

WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle

Hans-Werner Gellersen

Robert Wicke

Martin Gaedke

Telecooperation Office (TecO), University of Karlsruhe

Abstract: Maintenance of web applications is a difficult and error-prone task because many design decisions are not directly at run time, but rather embedded in file-based resources. In this paper we introduce the WebComposition system addressing this problem. This system is based on a fine-grained object-oriented web application model, and maintains access to it throughout the lifecycle for management and maintenance activities. Modifications of the model are made effective in the web by incrementally mapping the model to file-based resources.

Keywords: World Wide Web, Software Engineering, Web Application Maintenance, Object-orientation

1. Introduction

From a software engineering perspective the web is a new application domain. As is generally the case with new domains, engineering support is initially focussing on provision of enabling technologies and tools for ad hoc development. The dramatic growth of the web, though, is now leading to large-scale applications, often distributed over a number of sites and increasingly containing highly dynamic and interactive components. Such applications are no longer manageable with ad hoc methods. Software engineering for the web, subsequently referred to as *web engineering*, is challenged to provide structured approaches for development and maintenance of web applications.

In this paper, we will discuss modelling of web applications as foundation for web engineering tasks. We will point out, that for support of web engineering tasks fine-grained access to the concepts embodied in web applications is required. The decomposition of a web applications into file-based resources does not provide the fine granularity required for engineering tasks such as reuse and maintenance.

Subsequently, we will discuss the [state of the art in web engineering](#), presenting an overview first of existing tools and then of typical web engineering problems not addressed by these tools. In sections 3 and 4, we will introduce a new approach for support of web engineering activities. Our approach is based on supporting web applications throughout the lifecycle at the level of fine-grained objects rather than the coarse-grained level that file-based resources represent. We refer to these fine-grained objects as *web components* and to our approach as *WebComposition*. The [WebComposition component model](#) will be discussed in section 3. The model is implemented in the [WebComposition system](#), which will be discussed in section 4. The WebComposition system maintains fine-grained access to web components throughout the lifecycle, and is to be understood as open platform for web engineering tools rather than as self-contained engineering environment. In section 5, we will present a discussion of how the

WebComposition system can provide [lifecycle-spanning web engineering support](#). The remaining sections describe the [status of our work](#), plans for further work, and [concluding remarks](#).

2. State of the Art in Web Engineering

Existing Web Engineering Tools

Web engineering tools vary in the size and nature of web applications they support, and in lifecycle coverage. Regarding application size, tools may have the scope of supporting individual pages, web sites, or distributed web applications consisting of many sites. Regarding the nature of web applications, tools may support static publication-oriented hypertext, database-centric information systems, or highly dynamic and interactive applications. Static hypertext applications are characterized by both static links and static pages. Database-centric information systems on the web are characterized by dynamic page creation but static link structure. Dynamic applications are characterized by both dynamic page creation and dynamic link structure. Finally, regarding lifecycle coverage, tools may support different stages in the web application lifecycle.

A first generation of web engineering tools was limited to the scope of authoring individual pages of passive hypertext. These page editors have evolved from simple markup support to structure-oriented and directmanipulative editing. An example is Microsoft's FrontPage editor [1]. More recently, tools have been introduced to support development of sites rather than just pages. An example of a site builder tool is NetObjects' Fusion product [2]. Still, the scope is restricted to passive hypertext with a focus on presentation, for example supporting consistent layout across pages while providing only rudimentary support for navigation structures. Also the lifecycle scope is mostly restricted to ad hoc implementation. A very simple top-down site planning method provides only little design support, and maintenance support is restricted to appearance of layout elements. Maintenance is also impractical for large sites, as NetFusion does not support incremental publishing.

Another class of tools supports development of more dynamic applications. WebWriter [3] for example provides HyperCard-style abstractions for development of web applications containing scripts. WebObjects [4] is a more advanced product providing reusable interaction components, managing state in session-based applications, and supporting deployment of applications in a distributed environment. On the other hand, WebObjects does not support the organization and maintenance of resources. Also, despite introducing object-oriented concepts, reuse remains restricted as objects are embedded in static HTML templates. Yet another class of tools aims to address a wider scope of the application lifecycle. In particular, some work has been reported on integration of hypertext design methods with the web. For example, the RMCASE tool [5] supports web application development based on the RMM methodology [6]. This methodology though is only applicable to application domains that lend themselves to description in terms of entities and relations. Also, once a web application is deployed, modifications require complete re-generation of the resources.

Web Engineering Problems

Subsequently, some web engineering problems that are not sufficiently addressed by state of the art tools, will be pointed out. A first fundamental problem occurs when the design of a web application is deployed to an implementation of file-based web resources. Even highly intuitive and informal design approaches are based on concepts such as overall document structure and navigation paths. After mapping to file-based resources, these concepts are no longer accessible as entity. A navigation structure, for instance, is implemented by a set of links that happen to be embedded in a number of resources. Thus, maintenance of these design decisions is a difficult task, as these decisions are hard to trace in the implementation, and as their modification requires alteration of many resources, easily leading to inconsistencies.

Another often experienced problem is caused by the replication of design artifacts in an implementation. For instance, HTML code relating to the look & feel of a web site is commonly replicated on many web pages. Modification of such design artifacts requires multiple changes and easily leads to inconsistencies. A workaround is dynamic generation of such resources, which is a very questionable measure in those cases where page modification occurs at a substantially slower rate than page access. The lack of support for inheritance in web resources represents yet another key problem. In design, generalization and specialization are fundamental concepts for organization of web sites and web applications, for instance describing general page designs which can be refined to more specific designs for certain categories of pages. As more general design decisions can not be captured in abstractions, their maintenance can not be localized but requires modification of all effected specific design artifacts.

In summary, it has to be pointed out that file-based resources describe web applications at a granularity that is too coarse for reuse and for maintenance. Many design decisions are hard to maintain as they are difficult to access and often distributed over a number of run time artifacts. For better support of web engineering and in particular maintenance, a finer-grained web application model has to be maintained throughout the lifecycle.

3. WebComposition Component Model

Components

WebComposition is based on an object-oriented model of web software. According to this model, a web application is hierarchically decomposed into *components*. At higher levels in this hierarchy, a component may model a page or even a site but of course also application-specific concepts such as a document or a dialog consisting of a number of interrelated pages. Further down the hierarchy, components relate to parts of pages, such as for example tables, table entries, anchors, and referenced dynamic resources; of course, components at this level may also represent application-defined abstractions for page organisation, for example `overview section`, or `advertisement column`. The leaves in the component hierarchy are called primitives, the other components are called composites. The model does not prescribe the grain of primitives. For example, a text consisting of a number of paragraphs could be modelled as a primitive, if the text as a whole rather than its paragraphs presents the basic unit of manipulation.

A component is defined by its state and behaviour. The state of a component is defined by a list of properties. Properties are simply (name,value)-pairs. The value can be either a scalar or an array, and either be by-value or be a reference to a simple object. A simple object is a data container for a single

value of type `String`, `Integer`, `Boolean`, `Date`, `URL` or `ComponentReference`. Properties can for example model HTML attributes. A table component, for instance, would obviously have properties `border`, `cellspacing` and so on. Of course, a table component could have further properties, for example a default value for its entries.

The behaviour of a component is defined by operations on its state. All components support operation `getProperties`, `setProperties` and `generateCode`. The first two realize read respectively write access to a component's properties. The `generateCode` operation maps the state of a component to its representation in the web, for instance to HTML code. The relationship between component and its web representations is a model-view-relationship. For primitive components, for instance a chunk of text, this mapping is straightforward, and the operation can be thought of as consisting of a simple print statement. For composite components the mapping involves delegation of mapping to its children components. For simple composites such as a list, `generateCode` could simply iterate over its children, invoking their respective operations. In more complex cases, for instance tables, the composite has to generate its own code in addition to the code of its children.

Component Sharing and Prototyping

The WebComposition model supports *sharing* of components. For example, a navigation bar appearing on a number of pages would exist as a single non-replicated component in the model, shared by other components. In terms of model-view, a model (component) can have multiple views (web representations). Sharing is a means to reduce development effort through reuse. Further, it is a means to reduce maintenance effort through localization of potential modifications.

Components can be derived from other components based on a prototype-instance model^[7]. Following this model, a component may be an instance, but may also serve as *prototype* describing properties of a group of components. Any component can refer to any other component as prototype, inherit the prototypes properties, and add its own properties. The prototype-instance model is simpler than the class-oriented model. Both the class-oriented relationships *is-instance-of* and *is-subclass-of* are expressed in terms of *is-instance-of-prototype*, that is there is no distinction between classification and generalisation. With respect to components, that are represented by HTML code in the web, prototypes can be thought of as HTML templates. The benefit of prototype usage is similar to that of sharing (in fact, component instances share component prototypes). First, development effort is reduced through reuse of prototypes/templates, and secondly maintenance of common properties is localized.

Illustration of Component Model

Figure 1 depicts part of the component model of a press release system that we developed for the State Government of Baden-Württemberg (southwest Germany). The component in the bottom left models a web page with properties holding references to three components. Two of these properties, `logo` and `navigation`, are inherited from a more general component that models the corporate identity of the site and functions as web page template. The third property, `content` is defined by the more specific page component. `Content` refers to a component implementing a list of press releases, ordered by date of issue. This component is instance of a prototype for lists ordered by date. At prototype level, the mechanism for producing lists ordered by date is defined. At instance level, the title and list properties are initialized with references to specific string and list objects.

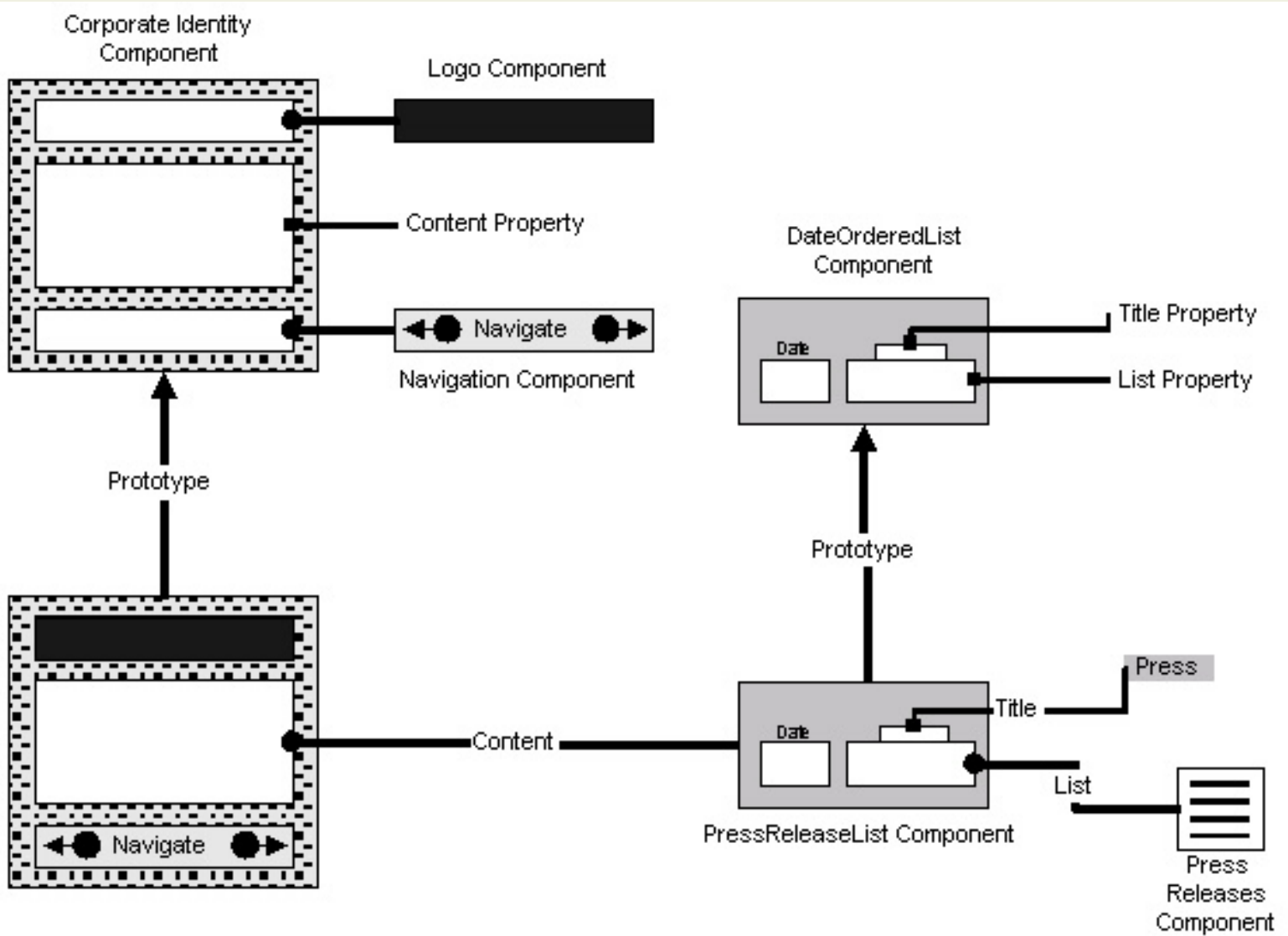


Figure 1. Component model of part of a press release system

4. WebComposition System

The WebComposition system is designed to maintain access to web applications at a fine granularity. It provides persistent storage for the component model of a web application, and maintains component access throughout the lifecycle. The component model can be mapped incrementally to file-based web resources. The system is transparent for the existing web.

System Architecture

Figure 2 shows the WebComposition system architecture. The Component Store provides persistent component storage based on an RDBMS. The Component Server provides component access to web engineering tools for support of development, operation and maintenance tasks. The Resource Generator maps a stored component model incrementally to file-based resources. It accesses the Component Store directly and not through the Component Server partly for reasons of performance but also because it requires meta-information about components that is not available from the Component Server.

The web server operates on the generated file-based resources and is not aware of the WebComposition system. So, the file system functions as a cache providing the web server with faster access to web resources than the WebComposition database system would be able to provide. This is in contrast to existing database-centric web applications where resources are always generated dynamically when requested by the server. In our approach, dynamic generation is driven by the WebComposition system rather than by access to resources. Drawing from our experience, we find that many dynamic components change their state at a much lower rate than their access rate. So it is straightforward to perform dynamic generation only when a change occurred and to provide access to a static copy of the dynamic model.

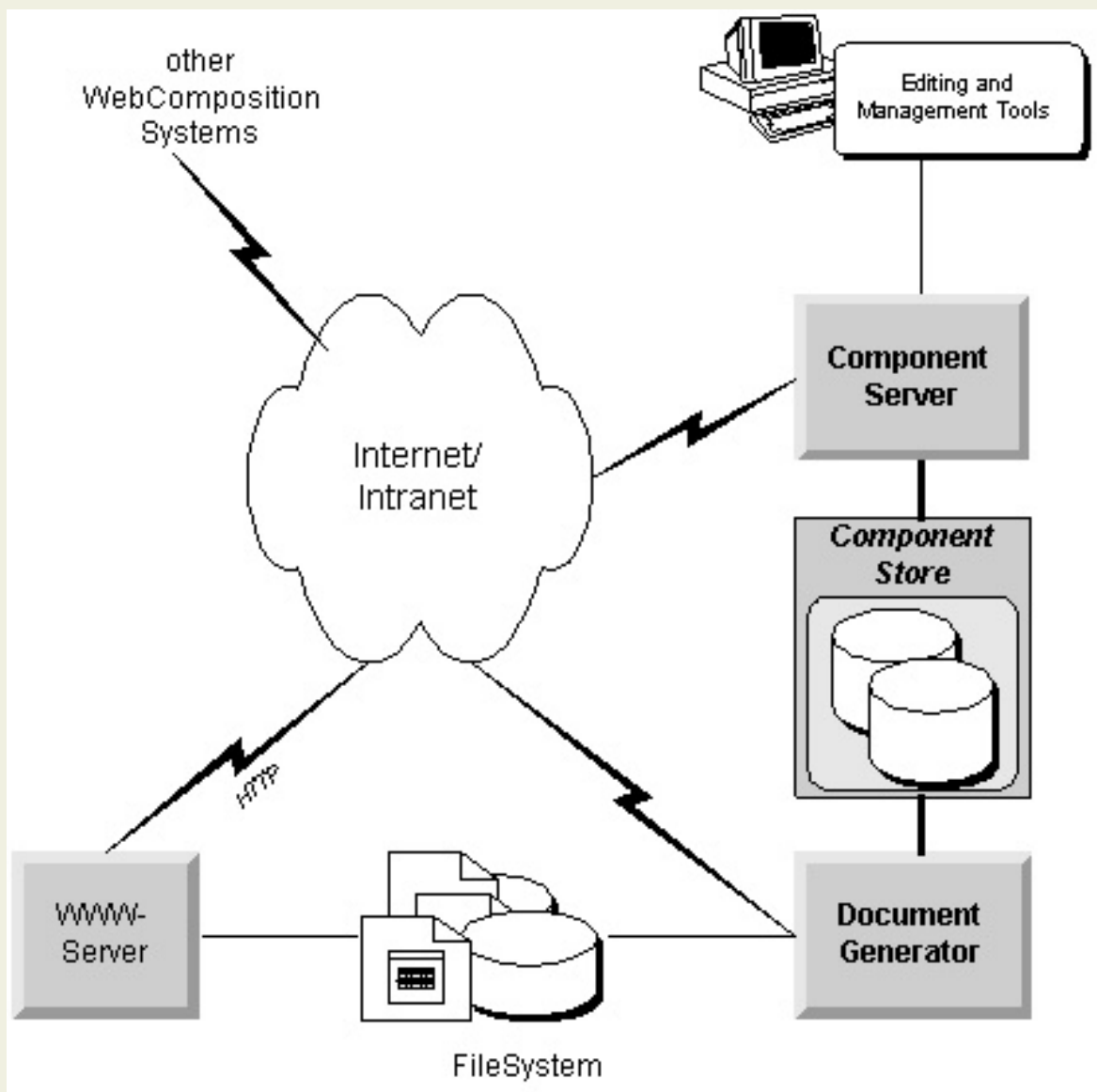


Figure 2. WebComposition system architecture.

The system components as shown in figure 2 are realized as servers for reasons of portability, distributability and scalability. The individual components may run on different platforms and may be distributed over the Internet. For example, a Component Store may hold the model of a web application distributed over a number of servers at different locations. From each of these locations a Resource Generator may access the ComponentStore through the Internet to map the location-related part of the component model to the file system at its site.

Component Store

Although the component model is object-oriented, we use a relational database management system for storage. Our primary motivation for the choice of an RDBMS was their wider-spread availability and acceptance in comparison to OODBMS. This is in line with our general goal of developing a support system that imposes only minimal system requirements on potential users. Besides, the simplicity of the prototype-instance model underlying our component model leads to a straightforward mapping to database tables. In order to increase portability of our system, we have implemented a database abstraction layer (DBAL) containing a minimal set of database-specific methods. The database access component (DBA-SQL), realizing database access through views and stored procedures, only uses methods provided by the DBAL. So, a port of the Component Store to another platform requires only modifications in the DBAL, while the DBA-SQL component is not affected. The DBA-SQL realizes database views for components and properties and stored procedures for their manipulation. Further, it implements revision control on component level. Besides DBA-SQL we have developed DBA-Java, a Java API for database access. In our implementation, the Java interface accesses the DBA-SQL through a JDBC/ODBC bridge.

Component Server

The component server provides access to components through a component access protocol. This protocol is transaction-oriented and supports checkin, checkout, unlock and get operations. Components are uniquely identified through a UUID and a version number. We have implemented the Component Server in Java, using DBA-Java for access to the Component Store.

The component server does not distinguish between instances and prototypes (in fact, there is no such distinction in the model anyway, cf. previous section). The component server though distinguishes between the properties a component defines itself and those properties inherited from other components. It grants read access to all properties but restricts write access to the instance-specific properties. Not yet implemented in the component access protocol but envisioned are two operations extending write access to inherited properties. First, a mutation operation that allows modification of an inherited property but then installs the modified property as instance property no longer subject to inheritance. Secondly, an operation is to be supported that allows modification of an inherited property and propagates the modification to the prototype that defined the property. This would enable modification of a group of similar components through exemplary modification of one of its members.

Resource Generator

The resource generator creates file-based resources from the component model of a web application. As described in [section 3](#), a web application is modelled as hierarchical decomposition into components. All the information required for mapping components to the file system is contained in the `generateCode` operations. For components of small grain this operation returns the code it contributes to a file-based resources. For components at the granularity of resources, this operation returns a filename. And for higher-level components, the operation may return a directory path or nil (when it returns nil, the directory tree is not further subdivided).

The resource generator can perform both a complete installation of a web application, and incremental modifications. For a complete installation, the resource generator proceeds top-down through the component hierarchy, top-down making directories, opening files and filling files with code. For

incremental code generation, the resource generator makes use of the component store's revision control. In this process, it generates those resources that contain components that have been modified. As resources themselves are represented by components, the dependencies among components can be evaluated to identify those resources that have to be newly generated.

5. Lifecycle-spanning Web Engineering Support

The WebComposition system is designed to maintain access to web applications at a fine granularity, and to support application-specific abstractions. In this section we will discuss the use of this facility for web engineering at different stages in the lifecycle.

First of all, the question arises how a component model of a web application comes into being. Of course, we could envision a highly structured approach identifying components middle-out and refining them, eventually specifying them with property editors at the level of detail required for resource generation. Yet we would be foolish if we neglected the current practice of web application development. While we believe, that a more structured approach is required we also acknowledge that for example page layouts are easier to develop with existing WYSIWYG editors. Also, we have to consider the use of filters, whose output is likewise not structured into components. In order to integrate the use of existing editing tools and filters with the WebComposition system, we plan to develop a tool parsing HTML and extracting components which then can be further processed, for example generalized. Such a tool would also support migration of already existing web applications to WebComposition-supported applications, although we clearly do not expect to fully automatize such a process.

At design time, the WebComposition system can function as repository for design artifacts, providing access for incremental refinement. As WebComposition supports a very fine-grained design representation, higher-level abstraction can easily be supported. For example, links which would be embedded in a file-based web representation, are directly accessible in the WebComposition system, so that abstractions such as typed links or navigation paths can be defined. Abstractions on top of the simple component model can be designed to meet the requirements of specific design methodologies. For example, in order to support the information mapping method, concepts such as Information Block, Information Map, and Hypertrail could be provided as abstractions on top of simple components. In general, we believe that different design methodologies ought to be supported, as web applications exhibit largely varying characteristics. In our web design experience, we found for example that ER-Modelling as underlying the RMM methodology is neither suitable for presentation-oriented hypertext, which can better be organized with for instance Information Mapping, nor for highly dynamic applications, which can better be developed with software engineering methods.

At run time, the WebComposition system serves as platform for tools that require access to the structure of a web application, including the page-internal structure. An example for such a tool that at run time regularly checks references to external resources and disables references whose resource can not be located. Another example, taken from the web-based press release system we developed for the State of Baden-Württemberg, would be a tool that moves components (*e.g. press releases*) after a certain display time (*e.g. 30 days*) from one resource (*News*) to another resource (*Archive*). Beyond supporting run time management of components, WebComposition in general supports manipulation of any component at run time. Thus, any maintenance activity can be carried out at run time; this includes maintenance activities that have not been foreseen at design time, for example alteration of a component that was thought to be static over its lifetime. Any modification of the component model, be it as consequence of a regular

management activity or a rather unanticipated maintenance activity, can be reflected in the file-based resources on which the web server operates, using the incremental resource generation facility of the WebComposition system. This facility ensures consistency among components, for example after removal of a component references to the component would automatically be removed as well. Also, consistency is ensured after modification of a component which is shared by other components. Similarly, the modification of a template is reflected in all components that are based on that template (i.e. prototype).

In summary it has to be noted, that the WebComposition maintains a web application model equally useful for design, run time management, and maintenance. Thus, the lifecycle with development, operation and maintenance is rendered seamless. The transition from development to operation is smooth as the development model (components) can be mapped incrementally to the operation model (resources). The transition from operation to maintenance and back to operation is virtually eliminated by tightly coupling the underlying models.

6. Status and Future Work

A first prototype of the WebComposition system has already been fully implemented and used for support of an internal web site. The prototype operates in a heterogeneous environment with NT and Unix platforms. Platform-specific components are the component store based on the Microsoft SQL-Server running on NT, and our web server running on a Unix machine. In the component store, a database abstraction layer has been implemented to localize platform-dependence. The component store realizes revision control and basic transaction management. The component server and the resource generator are both implemented in Java and use the DBA-Java API for database access over JDBC/ODBC.

In terms of tools operating on the WebComposition system, we have so far only implemented very simple component editors, enabling us to enter a component model into the system, and to modify their properties at run time. With this rudimentary system environment, though, we were able to test the WebComposition system. With this setup we were able to demonstrate the incremental generation of resources, a key feature of the WebComposition system. Further, we tested the system with a range of maintenance tasks that we considered as typical but poorly supported with state of the art tools. For example, we performed structural modifications such as mapping components, that originally were mapped to individual pages, now to parts within a single page while maintaining references to and among these components. Such a modification requires substantial recoding of HTML resources and easily results in loss of referential integrity. With the WebComposition system, the modification effort is reduced to editing of only few component properties: also, the system maintains referential integrity in the process of incremental resource generation.

Further work on the WebComposition system will on one hand address system usability and on the other hand address further research into disciplined web engineering. As for usability, we plan to develop easy-to-use component editors as well as integration of existing editing tools, which we perceive as fundamental for system acceptance. Further web engineering research is currently focussing on structured design and on reuse. For support of structured design, we are investigating higher-level abstractions on top of the WebComposition component model with the goal of integration with existing hypertext, information system and general software design methods. Regarding reuse, we are looking into the provision of reusable patterns defined in terms of component prototypes and instances, and into general reuse management.

7. Conclusion

At run time, major web design decisions are embedded in file-based resources and often even distributed. As a result, many web maintenance activities are difficult to perform and easily introduce inconsistencies. The WebComposition system introduced in this paper addresses this problem, realizing fine-grained access to web applications throughout the lifecycle. The WebComposition system is based on a prototype-based object-oriented model of web components. This model provides the fine granularity required for direct access to design concepts, for development of higher-level abstractions such as link structures, and for reuse of components and component frameworks. At run time, the WebComposition system functions as support system, transparent for the actual web. Management and maintenance activities are carried out by accessing the WebComposition system, which uses its resource generation facility to make modifications effective on the web.

We are not aware of other work directly comparable to WebComposition. Other work on integrating object-orientation has been reported but is aimed at different aspects. W3Objects [8], for instance, is in contrast to WebComposition based on a coarse-grained notion of objects encapsulating resources (the same notion is underlying the many efforts of integrating the web with CORBA objects). Major concerns of W3Objects are management and transparent migration on the level of resources. More closely related to our approach is the WebObjects system. In contrast to WebComposition, though, the WebObjects focus is on deployment management of session-based applications; maintenance support is not addressed by WebObjects.

Bibliography

- David Ingham, Mark Little, Steve Caughey and Santosh Shrivastava. *W3Objects: Bringing Object-Oriented Technology to the Web*, Fourth International World Wide Web Conference, Boston, Massachusetts, 1995, USA.
 URL: <http://www.w3.org/pub/Conferences/WWW4/Papers2/141/>
1. FrontPage Home Page, Microsoft Corp.,
 URL: <http://www.microsoft.com/FrontPage/>
 2. NetObjects Home Page,
 URL: <http://www.NetObjects.com/>
 3. Arturo Crespo and Eric A. Bier. *WebWriter: A Browser-based Editor for Constructing Web Applications*, Fifth International World Wide Web Conference, Paris, France, 1996.
 URL: http://www5conf.inria.fr/fich_html/papers/P35/Overview.html
 4. The WebObjects Home Page, NeXT Software Inc.,
 URL: <http://www.next.com/WebObjects/>
 5. Alicia Diaz, Tomas Isakowitz, Vanesa Maiorana and Gabriel Gilabert. *RMC: A Tool to Design WWW Applications*, The World Wide Web Journal, Issue One, Dec. 1995.
 URL: <http://www.w3.org/pub/WWW/Journal/1/isakowitz.187/paper/187.html>
 6. T. Isakowitz, E.A. Stohr and P. Balasubramanian. *RMM: A Methodology for Structured Hypermedia*

Design, Communications of the ACM, August 1995.

7.D. Ungar and R.B. Smith. *Self: The Power of Simplicity*, OOPSLA '87 Proceedings, p. 227-242, 1987.

8.David Ingham, Mark Little, Steve Caughey and Santosh Shrivastava. *W3Objects: Bringing Object-Oriented Technology to the Web*, Fourth International World Wide Web Conference, Boston, Massachusetts, 1995, USA.

URL: <http://www.w3.org/pub/Conferences/WWW4/Papers2/141/>

About the Authors

All authors share the following address:

*Telecooperation Office (TecO), University of Karlsruhe
Vincenz-Prießnitz-Str. 1, 76131 Karlsruhe, GERMANY*

<http://www.teco.uni-karlsruhe.de/>

Ph.: +49 (721) 6902-49, Fax: +49 (721) 6902-16

Email: we@teco.uni-karlsruhe.de (Web Engineering Group)

[Hans-W. Gellersen](#) studied Computer Science at the Universities of Karlsruhe/Germany and Auckland/New Zealand and obtained both his MSc and his PhD from Karlsruhe. Since 1992, he has been with the Institute for Telematics as Research Assitant and since 1996 also as manager of the Telecooperation Office (TecO), a technology transfer center with focus in telematics applications. His general research interest is in software engineering for telematics applications.

[Robert Wicke](#) is currently preparing a thesis on web data management for completion a research degree in Computer Science at University of Karlsruhe. He has been working part-time at the TecO since 1994 as webmaster and technical leader of various web related projects.

[Martin Gaedke](#) is currently completing a thesis on web engineering towards a research degree in Computer Science at University of Karlsruhe. Since 1994, he has been working part-time at the TecO, giving technical trainings on web technology.