EVOLUTION OF WEB-BASED APPLICATIONS USING DOMAIN-SPECIFIC MARKUP LANGUAGES

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.edu, URL: http://www.teco.edu -or- http://webengineering.org

ABSTRACT

The lifecycle of Web-based applications is characterized by frequent changes to content, user interface, and functionality. Updating content, improving the services provided to users, drives further development of a Web-based application. The major goal for the success of a Web-based application becomes therefore its evolution. Though, development and maintenance of Web-based applications suffers from the underlying document-based implementation model. A disciplined evolution of Web-based applications requires the application of software engineering practice for systematic further development and reuse of software artifacts. In this contribution we suggest to adopt the component paradigm to development and evolution of Web-based applications. The approach is based on a dedicated component-technology and component-software architecture. It allows abstracting from many technical aspects related to the Web as an application platform by introducing domain specific markup languages. These languages allow the description of services, which represent domain-components in our Web-component-software approach. Domain experts with limited knowledge of technical details can therefore describe application functionality and the evolution of orthogonal aspects of the application can be de-coupled. The whole approach is based on XML to achieve the necessary standardization and economic efficiency for the use in real world projects.

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 lifecycles 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 application (Barta & Schranz 1998, Gellersen & Gaedke 1999).

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 is a new application domain (Gaedke et al 1999a). This new discipline that during the last three 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 (Gaedke & Rehse 2000).

Web Engineering implicitly considers Berners-Lee's central demand for *heterogenity* of the system and *autonomous administration* of its resource (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 2000).

The positive experiences with component-based software development and its advantages (McLure 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 be described in the fifth section. Te 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 (SEI 1999). 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 (Gadeke & 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.

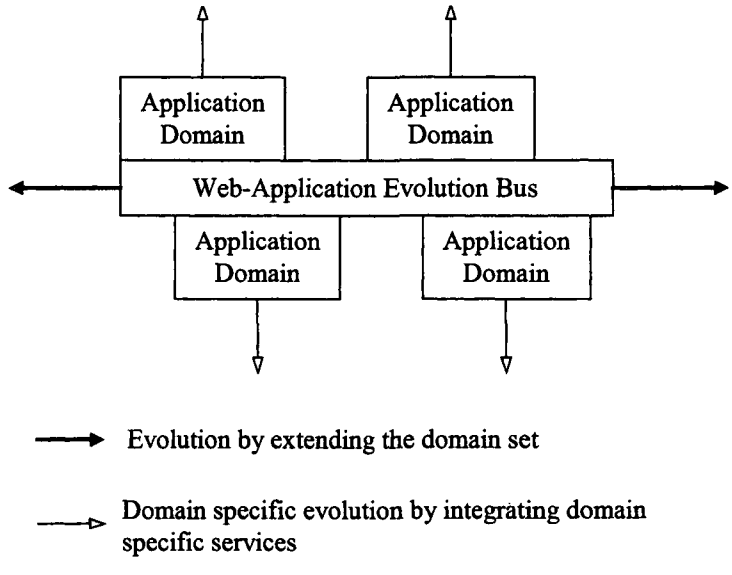

Figure 1: Dimensions of a Web-application's evolution space

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 catalogue. The integration of a new domain is realized by connecting a new initial domain component to the evolution bus.

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–APPROACH FOR COMPONENT-BASED WEB ENGINEERING

WebComposition Component Model

The abstract WebComposition Component Model, introduced in (Gellersen et al 1997), extends the implementation model of the Web to circumvent its semantic limitations for the modeling 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 artefacts 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.
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 lifecycle.

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 (Gaedke et al 1999b) 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 reader may refer to (Gaedke 1999).
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 an 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 worldwide via the Internet.
In (Gaedke et al 2000) 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).

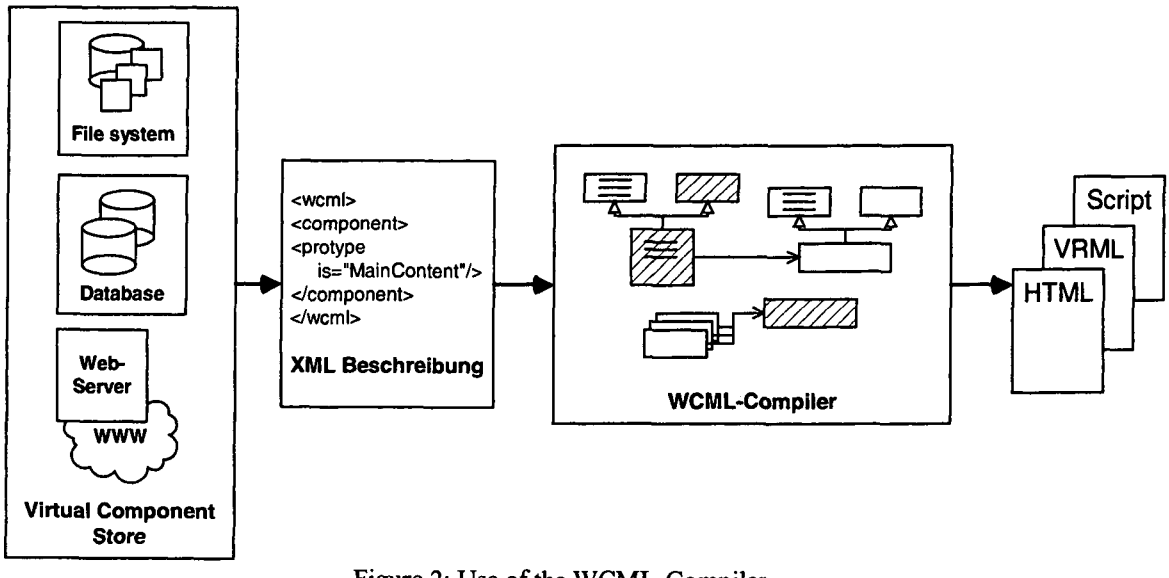

Figure 2: Use of the WCML-Compiler

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 et al 1999a) 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 2000) 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 controls the access to the service content by the user-interface.
- *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 by inheritance.

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.

```
<?xml version='1.0' ?>
<!DOCTYPE wcml SYSTEM "wcml2.dtd">
<WCML>
<COMPONENT uuid='StandardProduct'>
        <PROPERTY name='currency'>EUR</PROPERTY>
        <PROPERTY name='unitName'>piece</PROPERTY>
        <PROPERTY name='unitNamePlural'>pieces</PROPERTY>
        <PROPERTY name='quantity'>1</PROPERTY>
        ...
</COMPONENT>
...
```

Figure 3: Content component for products

Now this component can be used as a prototype by the content component of the SellMobilePhone service to add all data relevant for the mobile phone (Figure 4).

```
...
<COMPONENT uuid='CellphoneModel123'>
        <PROTOTYPE is='StandardProduct'/>
        <PROPERTY name='bandInfo'>dual</PROPERTY>
        <PROPERTY name='price'>229.99</PROPERTY>
        ...
</COMPONENT>
...
```

Figure 4: Content component for a certain product

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.

```
...
<COMPONENT uuid='UserInteraction234'>
        <PROTOTYPE is='UserInteraction'/>
        <PROTOTYPE is='NumericInputField'/>
        <PROPERTY name='datafield'>quantity</PROPERTY>
        <PROPERTY name='minvalue'>1</PROPERTY>
        <PROPERTY name='maxvalue'>99</PROPERTY>
        ...
</COMPONENT>
```

Figure 5: User-interaction component

The presentation components and process components are also defined in WCML. All 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>
...
</WCML>
```

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 2000).

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 labelling 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.

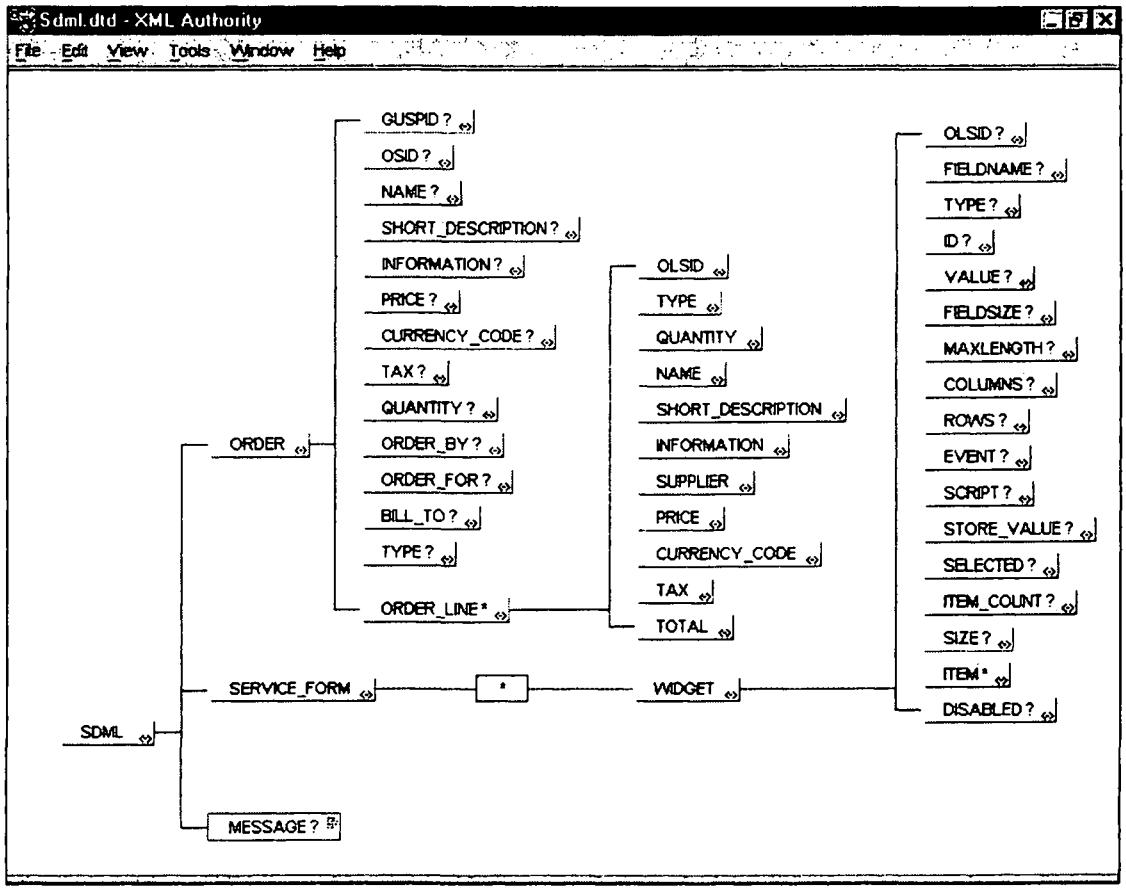

Figure 7: Document Type Definition of an SDML

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.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?xml:stylesheet type="text/xsl" href="ServiceCompiler.xsl" ?>
<!DOCTYPE SDML SYSTEM "orderservice_sdml.dtd">
<SDML>
<ORDER>
        <GUSPID>                <STRING>HandyOrderService</STRING>          </GUSPID>
        <OSID>                  <STRING>HandyOSID</STRING>                  </OSID>
        <NAME>                  <HTML>Handy OrderService</HTML>             </NAME>
        <SHORT_DESCRIPTION>     <HTML>Service for ordering mobile phones</HTML></SHORT_DESCRIPTION>
        <INFORMATION>           <HTML>If you have any questions call 5214</HTML></INFORMATION>
        <PRICE>                 <NUMBER>199.00</NUMBER>                     </PRICE>
        <CURRENCY_CODE>         <STRING>EUR</STRING>                        </CURRENCY_CODE>
        <TAX>                   <NUMBER>16.00</NUMBER>                      </TAX>
        <QUANTITY>              <NUMBER>1.00</NUMBER>                       </QUANTITY>
        <ORDER_BY>              <VARIABLE>USERID</VARIABLE>                 </ORDER_BY>
        <ORDER_FOR>             <VARIABLE>USERID</VARIABLE>                 </ORDER_FOR>
        <BILL_TO>               <VARIABLE>USERID</VARIABLE>                 </BILL_TO>
        <TYPE>                  <STRING></STRING>                           </TYPE>

        <ORDER_LINE>
                <OLSID>                 <STRING>OLSID-D1</STRING>           </OLSID>
                <TYPE>                  <STRING></STRING>                   </TYPE>
                <QUANTITY>              <NUMBER>1</NUMBER>                  </QUANTITY>
                <NAME>                  <HTML>D1-CARD</HTML>                </NAME>
                <SHORT_DESCRIPTION>     <HTML>D1-Telephone Card</HTML>      </SHORT_DESCRIPTION>
                <INFORMATION>           <HTML></HTML>                       </INFORMATION>
                <SUPPLIER>              <STRING>Deutsche Telekom</STRING>   </SUPPLIER>
                <PRICE>                 <NUMBER>29.0</NUMBER>               </PRICE>
                <CURRENCY_CODE>         <STRING>EUR</STRING>                </CURRENCY_CODE>
                <TAX>                   <NUMBER>16.0</NUMBER>               </TAX>
                <TOTAL>                 <NUMBER>33.64</NUMBER>              </TOTAL>
        </ORDER_LINE>
        <ORDER_LINE>
                ...
        </ORDER_LINE>
</ORDER>
<SERVICE_FORM>
                <WIDGET>
                        <OLSID>         <STRING></STRING>                   </OLSID>
                        <FIELDNAME>     <STRING>Quantity</STRING>           </FIELDNAME>
                        <TYPE>          <STRING>textfield</STRING>          </TYPE>
                        <ID>            <STRING>HandyOSID.Quantity</STRING></ID>
                        <VALUE>         <STRING>1</STRING>                  </VALUE>
                        <FIELDSIZE>     <STRING>5</STRING>                  </FIELDSIZE>
                        <MAXLENGTH>     <STRING>4</STRING>                  </MAXLENGTH>
                </WIDGET>
                <WIDGET>
                        ...
                </WIDGET>
</SERVICE_FORM>

<MESSAGE>Danke für Ihre Bestellung!</MESSAGE>
</SDML>
```

Figure 8: SDML description of an order service

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

A Service Factory does the transformation of a service description into a functioning service. The concept of a service factory is based on the factory design pattern by (Gamma et al 1995) and has been introduced in (Gaedke et al 1999a). 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) [21] or by using special WCML language constructs for the definition of factory methods [22]. In both cases using XML as the underlying technology significantly contributes to the efficiency and flexibility of the described approach.

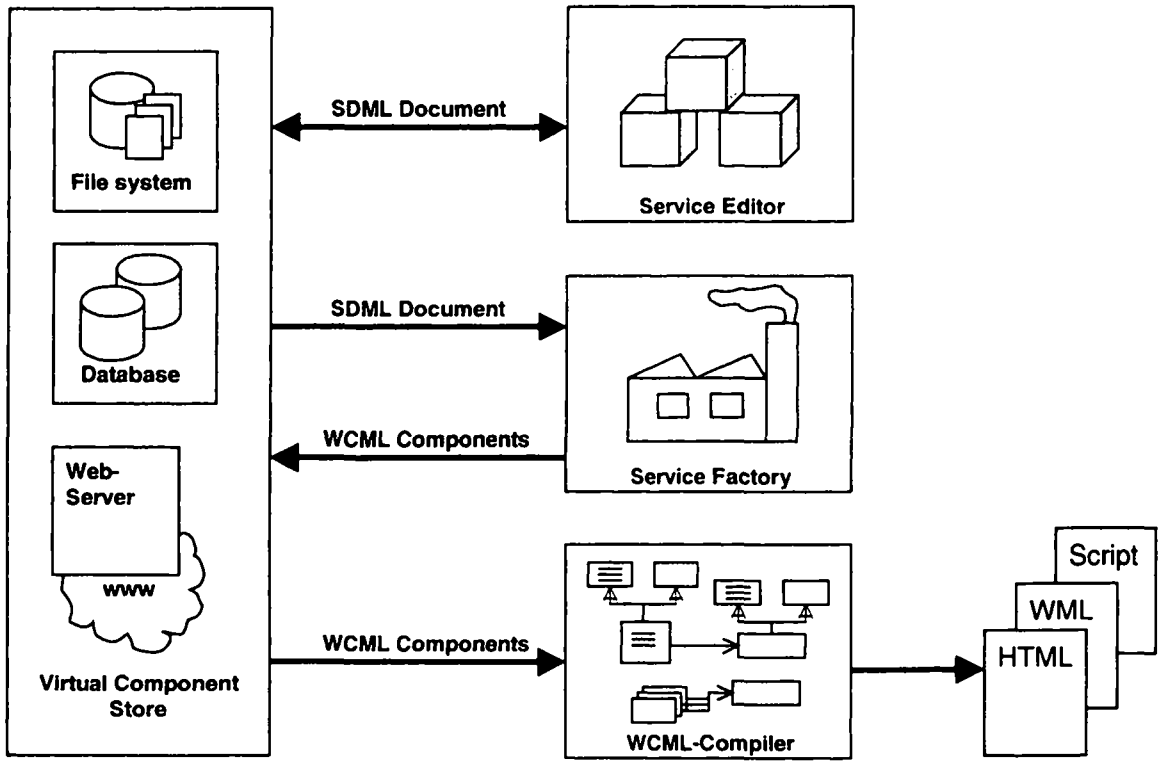

Figure 9: System architecture

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. Various sub tags could extend the SDML, e.g. the MESSAGE tag and the framework component for the visualization of order confirmations could be extended with multi-lingual support. The factory method for the SDML would have to be extended by XSL mapping templates for the new XML tags added to the SDML.

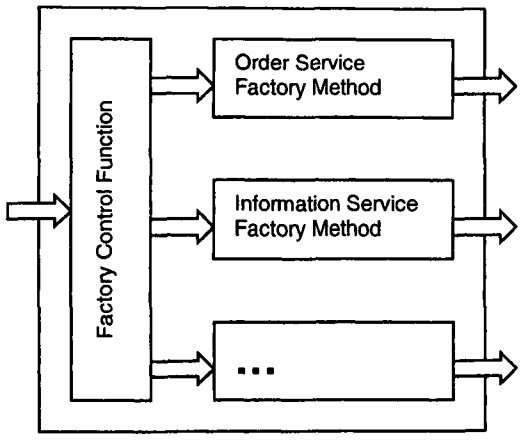

Figure 10: Internal Architecture of the Service Factory

Evolution of the Domain Set

Developing a new application domain can extend the domain set. This is done by defining a new SDML and by extending the WCML component framework by adding a new service prototype component. To enable the mapping of SDML service descriptions to WCML components an XSL style sheet 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 documents 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 developed 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 system components. The exchange of design artefacts between the various instances takes place indirectly via the Virtual Component Store that stores WCML components and a repository for SDML service descriptions.

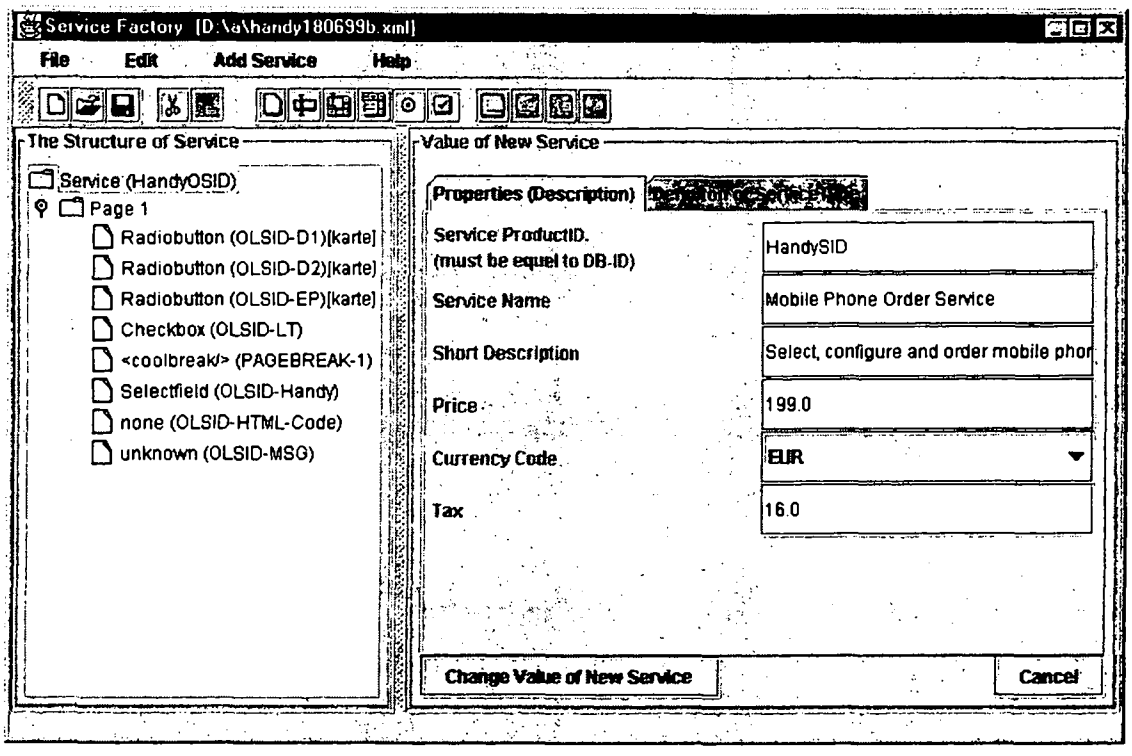

Figure 11: Visual editor for order services

APPLICATION OF THE APPROACH

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 HP. 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 behaviour 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.

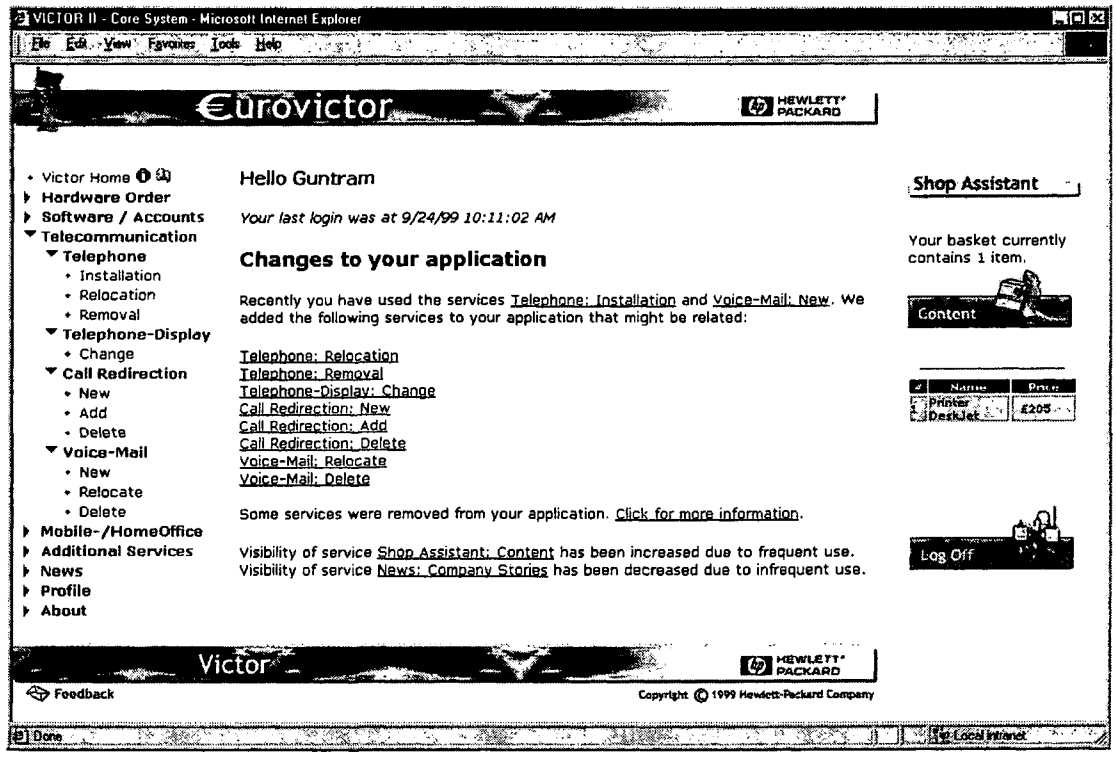

Figure 12: The Eurovictor system

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 lifecycle 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.

ACKNOWLEDGEMENTS

An earlier version of this paper was presented at the first XML conference (XML2000), Heidelberg, Germany, May 3-4, 2000. Comments from participants of this conference are gratefully acknowledged. The authors would further like to thank U. Stegemüller of the Hewlett-Packard Corporation, Germany for their cooperation, thoughtful feedback, and for providing inspiration for some of the ideas developed in the approach presented here.

EXAMPLES

WCML-Compiler and code examples are available at *http://webengineering.org*

REFERENCES

Barta R.A. and M. W. Schranz, "JESSICA: an object-oriented hypermedia publishing processor," **Computer Networks and ISDN Systems**, vol. 30(1998), pp. 239-249, 1998.

Berners-Lee T., "Information Management: A Proposal," , vol. 1998: **CERN**, 1990.

Cusumano M.A. and D. B. Yoffie, "Software Development on Internet Time," **IEEE Computer**, vol. 32, pp. 60-69, 1999.

Gaedke M., H.-W. Gellersen, A. Schmidt, U. Stegemüller, and W. Kurr, "Object-oriented Web Engineering for Large-scale Web Service Management," presented at **Thirty-Second Annual Hawaii International Conference On System Sciences (HICSS-32)**, Island of Maui, USA, 1999a.

Gaedke M. and J. Rehse, "Supporting Compositional Reuse in Component-Based Web Engineering," presented at 2000 **ACM Symposium on Applied Computing (SAC 2000)**, Villa Olmo, Como, Italy, 2000.

Gaedke M. and K. Turowski, "Integrating Web-based E-Commerce Applications with Business Application Systems," **Netnomics Journal**, Baltzer Science Publishers, vol. 2, pp. 117-138, 2000.

Gaedke M. and K. Turowski, "Framework for Maintaining Evolution of E-Commerce Applications in the Web," presented at 12th International Conference - Software and Systems Engineering and their Applications (ICSSEA '99), Paris, France, 1999.

Gaedke M. and G. Graef, "Development and Evolution of Web-Applications using the WebComposition Process Model," presented at **2nd Web Engineering Workshop at the 9th International World Wide Web Conference (WWW9-WebE)**, Amsterdam, The Netherlands, 2000.

Gaedke M., D. Schempf, and H.-W. Gellersen, "WCML: An enabling technology for the reuse in object-oriented Web Engineering," presented at Poster-Proceedings of **the 8th International World Wide Web Conference (WWW8)**, Toronto, Ontario, Canada, 1999.

Gaedke M., "WebEngineering.org Homepage," , vol. 2000, 1999.

Gaedke M., C. Segor, and H.-W. Gellersen, "WCML: Paving the Way for Reuse in Object-Oriented Web Engineering," presented at **2000 ACM Symposium on Applied Computing (SAC 2000)**, Villa Olmo, Como, Italy, 2000.

Gamma E., R. Helm, R. Johnson, and J. Vlissides, **Design patterns: elements of reusable object-oriented software**. Reading, Mass.: Addison-Wesley, 1995.

Gellersen H.W. and M. Gaedke, "Object-Oriented Web Application Development," **IEEE Internet Computing**, vol. 3, pp. 60-68, 1999.

H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebCompostion: an object-oriented support system for the Web engineering lifecycle," **Computer Networks and ISDN Systems**, vol. 29, pp. 1429-1437, 1997.

Graef G. and M. Gaedke, "An Evolution-oriented Architecture for Web Applications," presented at **Second Nordic Workshop on Software Architecture (NOSA'99)**, Ronneby, Sweden, 1999.

Lim W.C., **Managing software reuse : a comprehensive guide to strategically reengineering the organization for reusable components**. Upper Saddle River, NJ: Prentice Hall, 1998.

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

Tracz W, **Confessions of a used program salesman : institutionalizing software reuse**. Reading, Mass.: Addison-Wesley Pub. Co., 1995.

Szyperski C., **Component software: beyond object-oriented programming**. Reading, Mass.: ACM Press; Addison-Wesley, 1997.

SEI, "Domain Engineering," , vol. 1999: Software Engineering Institute, Carnegie Mellon University, 1999.

Ungar D. and R. B. Smith, "Self: The Power of Simplicity," presented at **OOPSLA '87**, 1987.

Segor C. and M. Gaedke, "Crossing the Gap - From Design to Implementation in Web-Application Development," presented at **Information Resources Management Association International Conference**, Anchorage, Alaska, USA, 2000.