Utilizing Abstract WebEngineering Concepts: an Architecture

Andreas Heberle, Jörn Rehse, Bernd Onasch, and Börje Sieling

B2ABC Webtechnologien für Geschäftsanwendungen GmbH,
Amalienstraße 79, 76133 Karlsruhe, Germany
[heberle, rehse, onasch, sieling]@b2abc.de

Abstract
In this paper we present an open, platform independent, and scalable architecture that implements WebComposition using an object-oriented approach in design and realization and an abstract component model as the platform of development. A web application will be introduced as a set of service components, each providing a closed part of functionality. These service components are designed for reuse to achieve an advantage in implementing applications that have similar domains of usage, thus the evolution of a web application is supported by the whole architecture. XML is used for the specification of an application and for the configuration of existing services as well as for communication. This improves the flexibility and guarantees the platform independence of the approach. All together the architecture implements a WebComposition Framework, that provides the basic infrastructure for the design and the development of complex and powerful web applications in less time than usual.

Keywords: Web Engineering, WebComposition, Component System, Web Application

1 Introduction

Web Engineering has been defined as the „application of systematic, disciplined, and quantifiable approaches for the cost-effective development and evolution of high-quality applications in the World Wide Web“ [Gaedeke, Graef00]. Its promise is the solution of problems regarding systematic reuse and the evolution of web applications.

The use of software component technologies is commonly suggested when considering how to ensure reusability, increase the semantic expressiveness of the implementation model, and how to improve the maintainability of applications. The use of components for design and implementation in Web Engineering has led to the term “Component Based Web Engineering” [Gaedeke, Rehse00].

A number of abstract models and systematic methods for a disciplined approach to application building for the web have been presented in the last years. The most prominent examples are RMM [Isakowitz95], OOHDM [Schwabe, Rossi, Barbosa96], and WebComposition.

Because it is a component based approach we chose WebComposition as the basic technology for our further work.

To utilize the results of Web Engineering research in future development projects for the web a technical support system is required that allows the usage of the abstract concepts presented in the Web Engineering models during development. Such a support system or architecture is a layer that hides the heterogeneous nature of the web implementation environment as well as the idiosyncrasies of the web implementation model (granularity, resource based, etc.). The web application developer implements against the interfaces of the architecture. The resulting implementation artifacts use the same concepts as one of the Web Engineering models (in our case WebComposition). The support system bridges the gap between the expressive power of the design and the code.

The need for technical support for bridging that gap has been known and presented for some time (“gap problem”, [Gaedke98]). Implementation technologies that address the problem, such as the WebComposition Markup Language (WCML) [Gaedeke, Schempf, Gellersen99], have been introduced. The existence of implementations of framework architectures that allow at least partially the use of Web Engineering concepts for real life development has been mentioned but the discussion or presentations of these architecture
implementations has not yet received wide attention. One of
the few exceptions is the Eurovictor framework
implementation at Hewlett-Packard (mentioned for
example in [Graef, Gaedke00]).

On the other hand there are a number of support
systems that are used for building and operating web
sites. Some of these systems were explicitly built for the
web. They often support server-side programming and
scripting such as ASP [Microsoft97], Cold Fusion
[Allaire97], or NetObjects [NetObjects00]. These
approaches favor the structuring of parts of an
application in resources [Gaedke98] hence they do not
solve the gap problem or move it to a more abstract level
only. They do not allow the implementation of
applications using the concepts of one of the Web
Engineering models.

Other support systems were not explicitly built for the
web but were adopted from traditional software
engineering. These are software component systems and
application servers that are commonly used for building
web applications. Examples for this type of support
systems are Sun’s Enterprise Java Beans [Sun00] and
IBM’s WebSphere [IBM00]. Being support systems for
component based software engineering these systems
allow the implementation according to an abstract
component model. The expressive power of the
implementation model is much higher than in the
aforementioned web support systems. Still these systems
don’t integrate all of the parts of a web application. The
focus of these systems is the business logic of a web
application. Their abstract model is not applicable to the
web specific parts of such an application hence they
address the problems of software engineering only while
leaving the problems of Web Engineering unsolved.

Existing support systems for web application
development and operation do not allow the usage of
abstract Web Engineering concepts in real life
implementations at the moment. Still these systems are
the reality of web application development today. When
trying to implement a web support system that allows
using Web Engineering concepts for development these
existing support systems must be used as a standard for
judging performance and scalability.

Summarily it can be said that implementations of
support systems that allow the usage of Web Engineering
models have not yet been thoroughly discussed while the
existing and well known support system implementations
lack full support of these models.

In this paper we will present an architecture and some
implementation issues of a support system (framework)
for building web applications according to the findings of
current Web Engineering research. The framework in
question is currently being developed under the name
websymphony [B2ABC00] at B2ABC GmbH, Karlsruhe,
Germany. At first we will give a short summary of the
WebComposition model. After that we will explain why
the architecture enables developers to utilize
WebComposition for developing web applications and
how the architecture ensures performance scalability, and
security.

2 The CBWE Model: WebComposition

Implementations of the architecture presented in this
paper allow the development of web applications using
WebComposition as component based Web Engineering
model. In this paragraph the basic foundations of the
WebComposition model will be explained.

WebComposition consists of a process model for
development and evolution as well as a component
model. Starting from a reusable empty frame the
application grows according to the WebComposition
evolution process model. The empty frame which is the
basis of any application and remains its skeleton through
its life cycle is called “evolution bus” in the
WebComposition model. The parts added to the
application are components as defined in the
WebComposition component model. Code artifacts are
modeled independently from underlying resources (files)
as uniform components.

The WebComposition component model is object-
oriented. It supports object-oriented concepts such as
(prototype-instance) inheritance and aggregation.
WebComposition components can be of arbitrary
granularity.

A domain component in the WebComposition model
is called a service. Services are the primary modeling
abstraction in WebComposition. A service should
integrate all aspects of a web application. In
WebComposition five different characteristics of a web
applications are considered. Each characteristic is
expressed in a WebComposition component. The
characteristics can be used to classify components and
divide them into groups. Being a component in an object-
oriented component model a service can consist of
several other components. A service is composed of one
component (possibly an empty component) from each
group. In WebComposition the following
groups/characteristics are used:

- **Content:** Components from this group supply
data to services. It is not defined how data is
stored and how it is retrieved.

- **Layout:** Components of this group are layout
templates defined in an object oriented way.
Commonly these components encapsulate markup language code (such as HTML or WML).

- **Navigation**: Separating navigation from layout allows modeling navigational patterns such as “guided tour” [German, Cowan00].

- **Process**: These components implement the business logic needed to execute an interactive service.

When implementing a framework that supports the service abstraction one of the main concerns is the integration of the different underlying technologies (heterogeneity) that play a role in the categories into a unified component model. In the following paragraph we will present an introduction to the architecture of such a framework and we will show how the framework allows working with the service abstraction.

### 3 The Architecture

In this section, we introduce the implementation architecture of websymphony. First, we outline the scenario of web applications from the technical point of view. Then, we describe the implementation packages that build the core of our framework and explain how they can be used to implement services. The composition of services is an essential point. We present several techniques for the implementation of this service composition. Together with a configuration mechanism based on XML, we are able to implement the WebComposition theory.

First of all, we have to describe our scenario of web applications. A client (browser) communicates via an internet protocol with an application that runs on the server. This application is implemented component based. Thus, the heart of the server system is a component system together with an application server that handles creation and access of components. Our framework provides the components to build up a web application. Figure 1 describes the scenario graphically.

#### 3.1 Packages of the Framework

Applications in websymphony consist of interacting services. In the following, we introduce the packages used to implement and administrate the interaction of these services in a user session of a web application. Figure 2 gives an overview of the implementation packages and shows dependencies between them. We distinguish communication, runtime, external, and extensions packages. In the following figures, we use the graphical notation of UML [UML00].

#### 3.1.1 Communication

- **External Communication** implements the communication of applications or services that run on different machines. Means that it supports different communication protocols with interfaces.
- **Internal Communication** is a package that implements the communication facilities for components or services respectively. Its functionality is similar to that of an object request broker (cf. for example the CORBA...
3.1.2 Runtime

- The **Service Manager** is similar to a naming service. It has to know where to find and instantiate a requested service. The Service Manager is only required if the system provides more than one service.
- As part of the process of a service, the configuration of a **State Machine** instance implements the main business workflow and the navigation inside a single service. The State Machine is event driven and one state can be used as a high-level transaction scope.
- The **Session Handler** is the first component that receives a notification from the network. It is responsible for the spawning of a new session or assigning a request to an existing session.

3.1.3 External Engines

- The **Layout** package provides a set of different render engines to compose a requested document, e.g., an HTML page.
- The **Data Store** is an abstraction layer for persistent storage media (databases and file systems). There will be a set of helper functions that can be used by a service to store its data in a consistent and secure way.

3.1.4 Extensions

- In the **Security** package a service finds the required algorithms to en-/decrypt data.

---

**Figure 2: Implementation packages of the framework**
Additionally, it extends the external communication with secure network protocol implementation. This package can be left out as long as no secret information handling is needed in the applications environment.

- The performance of a system can be improved by adding the Scaling package, which makes the framework capable of distributing sessions in a workstation cluster. Different implementations are possible and depend on the kind of services running on the system.
- The Administration Package contains components for user administration. For example defining different access rights depending on the status of a user. Additionally this package provides a repository for services including version management. The repository supports the configuration of new applications.

3.2 Implementation of the WebComposition Model

In websymphony, a service consists of layout, navigation, content, and the process part (see figure 3).

![Diagram of parts of a service](image)

**Figure 3: Parts of a service**

The layout part is the service dependent configuration data for the render engines in the Layout package. More complex layouts may require additional methods, implementations of them also belong to the layout part of a service. The data store provides the content information. The business logic and the navigation specification together with a definition of transactions define the state machine that implements the process (see figure 4). There exist additional dependencies between the layout and the process. For example, if the process expects an event “order sent”, the layout has to define a button that creates this event. Although it is not in the responsibility of the implementation to assure consistency of the specifications, we are able to find inconsistencies by checking the validity of XML specifications and configurations.

There are three ways of defining a new service: Configuration, Composition, and Coding.

**Configuration**: The framework provides service templates that can be instantiated via an XML configuration in order to implement a new service.

**Composition**: A new service can be created by composition of existing services. We distinguish three kinds of composition.

- Services are combined by means of the hypertext language, e.g. by frames or links in HTML (Hypertext Composition). This kind of composition requires that services are independent and do not need to communicate.
- A service inherits a service or delegates control to other services (Object-oriented Composition). This kind of composition describes a 1:n relation. It is encoded in the state machine. For example, the state machine, that implements the process part of a service, can be built up by the sequential execution of processes (state machines) of other services.
- Aspects of a service have to be modified in order to fulfill the requirements of the new application (Aspect-oriented Composition). This kind of composition is the most powerful one, but it requires the modification of code. It allows the interaction of existing services as well as the interleaving execution of services. At the moment there exist several attempts to perform this code modification automatically, but in
general it has to be done by the service writer.

**Hand Coding:** Of course, it is always possible to encode services from the scratch without using existing services.

The implementation of services depends on object-oriented features of the programming language. A component system is required as execution environment for the framework. Platform dependent properties of such components are encapsulated. Thus, our framework is not restricted to one special component system, but can be implemented for any of the common component systems COM/DCOM [Sessions97], CORBA [OMG00], or Enterprise Java Beans [Sun00].

4 Validation of the Architecture

We are able to implement the WebComposition model using the above presented websymphony framework. This allows the development of web applications on a sound theoretical basis. Nevertheless, for practical use there exist additional requirements. Tool support is needed in order to simplify the development process. Performance presented architecture fulfills these requirements.

4.1 Tool Support

Together with the framework, we provide tool support for service composition, user specific configuration, and implementation of services. Since we make heavy use of XML, we include XML-Editor, -Parser, and XSL-Transformer in the framework. The composition of services is specified by the language WCML [Gaedke, Schempf, Gellersen99] which is in fact an application of XML. Thus a validating XML-editor supports the definition of the service composition. User specific configurations, e.g. layout, are specified as XML data as well. Beside the definition of specifications, the framework simplifies the implementation. We provide factories [Gamm95] in order to produce specific implementations of abstract entities and use configurable components like the state machine and the data store.

4.2 Platform Independence

![Diagram: Scaling by redirecting HTTP requests](image)

Figure 5: Scaling by redirecting HTTP requests

and platform independence of the application are essential. Security concerns have to be considered, especially in the context of commerce applications and information services. In the following, we show that the

When examining issues of platform independence it is necessary to separate the concepts of platform independence of the architecture (specification) and the implementation. The presented architecture is platform
independent within a set of well defined constraints while an implementation of the architecture may be platform independent or not depending on the implementation technology (component system) that was chosen.

The constraints mentioned are a set of requirements that an environment has to fulfill to be suitable for implementing the architecture. All of the common component systems and a transaction server or component container (EJB, COM/DCOM/MTS, CORBA) fulfill these requirements. Hence the architecture is platform independent within this set of platforms.

Another form of platform independence is the platform independence of configurations or in this case service definitions. The presented framework uses XML to define services and configure generic components. This leads to the platform independence of service definitions. Service definitions for any implementation can be moved to other implementations of the architecture without any porting.

4.3 Scalability

An integrated part of the framework architecture is to support scalable system architectures. Scalable systems are required for different reasons, like starting an application with relatively small systems and expanding the systems of the application with growing resource usage. Some applications like large shops require too many resources for one standalone system and must be distributed over many systems.

One method of scalability is using independent systems that redirect incoming requests for new sessions on a load average estimation (see figure 5). In this case only new sessions may be redirected to balance the system load, allowing the application to store the session context privately at the cost of temporary unbalanced system loads.

Other methods involve an additional system to distribute the requests over different systems or an extended DNS service with round robin mechanism to distribute the requests. Another method of request distribution is to extend the application with a mechanism, that forwards unwanted requests to other hosts.

4.4 Security

In the framework security is supported on three levels:

- **User-level**: In the frameworks core, an abstract user model is provided to support a large variety of authentication mechanisms.
- **System-level**: Because of the encapsulation of the external communication, the most common secure network protocols can be integrated in the system.
- **Data-level**: Encryption of secret data can be done by using the operating systems security (encrypted file system) and crypto facilities in the database. If those are not supported, security can be achieved by passing the data through an encrypting stream.

5 Conclusions

WebComposition is a component based implementation model which supports the development of web applications completely. In the WebComposition model services are logical units (e.g. business processes, layouts, etc.) which are implemented as groups of components. Because of this fine grained object-oriented model, services can be used easily to build up new services or applications. Additionally, the model considers the evolution of web application explicitly.

In this paper, we presented an architecture for the implementation of the WebComposition model. We introduced an object-oriented implementation scheme for services, we described the configuration and composition of services on the implementation level, and showed different implementation techniques for service definition by composition. Our approach considers security, performance and scalability questions. Furthermore, we outlined tools that will support the development of web applications. The result is a flexible, platform independent architecture that allows for the easy implementation of web applications the performance of which is cared for by scalability. The evolution of web applications is supported through the design of the framework that in fact implements the evolution bus of the WebComposition model.

The presented framework architecture allows the usage of abstract Web Engineering concepts for the development of concrete web projects. The implementation of the architecture brings the utilization of WebComposition concepts for development to reality.

We are implementing the websymphony framework in C++ at the moment. Our first platform is Windows NT with the component system COM/DCOM.

Literature


[Microsoft97] Microsoft, Using Active Server Pages with Microsoft IIS 3.0, Redmond, WA, USA, 1997

[NetObjects00] NetObjects, NetObjects Homepage, http://www.netobjects.com (06-25-00)


[UML00] UML 1.3 Documentation, 2000