approach described in (Gaede 1998). The use of the factory/builder components is clearly within the PBO-step. The design secrets of this kind of components are how the page elements and information are structured and laid out.

In addition, the presentation layer consists of control components which are responsible for processing the user input and for launching the appropriate factory used to build the responding web page. They rely on the application layer’s service access point. Apart of managing the PAI, a control component decides with which PBO to continue and then launches it by using the appropriate page factory. The resulting dialogue step scheme is shown in figure 3. A control component is not bound to process only a single page’s user input; several pages semantically belonging together and therefore complete user-machine-transactions can be modelled within a single component, again analogous to the Web Composition approach (Gaede et al.1999). In addition, every control component has a querying interface. Through its interface, it is possible to query which page’s PAI-PBO step.

A page on the Web Server is represented by a small script which only calls the control component that is able to handle the PAI-PBO step. Since these scripts have an identical structure, it is possible to create them automatically by querying the presentation layer’s control comp-
ments. We wrote a small tool for doing so. Note that thus the whole navigational structure of the Web Application site is held within the control components. We experienced that modeling the site’s navigational structure through UML’s hierarchical state-charts fits very well for sites with plenty of page invariant elements, e.g. a content bar displayed on every page. In addition, the hierarchical boundaries within the state-chart give a hint on how to identify the scope of a control component, that is which dialogue steps together form a user-machine-transaction.

Figure 4 shows a simplified fragment of the overall navigation structure. Note, that the states represent the user screens and the arrows the PAIs. The PBOs are implicitly given by the states. The chart induces the use of a control component for the modeling of the super-state “User Management”. The dialogue steps of this user-machine-transaction will be completely hidden by this control component.

![State-chart fragment](image)

**Figure 4: State-chart fragment**

A coarse summary of the architectural framework is shown in figure 5. There are six stereotypes of components with three belonging to the presentation layer. The arrows indicate the use relations between the types of components. Note, that the use relations are reflexive throughout the stereotypes. For example, a factory responsible for the construction of a whole page will have to make use of the database proxies for getting the actual data, of a markup language structure element to represent the page and of several more factories to build up the sub page elements.
Figure 5: Coarse scheme of the architectural framework

Since we chose to adopt a bottom-up approach, as described in previous section, the components close to data, which are the data structure elements and the database proxies, are developed first. Afterwards the markup language structure elements, which directly rely on the data structure elements, are constructed together with the business logic elements. The pair of components developed in these first two steps, are fully orthogonal. Only at the end, the PBO and PAI components are non-orthogonally produced. The development order is shown in figure 6. One benefit of layered architectures is to be able to successively build and test. Note, that we made use of this benefit throughout the development steps.

Figure 6: Development steps

Let’s take the following situation as an example:

A proxy for a data source has to be constructed. The task of building a proxy for this kind of data source is well understood and unexpected feedback between different phases should be at a minimum. Therefore it is decided to use the waterfall model for the proxy’s development.
During the architectural design step it is figured out, that a data structure component has to be ready before the component is implemented. That data structure component has to provide the functionality of trees and extensive querying methods of the tree’s attributes. In fact, the development of a tree component has already begun before, since we already expected that it would be of use sooner or later. A tree component had already been built in the past but it lacks of certain important querying methods. Therefore the tree component development process is figured out as feasible as a reuse process. The third component to be built is a PAI control. Since we are uncertain about several risks, it is decided to apply the spiral model for its development. During the first cycle we analyze these risks, and we identify as the biggest risk that the underlying data source performance could not be enough. This leads to the decision to build a prototype with a minimum of functionality during the second cycle. In order to evaluate and complete this prototype in the second cycle, it has to be tested with the proxy component of the data source which is however not yet ready. Figure 7 shows how these three development processes of the components are synchronized by the reuse management. Note how the development processes always fit perfectly to the components to be developed, all this enabled by the WCPM being open.

**Figure 7**: Example of development process synchronization

**IMPLEMENTATION AND DEPLOYMENT**

We selected a straightforward implementation of the components in ActiveX being aware of the portability considerations, since the deployment environment of a Microsoft Internet Information Server put severe limitations to other approaches. We used Visual Basic for the coding of most components except of the data structure components implemented in Visual C++ for better performance. Besides, a two tier change management system (Jones 1996)
was set up to prohibit changes within the development system from affecting the production system. During the implementation the fact that the middle tier, the quality assurance system, was missing didn’t harness the reliability because the application was only of medium size (Roche 1994). Even after two months of maintaining, we haven’t observed that a middle tier is really needed.

Since the Web Application was targeted at a new problem domain, its users were new to such a kind of application and had not filed all their requirement request during the earlier stages of the development process. That’s why the deployment of the Web Application came along with several unforeseen change request. Nevertheless, the architectural framework was flexible enough for quick and reliable functionality enhancement. This was partly due to the flexibility of restructuring data sources, a flexibility achieved through the database abstraction layer. The customers were very satisfied of their requests being taken into account so fast.

![Figure 8: Screenshot of the Web Wizard2 application](image)

CONCLUSION

We took over a place where prior projects which focused on building Web Applications either were completed far behind schedule or contained major bugs. Even worse, after completion these applications were difficult to maintain and extend, because they tent to fall apart after two or three months of maintenance thus demanding for full redesigning and recoding. That is why tried to work out a flexible but robust architectural framework for components and to draw benefits from applying a suitable adjustment of the Web Composition Process Model. By doing so, we were able to complete our development project within half the time scheduled. Unbelievable as it might sound, our Web Application has not shown a single major bug until today, even though new functional requirements reported by the clients during the deployment phase were swiftly added on a tight schedule. Unfortunately, no detailed statistics with concrete numbers were available from prior projects for better comparison.

Nevertheless, for future Web Application projects, we suggest to extend the component store to a component repository for faster retrievability of reusable components. Furthermore, we propose the use of platform independent component technology in order to avoid that the
application built is restricted to one internet server’s technology. Additionally we admit that the fully dynamic creation of every page wasn’t really needed to be set in place and that a component model like the Web Composition component model described in (Gaede 1998) which allows to decide which parts should be created dynamically is far more appropriate for saving server resources.

Throughout the development, we observed that after a period of time the working staff got acquainted to the idea of component based Web Engineering and drove their first experiences with it from our Web Application. Thus the positive effects of the completed project go far beyond itself by improving the skills of the working staff. All the stakeholders requests were quickly taken into account and realized, so that the project crew got rewarded by a unprecidated customer satisfaction.

LITERATURE


Gaede, M.; Rehse, J.; Graef, G.: A Repository to facilitate Reuse in Component-Based Web


**McClure, C. L.:** Software reuse techniques: adding reuse to the system development process. Prentice Hall, Upper Saddle River, N.J. 1997.

**Microsoft Corp.:** FrontPage home page, URL: http://www.microsoft.com/FrontPage, 1996


**Prechelt, L.; Unger, B.:** A Series of Controlled Experiments on Design Patterns: Methodology and Results. Proc. Softwaretechnik ’98 GI Conference (Softwaretechnik-Trends 18(3)), pp. 53-60, Paderborn, Germany, 1998.


**Schwabe, D.; Rossi, G.; Barbosa, S.:** Systematic Hypermedia Design with OOHDM. ACM International Conference on Hypertext’96. Washington, USA 1996

