OBJECT-ORIENTED WEB APPLICATION DEVELOPMENT

Most Web applications are still developed ad hoc. One reason is the gap between established software design concepts and the low-level Web implementation model.

HANS-W. GELLERSEN AND MARTIN GAEDKE
University of Karlsruhe

The Web has evolved into a global environment for delivering all kinds of applications, ranging from small-scale and short-lived services to large-scale, enterprise workflow systems distributed over many servers. Applications that use HTML-based front ends benefit from the pervasive distribution of Web browsers for universal, cross-platform access. Another striking advantage of Web delivery lies in the concept of thin clients and centralized maintenance, facilitating instantaneous deployment of software updates at minimal cost. While the popularity of the Web and its advantages as a client-server platform have led to countless HTML-based applications, the development of Web applications is still mostly ad hoc. There is no rigorous, systematic approach, and most current Web application development and management practices rely on the knowledge and experience of individual developers.

One reason for the lack of a structured approach may be in the Web's legacy as an information medium rather than an application platform. Web development is seen primarily as an authoring problem rather than a software development problem to which well-established software engineering principles should apply.

We believe another reason is that the Web implementation model does not relate well to state-of-the-art software development models. Web implementation is based on low-level technologies that do not provide high-level abstractions for sharing and reuse. The lack of suitable abstractions makes it difficult to construct frameworks that capture architectural design decisions for reuse in different parts of an application or in different application projects. The lack of abstraction also makes it difficult to maintain and evolve Web-based applications. Design decisions are very hard to track in a low-level implementation; as the application evolves, changes can easily lead to inconsistencies. Because Web-based applications tend to evolve quickly, with frequent updates and redesigns, poor maintainability is a critical problem.

1089-7801/99/$10.00 ©1999 IEEE

In this article we summarize work reported earlier on WebComposition,[1] a model for Web application development, then introduce the WebComposition Markup Language—an XML-based language that implements the model. WCML embodies object-oriented principles such as modularity, abstraction, and encapsulation. It facilitates the description of higher level concepts and frameworks for reuse that can bridge the gap we currently perceive in Web application development between high-level design and low-level implementation.

We begin with a discussion of why the basic Web implementation model does not work as a development model for Web applications. Related work is presented in the sidebar, "Structured Web Application Development," on page 62.

WEB APPLICATION DEVELOPMENT PROCESS

We define a Web application as any software application that depends on the Web for its correct execution. Obviously, software explicitly designed for delivery over the Web falls under this definition—for example, Web sites or Web-based journals. These kinds of software are characterized by a strong notion of content.

We also include software that uses the Web infrastructure for its execution. For example, many information systems that were designed and built prior to the Web are now made available as Web applications through the use of browsers. Beyond legacy information systems, there is also software that is less typical for delivery over the Web but still dependent on it as the client-server platform for its execution. Two recently reported examples addressed the coordination of distributed applications[2] and remote monitoring.[3]

General Software Development

Conventional software processes usually address four development phases: analysis, design, implementation, and maintenance/evolution.

During analysis, developers build a model of an application in terms of the domain. Ideally, this analysis concentrates on the application's *problem space*—separate from software considerations, which are part of the *solution space*. Accordingly, this phase should not be affected by whether or not the application is to be delivered over the Web.

Based on the analysis, a model of the software solution is defined during the design phase. Obviously, the Web application development process—in contrast to a general software process—assumes a Web-based software solution and must accommodate this.

The implementation transforms the design into actual software. For Web applications, this phase has to rely on available Web implementation technology.

Finally, maintenance is concerned with modifications to the software, which may occur at the implementation, design, or even analysis levels. General software processes are hard to relate to the Web implementation model, as will be discussed below. The Web implementation model is based on flat decomposition of applications into resources. Resources have a unique address, and they are delivered on request from Web server to Web client. They can be static or dynamically generated from a script, but they are inherently specific, which means that they cannot capture abstractions.

Document Delivery Applications

The World Wide Web was originally designed as an information medium for distributed research teams.[4] A key objective was to make it as easy as possible for authors to deliver documents, and the notion of Web application development essentially boiled down to document development by an author or small group of authors. For this kind of development, the life cycle comprises informal analysis of what is to be presented, informal design of how to structure it into hyperlinked chunks of information, and implementation through markup. Following implementation, the documents are maintained by the authors themselves.

The Web implementation model was designed to meet these life-cycle requirements. It is deliberately simple, based on the notion of *resources* that model mostly self-contained chunks of information. Resources are authored and maintained rather independently of other resources, and links are the means by which resources can be combined into coherent sets of documents—for example, a Web site.

The resource concept fits the document-development life cycle very well, and resources support the principles of modularity in this context. However, the use of the Web has moved far beyond its original scope, even in document delivery. It has become a tool for high-end publishing with complex requirements related to layout, corporate identity, and the integrity of large webs of information. The life cycle of a company Web site is now typically based on intensive requirements analysis in terms of content, structure, access, and corporate identity. The design must decompose these requirements into resources and hyperlinks, and also address layout. Maintenance must focus on the evolution of content, but also on site integrity.

The life cycle is no longer author-centric. Requirements analysis involves, for example, information analysts as well as marketing people and other stakeholders. Design addresses both database development and graphic presentation. Implementation requires Web programmers to use features beyond simple markup, and maintenance involves site managers and Webmasters.

The resource notion underlying the Web implementation model does not meet the requirements of this kind of life cycle. Most notably, there is no separation of concerns regarding content and layout. Further, the Web implementation model does not provide abstractions to capture structural design for reuse, even though layout and navigation structures are commonly reused in different parts of a site.

Life Cycle of General Applications Delivered over the Web

Nor does the resource model address the life cycle of general applications delivered over the Web. Decomposing applications into resources is not logical. The resource model requires separate concerns such as user interfaces, application logic, and—for example—database back-ends to be embedded in an intermingled way.

Because of the request-response style protocol between client and server, Web applications are structured, in effect, as finite state machines (FSM): The nodes correspond to resources and the transitions leading away from a node correspond to hyperlinks or form elements within the node. This distributes the application logic over a number of resources in chunks of script code, in document-embedded links, and in form elements. Content is mixed with application logic and typically embedded in script code that implements the application logic. Further, the user interface is declarative and specified as documents to be rendered in standard browsers. A Web application typically generates the user interface declaration dynamically to reflect application and interaction state.

In summary, the delivery of applications in the Web environment is radically different from the usual ways of delivering software, and imposes a completely different structure and approach on application development. Obviously, design and maintenance of such applications should be in

STRUCTURED WEB APPLICATION DEVELOPMENT

Several communities are addressing the need for more structured development of Web applications.

Commercial IDEs. There is a wide range of commercial products. Integrated development environments (IDEs), however, fall far short of covering the whole development process and are mostly geared toward ad hoc implementation based on tool-specific abstractions. Further, they are rather self-contained and thus difficult to integrate with other tools and methods in a development process. The models underlying IDEs do, in general, abstract from low-level technologies but not to the extent required to establish conceptual integrity from the design to the implementation and maintenance phases of development. For example, some IDEs address separation of general layout from content, but there are no general mechanisms for separation of concerns.

Hypermedia Process Models. In contrast to commercial contributions toward disciplined Web development, which are focused on products, the hypermedia community has proposed development models focused on processes. Two examples are OOHDM[1] and RMM.[2] The proposed methods address primarily analysis and design. The underlying models are powerful but geared toward the hypermedia application domain. For example, RMM is based on the notion of entities and relations, which suit information system development but not software applications modeling in general.

CASE tools based on hypermedia development models use automated code generation for mapping a design to a Web implementation; see, for instance, RMCase.[3] To ensure integrity throughout the life cycle, all maintenance/evolution activity must be carried out at design level, prohibiting access to implementation detail. This is a serious constraint considering the general drive of Web applications to take up the latest implementation technologies during evolution.

Object-Oriented Development. There is also work considering Web development from a more general software process perspective. We have described initial work on an object-oriented Web component model, WebComposition, and its role in the Web application life cycle.[4] The model (summarized in the main text) is an abstraction of the Web implementation model, terms of user interface, application logic, and database back end rather than resources. Current Web implementation technology is too low level to support a proper development process.

Case Study in Web Application Development

To illustrate some problems we've found to be typical of Web application development, we will describe a small part of a travel assistant system that we developed over the past year. The application integrates travel booking and routing systems for intermodal travel planning. The Web was chosen as the integration platform to capitalize on already existing travel information systems and on HTML browsers as a means of global system access.

One requirement was for the application to support system access from mobile handheld computers.[5] In principle, state-of-the-art handheld computers support HTML browsing, but the small displays pose problems in rendering HTML. A one-size-fits-all design is obviously not satisfactory for delivering Web-based user interfaces over both standard desktop browsers and handheld browsers.

Instead, Web-based user interfaces must be adapted to the different browser characteristics and delivered accordingly. The user interfaces obviously differ primarily in page layout, while the content—messages, form elements, and so on—remain the same. Thus, the user interface design includes a recurring pattern, based on the well-known decorator pattern.[6] Figure 1 illustrates this simple pattern: an HTML-based user interface screen is defined on an abstract level by a set of screen elements—basically the content of the screen. From this abstract screen, specific screens (that is, HTML pages) are derived for each target browser platform. Figure 2 applies this pattern to the design of the login screen for two platforms in the travel assistant system.

Because the Web implementation model does not support abstraction, it cannot capture general frame-

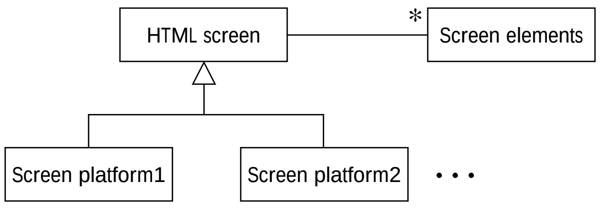

Figure 1. HTML user interface pattern for browser-adapted application delivery.

with each component encapsulating its mapping to a Web implementation in a dedicated method or service.

Similar work was presented by Barta and Schranz[5] and by Coda et al.[6] Barta and Schranz take a language-based approach for object-oriented description of Web applications. For analysis and design, they adopt RMM concepts, which reflects their focus on hypermedia information systems. In contrast, Coda et al. propose a general software process to bridge the gap between design and Web implementation. Their proposed object-oriented implementation technology, WOOM, is a generative model based on objects that model Web implementation primitives—in particular, HTML elements.

Like the objects in WebComposition, WOOM objects encapsulate the mapping to a Web implementation in a dedicated method.

REFERENCES WITH URLS

1. D. Schwabe, G. Rossi, and S. Barbosa, "Systematic Hypermedia Design with OOHDM," *Proc. ACM Int'l Conf. Hypertext*, ACM Press, New York, 1996, pp. 116-128; also available online at http://wwwx.cs.unc.edu/~barman/HT96/section1.html.
2. T. Isakowitz, E.A. Stohr, and P. Balasubramaninan, "RMM: A Methodology for Structured Hypermedia Design," *Comm. ACM*, Vol. 38, No. 8, Aug. 1995, pp. 34-44; also available online at http://rmm-java.stern.nyu.edu/rmm/papers.rmd.ps.
3. A. Diaz et al., "RMC: A Tool to Design WWW Applications," *World Wide Web J.*, Vol. 1, No. 1, Dec. 1995; available online at http://www.w3.org/pub/WWW/Journal/1/isakowitz.187/paper/187.html.
4. H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle," *Proc. Sixth Int'l WWW Conf.* (WWW6), *Computer Networks and ISDN Systems*, Vol. 29, 1997, pp. 1,429-1,437; also available online at http://www.teco.edu/~hwg/www6/PAPER232.html.
5. R.A. Barta and M.W. Schranz, "Jessica: An Object-Oriented Hypermedia Publishing Processor," *Proc. Seventh Int'l WWW Conf.* (WWW7), *Computer Networks and ISDN Systems,* Vol. 30, Elsevier Science, Amsterdam, 1998, pp. 281-290; also available online at http://www7.scu.edu.
6. F. Coda et al., "Towards a Software Engineering Approach to Web Site Development," *Proc. Ninth Int'l Workshop on Software Specification and Design* (IWSSD-9), IEEE Computer Society, Los Alamitos, Calif., 1998.

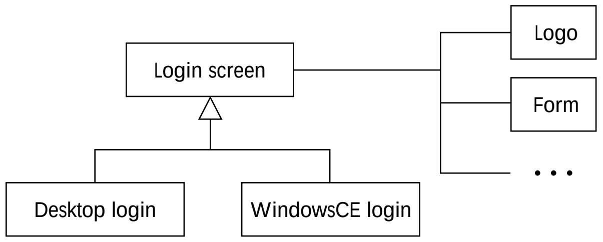

Figure 2. Design of HTML-based login screens for two browser platforms.

works that are defined in terms of abstract objects, such as the one described in Figure 1. Nor can it model the more specific design in Figure 2, which is also based on an abstract object and the notion of specialization by inheritance. The lack of abstraction means that a Web implementation cannot factor properties shared between objects into a generalized object but must build them into each specific object. The alternative of delegating shared code to a third object is also only minimally supported through extensions such as server-side includes.

Obviously, the lack of abstraction hampers construction of general components and frameworks. Further, it does not support the modular separation of concerns: Code for user interface elements cannot be separated from code for page layout.

Another problem, also illustrated in our small example, is the gap between higher level design and implementation. There is a drastic transition from the design—as represented in the object model shown in Figure 2—to its implementation. It is difficult to redesign during system evolution because a Web implementation cannot capture the concepts of the original design. The development process is not reversible, which means that design decisions are difficult to track and to access in the implementation.

AN OBJECT-ORIENTED MODEL FOR WEB APPLICATIONS

WebComposition is an approach to structured Web development that applies established object-oriented software development principles to the World Wide Web. The approach is based on a Web component model that abstracts from low-level Web implementation technologies to support seamless, reversible development of Web applications.

Figure 3 illustrates the overall WebComposition architecture. A resource generator maps the component model to a standard Web implementation. The model is maintained throughout the Web application's life cycle, facilitating component reuse and application evolution at a higher level of abstraction. In other words, the component model maintains the developer's view of an application, from which the Web view is derived incrementally.

We will briefly describe the component model and the concepts for resource generation (for more detail, see Gellersen et al.[1]). Then we present a new development, the WebComposition Markup Language, that implements the WebComposition concepts based on the World Wide Web Consortium's eXtensible Markup Language (XML).[7]

WebComposition Component Model

WebComposition defines an object-oriented model that uses components as a uniform concept for

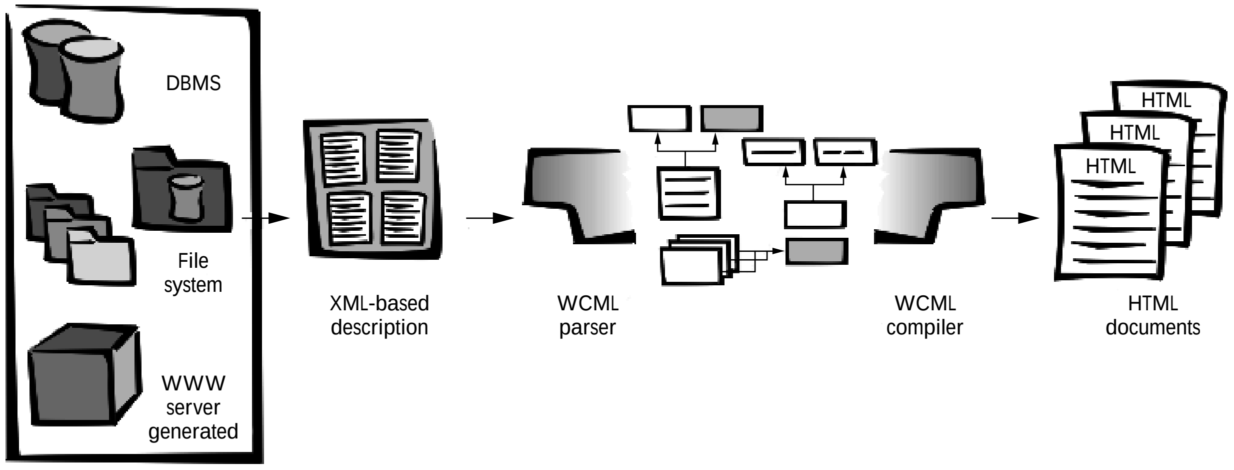

Figure 3. Overall WebComposition architecture. A resource generator maps the component model to a standard Web implementation.

```
<wcml>
<component id='CHeader'>
  <property name='text' value=“ />
  <property name='level' value=“ />
  <property name='content'>
    <H<<refprop name='level' />>
    <refprop name='text'>
    </H<<refprop name='level' />>
  </property>
</component>
<component id='CFooHeader'>
  <prototype is='CHeader' />
  <property name='text' value='This is a level 2 header' />
  <property name='level' value='2' />
</component>
</wcml>
```

Figure 4. Code describing the structure of a WCML document.

modeling Web entities at arbitrary levels of granularity and abstraction. In contrast to resources, components are not fixed to a certain grain size but designed instead to capture design artifacts at their natural granularity. For example, components can capture a content unit as design artifact independently of a Web page, which is a separate design artifact.

Support of arbitrary granularity means that components can model Web entities as small as individual links or layout resource fragments. They can also be associated with a complete resource—for instance, an HTML document or a script generating a Web document. In contrast to other proposed object-oriented Web models, such as WOOM,[8] WebComposition does not require object-components to be composed or derived from a given set of primitives; any set of primitives can be modeled as application building blocks—for instance, user interface primitives or primitives related to a specific design method.

Components can reference other components to model aggregation (has-part) or specialization (inherits-from). For example, a component modeling a page can reference components modeling header, body, or other parts for the purpose of delegation—in particular, delegation of the implementation. As another example, a component modeling a navigation structure can reference the components modeling the involved links and anchors.

By means of a special reference type, components can reference so-called *prototype components* from which they inherit state and behavior. Any component can function as a prototype. Thus, the WebComposition model is based on a *prototype-instance paradigm*, which eliminates the distinction of instances and classes as known in most object models.[9] We find this paradigm naturally suited to Web application modeling, first because many Web entities—namely, those modeling content—are unique and, second, because prototyping reflects the copy-and-modify type of reuse often applied in Web development. Because it is easy to emulate a class-oriented view from prototypes, the Web-Composition model aligns conceptually with any object-oriented design model.

The WebComposition model is a development model, not a runtime model. Implementation of a Web application from the model requires each component to implement a service for mapping its state to a representation in the standard Web implementation model. This mapping is not restricted to HTML; for example, components can in principle also be mapped to technologies such as CSS[10] and XSL.[11] Further, WebComposition defines a resource generator as a function that incrementally maps a WebComposition model of an application to resources in the operational Web environment. Incremental mapping is based on evaluation of component dependencies and tracking of changes committed in the component model.

The WebComposition model capitalizes on the well-known properties of object-oriented design—modularity, abstraction, encapsulation, and extensibility, while also retaining generality. The model is clearly defined, both on a conceptual level and as XML-based implementation technology in the WebComposition Markup Language.

WebComposition Markup Language

WCML is based on XML,[7] a metalanguage that facilitates definition of a tag-based textual format for semantic markup of documents or data. The XML document type definition of WCML describes a markup notation for WebComposition concepts—that is, for component descriptions, properties, and relationships. Figure 4 presents code that describes the structure of a WCML document.

A WCML document contains one or more components. Each component has an identifier for referencing; by convention, the identifier starts with a capital C. Notably, the structuring of components into documents is independent of whether the components relate to the same document in the target Web implementation. WCML documents

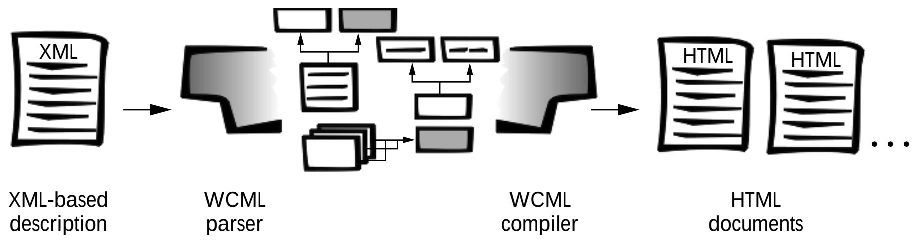

Figure 5. Compiling WCML to map component model to Web implementation.

```
<component id='CLogin'>
  <property name='image' value="/>
  <property name='version'>Version 1.122.58</property>
</component>
```

Figure 6. Code defining general login

```
<component id='CDesktopLogin'>
  <prototype is='CLogin'/>
  <property name='image' value='tecologo.gif'/>
  <property name='content'>
    <refprop name='content'
        from='CLogo' prototype='CDesktopLogin'/>
    <refprop name='content' from='CApplicationTitle'/>
    <refprop name='content' from='CForm'/>
    <refprop name='content' from='CRegister'/>
    <refprop name='content' from='CCopyright'/>
  </property>
</component>
```

Figure 7. Code for the desktop login screen.

can be used to organize components into modules that, for instance, capture a specific framework or a set of domain-specific building blocks. While HTML documents are a unit of application delivery at runtime, WCML documents are a unit of application development that support modularity and reuse.

A component is defined by a set of properties, which are simple name-value pairs. For instance, the component CHeader models an HTML header with properties defining text and heading level. According to the WebComposition model, every component must specify its own Web implementation. In WCML, this is accomplished with a special kind of property, content. In the Figure 4 code for CHeader, the content property describes the mapping of CHeader's state to a representation in HTML. The second component, CFooHeader, is derived from CHeader, which is specified with the prototype tag. Components inherit properties of their prototypes but can override them. In this case, CFooHeader defines specific values for text and level, and inherits the content property from CHeader.

The example shows the use of abstraction. Specific headers can be defined while simply inheriting the content property that defines the Web implementation. General changes to how a header is implemented in the Web could be carried out at an abstract level by modifying CHeader's property content. Such a change would affect all components derived from CHeader unless these components define their own content property.

WCML code generation is based on the content property and takes advantage of the availability of XML parsers for all major development platforms. Figure 5 illustrates the compilation process. As an XML-based description language, WCML enables exchange of components across operating system platforms. XML itself is based on Standard Generalized Markup Language technology for creation of interchangeable, structured documents, and so parsing of XML documents in general, and WCML in particular, is straightforward.

Abstraction and Reuse in WCML Applications

Let's return to the example login screen earlier. WCML can directly implement the login screen design shown in Figure 2. The generalized login screen defines properties that are shared by the browser-specific login screens. These are an image and a version number, both to be displayed in the logo, which is part of the login screen. Figure 6 shows the code defining these properties.

Figure 7 is the code describing the desktop login screen. The CDesktopLogin component uses the CLogin component as a prototype to inherit the version and image properties; it overrides the image property. In addition, CDesktopLogin defines a content property.

In the definition of its content, CDesktopLogin refers to other components that model parts of the login screen: a logo component based on an image and a version number, and components for the application title, a form element, a registration control, and a copyright note. CDesktopLogin delegates the content definition to these components by means of the refprop tag, and defines its content as a simple concatenation of the delegated contents. In the case of the logo component, CDesktopLogin passes on its own state by defining itself as a prototype of CLogo within the scope of the refprop reference.

The component for the Windows CE login screen is likewise derived from CLogin, inheriting the version number and defining an image for the logo, as shown in Figure 8.

```
<component id='CWindowsCELogin'>
  <prototype is='CLogin'/>
  <property name='image' value='tecologo256color.gif'/>
  <property name='content'>
    <table border="0" cellspacing="5"><tr><td>
    <refprop name='content'
        from='CLogo' prototype='CWindowsCELogin'/>
    </td><td>
    <refprop name='content' from='CApplicationTitle'/>
    </td><td>
    <refprop name='content' from='CForm'/>
    </tr></td></table>
    <refprop name='content' from='CCopyright'/>
  </property>
</component>
```

Figure 8. Code for the Windows CE login screen.

By using the same prototype as CDesktopLogin, both components effectively share the code that is generalized in CLogin. Further code sharing occurs by delegation to the same components referred to in the content property. In contrast to CDesktopLogin, this login screen arranges the content of its parts horizontally in a table, adapting to the Windows CE display. Another difference is that the Windows CE login does not contain a registration control, as registration of new users is not supported from mobile devices in the travel assistance system.

Figure 9 shows the two resulting screens, providing the same interface with different layout adapted to the platform requirements.

WCML Applications: Modifiable and Extensible

WCML components can have arbitrary granularity, which means that an application can be decomposed to the actual units of change. Regarding our example, the decomposition of login screens into smaller parts is a contribution both to reuse and to modifiability. Changes in parts of the login screen can be encapsulated in a component. For example, a modification of the form element would be localized in the CForm component and leave the definition of the login screen untouched.

Besides modification at different levels of decomposition, WCML supports modification at different levels of abstraction. General design decisions can be captured in abstract components, facilitating reconsideration in abstraction from implementation detail. As a simple example, the version number captured in CLogin can be modified without touching the specific login screen components.

New components can be added to WCML implementations of Web applications easily by reusing and modifying any component code in the system. For instance, a new login screen for another browser does not have to be defined as a prototype of CLogin but can use a more specific login screen as prototype, as shown in Figure 10.

Conceptually, CPsionLogin is derived from CLogin, but for more effective code sharing it derives its implementation from CWindowsCELogin.

CONCLUSION

The phenomenal popularity of the Web and its advantages as a client-server software platform have

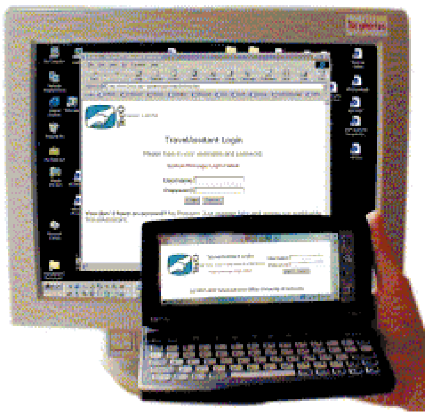

Figure 9. Travel assistant login screens for desktop browser and handheld browser.

```
<component id='CPsionLogin'>
  <prototype is='CWindowsCELogin'/>
  <property name='image' value='tecologo16gray.gif'/>
</component>
```

Figure 10. Code for a specific login screen as a prototype.

led to a wide range of Web applications, but development is still largely ad hoc. The Web implementation model imposes a structure on Web applications that does not relate well to established software development models, rendering it difficult to adopt structured software processes for the Web domain. Existing development environments and design methods provide abstractions from low-level implementation technology but lack generality and do not sufficiently address system maintenance and evolution.

The WebComposition model defines an object-oriented model for Web applications that abstracts from the Web implementation model and gives developers the power of object-oriented concepts for constructing reusable frameworks, for reuse by inheritance and delegation, and for improved modifiability and extensibility.

We believe that WCML, which is based on XML technology, contributes further toward a more seamless and reversible development process by enabling the definition of higher level abstractions for design-level modeling in a markup language. To investigate this claim, we are in the process of building a CASE tool based on OOHDM for analysis and design, and WCML for implementation.[12] ■

REFERENCES

1. H.-W. Gellersen, R. Wicke, and M. Gaedke, "WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle," *Computer Networks and ISDN Systems*, Vol. 29, 1997, pp. 1,429-1,437; also available online at http://www.teco.edu/~hwg/www6/PAPER232.html.
2. P. Ciancarini et al., "Coordinating Multiagent Applications on the WWW: A Reference Architecture," *IEEE Trans. Software Eng.* (special issue on Mobility and Network Aware Computing), Vol. 24, No. 3, 1998, pp. 362-366.
3. R. Itschner, C. Pommerell, and M. Rutishauser, "GLASS: Remote Monitoring of Embedded Systems in Power Engineering," *IEEE Internet Computing*, Vol. 2, No. 3, May 1998, pp. 46-52.
4. T. Berners-Lee et al., "The World-Wide Web," *Comm. ACM*, Vol. 37, No. 8, Aug. 1994, pp. 76-82.
5. M. Gaedke et al., "Web Content Delivery to Heterogeneous Mobile Platforms," paper presented at the Workshop on Mobile Data Access at the 17th Int'l Conf. on Conceptual Modeling, Singapore, 1998; available online at http://www.teco.edu/~gaedke/webe/er98/.
6. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1994.
7. World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 Specification*, tech. report, available online at http://www.w3c.org/TR/REC-xml.
8. F. Coda et al., "Towards a Software Engineering Approach to Web Site Development," *Proc. Ninth Int'l Workshop on Software Specification and Design* (IWSSD-9), IEEE Computer Society, Los Alamitos, Calif., 1998.
9. D. Ungar and R.B. Smith, "Self: The Power of Simplicity," *Proc. OOPSLA 87*, ACM Press, New York, 1987, pp. 227-242.
10. World Wide Web Consortium, *Cascading Style Sheets (CSS) Level 1 Specification*, tech. report, available online at http://www.w3c.org/TR/REC-CSS1.
11. World Wide Web Consortium, *Extensible Style Sheets (XSL)*, Working Draft, available online at http://www.w3c.org/TR/WD-xsl.
12. H.-W. Gellersen et al., "Patterns and Components: Capturing the Lasting amidst the Changing," paper presented at The Active Web, British HCI Day Conf., Staffordshire Univ., 1999, available at http://www.teco.edu/activeweb/.

Hans-W. Gellersen is a research scientist at the University of Karlsruhe, leading the Telecooperation Office (TecO) for applied computing research with partners in industry. His research interests are in methods and tools for disciplined development, operation, and evolution of WWW applications, and handheld and ubiquitous computing technologies. Gellersen received a doctoral degree in 1996 and a master's degree in computer science in 1992 from the University of Karlsruhe.

Martin Gaedke is a research assistant at the Telecooperation Office of the University of Karlsruhe, and the technical lead in collaborative Web engineering projects. His research interest is in application of software engineering practice to applications in the WWW, and specifically in design patterns and component technology for the WWW. Gaedke obtained a master's degree in computer science from the University of Karlsruhe in 1997.

Readers may contact Gellersen at Telecooperation Office (TecO), University of Karlsruhe, Vincenz-Priessnitz Str. 1, 76131 Karlsruhe, Germany; e-mail hwg@teco.edu.