EVOLVING WEB APPLICATIONS THROUGH COMPONENT BASED DOMAIN ENGINEERING

GUNTRAM GRAEF, MARTIN GAEDKE

Telecooperation Office (TecO), University of Karlsruhe, Vincenz-Prießnitz Str. 1, 76131 Karlsruhe, Germany, Tel.: +49 (721) 6902-89, Fax: -16, E-Mail: {graef|gaedke}@teco.uni-karlsruhe.de, URL: http://www.teco.uni-karlsruhe.de

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 even more 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, but such an approach requires dedicated support. The architecture suggested in this contribution addresses this issue by supporting the component-based development and evolution of Web applications. Furthermore by introducing domain specific markup languages for the description of Web-services a mechanism is introduced that makes it possible to abstract from many technical aspects related to the Web as an application platform. Thus application functionality can be described by domain experts with limited knowledge of technical details and the evolution of orthogonal aspects of the application can be decoupled. The whole approach is based on XML-technology to achieve the necessary flexibility and economic efficiency for the use in real world projects by using widely available standard tools.

KEYWORDS: Web Engineering, WebComposition, Domain Component, Domain Engineering, Evolution

INTRODUCTION

By offering ubiquitous access to any kind of information and applications the World Wide Web (Web) was able to establish itself as the dominant platform for the delivery of hypermedia applications. Under the influence of increasing competition, especially in the area of electronic commerce, these applications are subject to constant change concerning their functionality, their application interfaces or the information they offer (Cusumano/Yoffie 1999). These applications, that we will also refer to as Web-applications, can be strongly characterized by their underlying implementation model. Due to the rapid speed of technological innovation common to the Web the life-cycles of Web-applications become very short. The applications have to undergo an evolutionary process that never stops. Nevertheless, in the large majority of cases no disciplined approach is employed to address the increasing complexity of Web applications (Barta/Schranz 1998; Gaedke 1998).

The insight that the development and evolution of Web applications requires a dedicated
support through models, methods and principles of software engineering, comparable to that employed during the development of traditional applications, seems to suggest that Web application development should be based on a solid foundation of software engineering methodology. From a software engineering point of view the World Wide Web with its unique character and special properties is a new application domain (Gaedke 1998). This new discipline that during the last two years has established itself as *Web Engineering* offers both a cost reduction and an increase in quality during the development and evolution of Web-applications:

WEB ENGINEERING – The application of systematic, disciplined and quantifiable approaches to the cost-effective development and evolution of high-quality applications on the World Wide Web.

Web Engineering implicitly considers Berners-Lee's central demand for heterogenity of the system and autonomous administration of its resources (Berners-Lee 1990). This demand that we will refer to as the basic principles of the Web, is a major obstacle for current approaches to the development and maintenance of Web-applications, which will become obvious in section 2 of this contribution. Furthermore, a fine-grained and reuse-oriented implementation is necessary to allow for the federation of existing Web-applications or application parts into new applications (Gaedke/Turowski 1999).

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. This is also a prerequisite to be able to fully take advantage of applying modern reuse oriented software engineering processes to Web-technology.

In the following sections of this contribution we will present a framework in which the evolution of a Web-application can take place. In the third section we will describe the WebComposition Markup Language that can be used to describe Web-applications as components. We will introduce the concept of domain specific markup languages in the fourth section and describe how it can be used to further facilitate the development and evolution of functionality for the Web. A commercial application that is based on the described architecture will briefly we described in the fifth section. The last section contains a conclusion and an outlook on further research.

**EVOLUTION FRAMEWORK**

The requirements for a software system change as time goes by. It is obvious that many kinds of influences are responsible for this, such as new regulations, changes in corporate identity or an extension of functionality. Such maintenance tasks are difficult to handle if the application has not been designed with the possibility of later changes and extensions in mind (Gaedke/Turowski 1999).

To allow for a disciplined and manageable evolution of a Web-application in the future it makes sense not to design the initial application on the basis of the concrete requirements identified at the start of the project. Instead the initial application should be regarded as an empty application that is suitable for accommodating functionality within a clearly defined evolution space.
This approach is based on domain engineering which has been described as a process for creating a competence in application engineering for a family of similar systems (Institute 2000). During an analysis phase the properties of an application domain are determined. During a design phase this information is transformed into a model for the application domain. From this the required evolution space can be determined and, during the implementation phase of the domain engineering process, the initial application can be constructed as a framework ready to accommodate any kind of functionality that lies within the evolution space of the domain.

This view can be extended to several application domains. Therefore the term basic evolution bus has been introduced in (Gaedeke/Graef 2000) to describe the basic architecture of a Web-application. The evolution bus is the initial application for all abstract application domains of a Web-application.

![Diagram of Web-application evolution bus](image)

**FIGURE 1**: Dimensions of a Web-application's evolution space

It enables the management and collaboration of domain-components, i.e. components that implement specific application domains such as Web-based procurement, reporting or user driven data exchange. These initial domain-components also represent prototypes for future domain-components of the same application domain. The evolution can take place in two clearly defined ways (figure 7):

- **Domain specific evolution** – The extension of a domain through new domain-components, e.g. by prototyping an existing domain-component. Another possibility is that the domain itself changes or that it receives more functionality, which requires the modification of the domain's initial component that serves as a prototype for other domain components.

- **Evolution of the domain set** – The evolution of an application is also possible through the modification of the domain set. The extension of an application's functionality by adding a new application domain takes place e.g. when a shopping basket and corresponding functionality is added to a Web-based product catalog. The integration of a new domain is realized by connecting a new initial domain compo-
A framework for implementing the evolution bus and the domain components can be developed with the WebComposition Markup Language that will be introduced in the following section.

WEBCOMPOSITION MARKUP LANGUAGE (WCML)

THE WEBCOMPOSITION COMPONENT MODEL

The WebComposition Component Model, introduced in (Gaede 1999), extends the implementation model of the Web to circumvent its semantic limitations for the modelling of applications. It enables the development of components from which Web-applications can be composed. Such components are a code-abstraction of an arbitrary target language. Via a mapping mechanism from the component model to the Web implementation model it is possible to map components to Web-resources that contain e.g. HTML, WML (Wireless Markup Language) or script-code. In contrast to Web-resources WebComposition components can encapsulate design artifacts of arbitrary granularity. This can range from simple HTML element properties such as the used font-type to complex business processes or the implementation of design patterns. It is also possible to create a new component as a composition of existing components.

The WebComposition Component Model offers object-oriented semantics that can be employed to define relations between components such as inheritance, aggregation and polymorphism. In contrast to class based languages such as Java or C++ WebComposition uses prototype instance inheritance as described in (Ungar/Smith 1987). Hereby new variants can be created from prototypes that can be modified by overloading individual properties. During the description of a new component each existing component of the model can be referred to as a prototype. Thus in principle all parts of an application can be reused.

The WebComposition Component Model addresses the problems related to reuse on the Web and the mapping of fine-grained design entities to a document based implementation model by supporting the implementation of a Web-application within a fine-grained component model.

The evolution of Web-applications takes place through the manipulation or the addition of components that are kept persistent and accessible in a component store during their whole life-cycle.

THE WEBCOMPOSITION MARKUP LANGUAGE

The WebComposition approach suggests the WebComposition Markup Language (WCML) to support the development of components. This language has already been introduced in (Gaede 1999) and will thus only be briefly described here. WCML is an application of the eXtensible Markup Language (XML) and therefore inherits some of its advantages. The syntax of WCML documents can easily be checked against a Document Type Definition (DTD). WCML is platform independent and easy to process. The development of tools is strongly facilitated through the availability of XML-parsers and libraries in the public domain. Furthermore a variety of existing tools can be used such as XML-editors. For a more detailed language description and further information the
Components can be stored in a Component Store. This can be a database, but file-systems or Web-servers can also serve as Component Stores. Via a URI-addressing scheme several Component Stores can be integrated into Virtual Component Stores to allow for component reuse beyond a single Component Store. With the help of Web-servers and Virtual Component Stores, components can be directly reused world-wide via the Internet.

In (Gaedke 1999) a compiler has been introduced that performs a mapping of WCML-components to Web-resources, as visualized in FIGURE 2. The compiler uses a freely available XML parser component and employs it to read WCML components from a Virtual Component Store, to create a parse-tree and to perform a syntax check against a WCML DTD. Then the compiler resolves all references and inheritance relationships between WCML components. Finally the presentation methods of the components are executed and the required target resources are generated. This completes the mapping to the implementation model of the Web (or another target platform).

**FIGURE 2: Use of the WCML-Compiler**

**DESCRIPTION OF SERVICES WITH WCML**

Considering the business applications accessible via Web-technology one can state that in general applications offer a set of services to a user in order to support him during various tasks. Examples are the retrieval of information, the ordering of products or the issuing of a complaint. In a broader context via each service an enterprise offers a product to its customers that is not necessarily limited to the scope of the Web-application system. It could also involve existing business applications or the whole enterprise.

Therefore in (Gaedke/Turowski 1999) the concept of services as entities for modelling Web-applications has been described in more detail. In that contribution a service is described as consisting of several components. First, a service contains information that an
organization associates with it. This content is usually available in a media representation such as text or graphics. Second, a service contains layout and navigation that are used to make the service content available. It also contains directives describing the user-interaction and the consequences triggered by user-actions. Another important part of a service is the implementation of the business process i.e. the technical description of the interaction with the business application systems such as *Enterprise Resource Planning Systems*.

Technically a service can be represented as an object with an inner state that can be manipulated by the user. The audiovisual representation during run-time and the user-interface of a service are determined by Web-pages. The definition of the process is contained in process components in the back-end. Data, presentation and process are separated. Thus it is e.g. possible to accommodate different human-machine or machine-machine interaction modalities by simply exchanging the presentation. In (Gaedke/Turowski 1999) it has been described how the most important aspects of a service can be encapsulated within several WCML components:

- **Service Content**: A WCML component describing the properties and information content of a Service (e.g. textual content).

- **Service Layout and Navigation**: WCML components defining with which layout a service should be displayed and how the information should be made available to the user. This includes how navigation between pages should take place. An example would be a component that implements the Web design pattern *decorator* to display the service according to the corporate identity of an organization. Furthermore the Web design pattern *guided tour* may be used to provide a guided navigation through the pages of a service.

- **Service User-Interaction**: A WCML component that controls the interaction process between a user and the service. This includes the description of the user-interface and its integration with the service content.

- **Service Processing**: A sequence of calls to components to perform further processing in business application systems. These calls reflect the automated part of the business process. They use a middleware layer within the application framework to access (legacy) application systems.

A service can be composed from these components and therefore in itself represents a component. A service can also serve as a prototype. This makes a service reusable.

**SERVICE EXAMPLE: ORDERING MOBILE PHONES**

The following example clarifies the definition of services with WCML. A service-component shall offer a product (in this case a mobile phone) for sale. The service must describe the phone (content), it must display the information in a browser (layout and navigation) and it must define the ordering process for the telephone (process).

Each WCML component can be identified via a unique name (**UUID**). The description of a service for selling mobile phones (**SellMobilePhone**) is based on a component for the description of standard products. The component **StandardProduct** (figure 3) defines the *properties* common to all products. A property can be an attribute or a parameterless function of the component.
Now this component can be used as a prototype by the content component of the Sell-MobilePhone service to add all data relevant for the mobile phone (figure 4).

The component CellphoneModel123 assumes the properties of the prototype through inheritance from StandardProduct. With the inherited properties e.g. the currency of the provided price of 229.99 is now clearly defined as in Euro (EUR).

The component UserInteraction234 describes the user-interaction. The code extract in figure 6 shows a simple user-interaction for the input of the quantity of a product order. The example makes use of ordered multiple-inheritance in WCML to access several prototype components at once.

The presentation components and process components are also defined in WCML.
components are now composed to a service by a service component (figure 6).

...<COMPONENT uuid='SellMobilePhone'>
  <PROTOTYPE is='CellphoneModel123'/>
  <PROTOTYPE is='UserInteraction123'/>
  <PROTOTYPE is='Layout'/>
  <PROTOTYPE is='Navigation'/>
  <PROTOTYPE is='OrderProcess'/>
</COMPONENT>...

FIGURE 6: Service component

By exchanging the content component CellphoneModel123 the same service could be used for selling other types of mobile phones. By changing the process component the underlying business process could be changed, e.g. to offer the phone in a bundling action. By changing the layout component or the navigation component the presentation of the service can be modified.

The processing of the business data that results from use of the various services takes place within the business application systems of the organization. These can be heterogeneous environments with several legacy systems. Within the application framework each system is represented by an agent. The communication with these applications must take place via a simple and standardized distributed mechanism to make the services built above this mechanism reusable. For the business process a business-to-business communication between the applications of several organizations might be necessary.

A suitable approach based on widely accepted communication standards has already been introduced in (Gaedke/Turowski 1999).

DOMAIN SPECIFIC MARKUP LANGUAGES

To further facilitate the domain specific evolution of an application we are going to introduce the concept of Service Domain Markup Languages (SDMLs). SDMLs are domain specific languages based on XML. They are used to standardize domain components and to support the interaction between domain components and the evolution bus.

An SDML abstracts from those properties of a service common to all services of a service domain. It serves as a tool to easily specify individual services in the context of a service domain. The semantics and complexity of such a language corresponds exactly to the decision space available to a service developer working within a service domain.

Figure 7 shows an extract from the Document Type Definition of a very simple SDML for the domain "product ordering" in the context of an existing E-Commerce system. The labeling of the language elements (tags) has been based directly on the vocabulary used by experts in the corresponding domain. The language enables a domain expert without advanced programming knowledge to describe all aspects of a specific product ordering service. A service developer can describe the structure and content of an order
via an order object (ORDER) while he can define the user-interaction in the SERVICE_FORM part of the SDML document tree. With the MESSAGE Tag the developer can define a confirmation message displayed after the order has been accepted.

Figure 8 shows an extract of a service description for a concrete product ordering service for mobile phones. In the first part general properties of the service and the associated order such as service name, standard price and value added tax rate are defined. Then various order lines are described within the ORDER_LINE tag. The code example shows an order line for D1-telephone cards that can be included as accessories with a mobile phone order. Inside the SERVICE_FORM tag the various user-interface elements are defined in the same order as they will be displayed later. The example code given describes a text input field whose content is linked to the quantity-property of the order.
In this example the service developer can modify service content (ORDER section) and
user-interaction (SERVICE_FORM section). Layout, navigation, certain aspects of
content and user-interaction and the processing are the same for all services of this
service domain and are therefore not part of the SDML. They are encapsulated within
WCML-components of the framework and are maintained by experts such as Web-
designers and Web-engineers. In other service domains the decision space might be
quite different though. The SDML for another domain may e.g. contain processing
statements or layout properties.
SERVICE FACTORY

The transformation of a service description into a functioning service is done by a Service Factory. The concept of a service factory is based on the factory design pattern by (Gamma/Helm/Johnson/Vlissides 1995) and has been introduced in (Gaedke/Turowski 1999). In our case a mapping is performed from an SDML to a service component in WCML. That generated service component usually is tightly integrated with existing framework components via inheritance and composition operators. The mapping rules for a single SDML are encapsulated in a factory method that is selected and invoked from a control function. Because both SDMLs as well as WCML are based on XML such a factory method is easily implemented with either the eXtensible Stylesheet Language (XSL) (Graef/Gaedke 1999) or by using special WCML language constructs for the definition of factory methods (Gaedke/Segor 2000). In both cases using XML as the underlying technology significantly contributes to the efficiency and flexibility of the described approach.

![Diagram of System Architecture](image)

**FIGURE 9: System architecture**

DOMAIN SPECIFIC EVOLUTION

Domain specific evolution takes place by the description of new services with an SDML and by the extension of an SDML itself together with the underlying WCML component framework. An example would be the extension of the previously described SDML for order services by a multi-lingual order confirmation. The SDML could be extended by various sub tags for the MESSAGE tag and the framework component for the display of
order confirmations could be extended with additional functionality such as the capabil-
ity of storing versions for several language or a mechanism to select and display a lan-
guage version according to user preferences. The factory method for the SDML would
have to be extended by XSL mapping templates for the new XML tags added to the
SDML.

![Diagram](image)

**FIGURE 10: Internal Architecture of the Service Factory**

**EVOLUTION OF THE DOMAIN SET**

The domain set can be extended by developing a new application domain. This is done
by defining a new SDML and by extending the WCML component framework by add-
ing a new service prototype component. To enable the mapping of SDML service de-
scriptions to WCML components an XSL stylesheet that implements a factory method
for the Service Factory has to be added.

**TOOLS**

Using XML based markup languages for describing services facilitates the development
of special development tools. Various tools and libraries for dealing with XML docu-
ments are publicly available and XML itself provides a clearly defined structure that
forms the basis for easy manipulation of documents. Figure 11 shows an editor that can
be used to visually program order services. It is based on the SDML introduced in the
previous example. The editor maps the SDML bijectively to the graphical i/o-elements
of the user-interface. It has been developed in Java using an XML parser component that
is freely available. Such an editor provides a domain expert with a tool that allows her to
develop simple services without sophisticated programming knowledge. A service de-
veloped with such a tool can also be used as a starting point for a Web-engineer who
may use the generated WCML components and extend them to develop a new service
leaving the boundaries of the current domain.

Figure 9 shows how a service editor and a Service Factory work together with the sys-
tem components introduced before in section 3. The exchange of design artifacts be-
tween the various instances takes place indirectly via the Virtual Component Store that
stores WCML components and a repository for SDML service descriptions.
APPLICATION OF THE DESCRIBED ARCHITECTURE

In a joint project between the Telecooperation Office (TecO) at the University of Karlsruhe and Hewlett-Packard (HP) a service-oriented e-commerce application, Eurovictor, has been developed based on the WebComposition Component Model and domain-specific markup languages. The aim of the project was to develop a support system that enables development, management, maintenance and evolution of services within the heterogeneous environment of the European intranet of the HP company. An evolution bus was realized as part of the project. The evolution of the application took place through the integration of several domain specific services such as presentation of information, software orders or product purchase. These services serve as prototypes for the domain specific evolution of the application.

Figure 12 shows the Web-application's start page. The left part contains a menu that allows for a selection of services. In the middle and on the right side two special services can be found: A service for the adaptation of the application to the behavior of the current user and another service offering shopping basket functionality. The adaptation service demonstrates the flexibility the application gained due to the domain-component specific architecture it is based on.

A large part of the domain specific evolution of the application has been triggered by its international scope. As an example new services are constructed through inheritance that are adapted to national layouts, languages or legal regulations. Meanwhile in almost all European countries services are developed in a distributed and decentralized manner and made available via the Eurovictor system.
CONCLUSION

The Web has been established as a major platform for applications, although the underlying implementation model complicates the development and evolution of Web-applications. To manage the evolution of complex Web-applications during their complete life-cycle it is necessary to use a development model that allows us to compose applications from reusable components of moderate complexity and arbitrary granularity. The WebComposition Component Model with the language WCML is suitable for that purpose.

We further facilitate the development and evolution of Web-applications by introducing domain specific markup languages for Web-Services and the related mechanism of a Service Factory. Services are based on initial domain components that each serve as a prototype for a service domain. The prototypes of all service domains are based on an evolution bus as a basic framework. A Web-application can evolve within this evolution bus in two different ways. First, an evolution can take place within a service domain by modifying the initial service component or by adding a new service component to the component set of a domain - usually by proto-typing. Second, the domain set itself can evolve by adding new initial domain components and connecting them to the evolution bus. Different aspects of a Web-application can be separated and assigned to people with different roles working on different levels of abstraction. This ranges from the visual programming of a simple service by a domain expert to the modification of an initial domain component by a Web-engineer.

The approach described in this contribution has been successfully applied to a large and internationally distributed Web-application at Hewlett-Packard.
EXAMPLES

The WCML-Compiler and code examples are publicly available for download and testing from: http://www.webengineering.org.

LITERATURE


