M. Gaedke, C. Segor, H.-W. Gellersen (2000): WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. 2000 ACM Symposium on Applied Computing (SAC 2000), Villa Olmo, Como, Italy, March 19-21, 2000.

WCML: Paving the Way for Reuse in Object-Oriented Web Engineering

Martin Gaedke Christian Segor Hans-Werner Gellersen

Telecooperation Office
University of Karlsruhe
Vincenz-Prießnitz-Str. 1
76131 Karlsruhe
Germany
Tel.: ++49 721 690279

{gaedke, segor, hwg}@teco.edu

ABSTRACT

Since 1990 the Web has changed from a distributed hypertext system to an efficient environment for application delivery. Due to the legacy of the Web implementation model the development, management, and evolution of complex Web applications suffer from the coarse-grained model-entities. The demands can not be achieved without the application of programming technology and software engineering practice to applications in the Web, which is also referred to as Web Engineering. It has been recognized that the gap between design models and the implementation model of the Web leads to a development process that is hard to maintain, because fine-grained design entities get lost in the coarse-grained implementation model. Therefore, the coarse-grained implementation model hinders the maintenance and reuse of parts of an application. In this paper, the impacts on Web Engineering due to the coarse-grained Web implementation model are discussed and typical solutions that are related to the above problems are introduced. Then, the object-oriented WebComposition Markup Language will be presented as basis for a generic approach to component-based Web-application development.

Keywords

Web Engineering, XML, Component-based Web Development, Object-orientation, Software Reuse.

1. INTRODUCTION

The development of applications for the World Wide Web has progressed in the last few years and the Web has changed from a simple medium for publishing into a standard platform for distributed applications [3]. While the Web was originally designed to provide and link large quantities of information for distributed research teams, it is now also used by companies and institutions to make essential information available, to connect their legacy systems, and in general to deploy Web-enabled applications. Nevertheless, the development discipline usually has remained "ad-hoc", thus resulting in applications of poor quality and causing tremendous costs for maintenance and further evolution. The reason is the obvious lack of structure in these "ad-hoc"-engineered applications, and therefore also a lack of traceability.

It is widely recognized [2, 4, 8] that the lifecycle of Web applications is no longer manageable without Web Engineering, i.e. the application of software engineering practice to the Web. Every engineering discipline should be based upon an approach to system design which maximizes reuse of existing components [12] - unfortunately, design and code reuse for quality improvement and cost reduction in the Web is a tiring venture. The main cause for this problem can be found within the Web implementation model itself: because of its deliberately simple and coarse-grained nature [3, 6], it is perfect for easy authoring and straightforward publications of documents, with file-based resources being a suitable unit for development and modification. Unfortunately, this structure is more or less useless as a base for software engineering techniques.

State-of-the-art Web applications require a new kind of characteristics, for instance accessibility anywhere and anytime by anyone, accommodation of different client capabilities (e.g. varying screen resolutions [4]), and adaptation to the fast growing technology they operate with. The WebComposition approach [6] has been proposed as a foundation for various Web engineering tasks. Its main purpose is to maintain fine-grained access to an object-oriented model and automatically map entities of this design model to the resource-based Web implementation model. Based on the WebComposition approach we introduce the WebComposition Markup Language (WCML) in this paper. In the following section we will discuss the implications of mapping (OO-) design concepts to the Web implementation model and how Web Engineering can profit from reusing both design and code fragments. Section 3 elaborates on WCML, an XML-based markup language that allows object-oriented development of Web-based applications. The WCML processor is described in section 4; and section 5 focuses on how WCML encourages the reuse of design and code. In section 6 we take a short glimpse on related work and draw conclusions.

2. WEB EVOLUTION - WHY REUSE MATTERS

The Web implementation model, founded on file-based resources, does not provide the possibility of modelling higher-level design

concepts that go beyond the granularity of a file. Structures such as dialogues in session-based applications, user interface objects in interactive applications, or corporate identity elements represented by a single design-entity have to be duplicated in many resources. This makes it hard to maintain the application and enforce a disciplined evolution of the application without destroying the concepts of the original design.

Although there are design methods and systems available that support the mapping of higher-level concepts and fine-grained entities to the Web (such as OOHDM[11], RMM[7], Jessica[2], or TML[8]), the reverse mapping and possible distribution for maintenance and reuse of higher-level concepts is only poorly supported. Furthermore, because of the lack of structure in Web application code, it is hard to reuse code in an application, define code for reuse, or reuse code for different target systems. These restrictions in the overall development process prevent lower production and maintenance costs along with increased quality.

The reuse of design and code is successfully practised and a main task of software engineering [1, 9, 10]. A few examples of how reuse facilitates the development during the lifecycle of applications are:

- The reuse of applications by support for different systems or machines
- The reuse of functionality provided by standard libraries like mathematical libraries
- The reuse of component-based software such as Java Beans or DCOM/COM components
- And, finally, the reuse of design knowledge like design pattern and frameworks

However, these practices seem to be less common in today's Web application development.

Furthermore, and even worse, the maintenance of large Web applications is most often carried out by site engineers instead of the original authors of the contents. Often, these engineers end up mentally reconstructing the original higher-level design concepts from the implementation itself, in order to keep the semantics when modifying the site. Certainly, this error-prone strategy results in the loss of application integrity and leads to inconsistent systems.

To put it all in a nutshell, there are two main aspects complicating or even preventing the usage of software engineering techniques for the development of Web applications:

- Design and code reuse can only be done in a very unsatisfactory way.

  Design concepts frequently relate only to fragments within a resource, to structures composed of such fragments, or to interlinked resources. As these concepts are not easily accessible from within a resource-based Web implementation, they are hard to reuse. In fact, reuse can be done only by replicating the respective design concepts.

- There is hardly any support for the object-oriented concept of inheritance in Web development.

  In design, generalization, polymorphism and specialization are fundamental concepts for organization of Web sites and Web applications, for instance describing general navigational concepts or page designs which can be refined to more specific designs for certain categories of pages. Nevertheless, when it comes to implementing the design conventional methodologies fall back upon the simple resource-based development technique that follows the Web implementation model.

In the following section we describe the WebComposition Markup Language, whose main intention is to provide developers with an object-oriented technology to enable them to use their software engineering skills in spite of the coarse-grained Web implementation model Since the general architecture of the WebComposition approach remains transparent for the already existing infrastructure (such as Web servers etc.), no severe changes of the configuration are necessary to use WCML.

3. THE WEBCOMPOSITION MARKUP LANGUAGE (WCML)

The WebComposition Markup Language is an application of the eXtensible Markup Language (XML) [14] that paves the way for object-oriented Web engineering based on the WebComposition approach. WCML enables developers to reuse designs and code fragments by providing a simple notation that is capable of defining objects and their relationships.

3.1 The WebComposition Model

In WebComposition, Web entities are modelled as components with a state and a set of operations specifying the component behaviour. Components can model Web entities with respect to a variety of target languages and of arbitrary granularity, i.e. links, anchors, layout fragments, or even complete pages, scripts, or groups of resources. Components can reference other components to model aggregation (has-part relationship) or specialization (inherits-from relationship). WebComposition is based on a prototype-instance OO model [13] as opposed to a class-oriented OO model. Components may be used like an abstract class, i.e. every component can be a prototype for another component.

Prototyping is a mechanism to implement code sharing among objects. Another possibility to share the code of a component is to allow multiple references on the same component. Sharing is fundamental to reuse and for maintainability as it helps keeping modifications local.

Components described in WCML reside in a WCML document, which we refer to as a virtual component store, in conformity to the WebComposition system described in [6]. In the following sections, we present the specification of the WebComposition Markup Language by describing the XML elements and their meanings. Like all XML documents a WCML document consists of a prologue and the content containing the markup. Figure 1 shows an example virtual component store, which will be referred to throughout the remainder of this section.

3.2 Components and their Properties

According to the WebComposition Model, a WCML document consists of a set of component declarations, with their properties and their relationships defined by inheritance and aggregation. As mentioned above, a set of components is stored in a virtual component store, i.e. a file-based WCML-document.

Each component is identified by a universally unique identifier (UUID); for the sake of clarity and readability we will refrain from using "real" UUIDs in this text. Instead, we fall back upon hu-

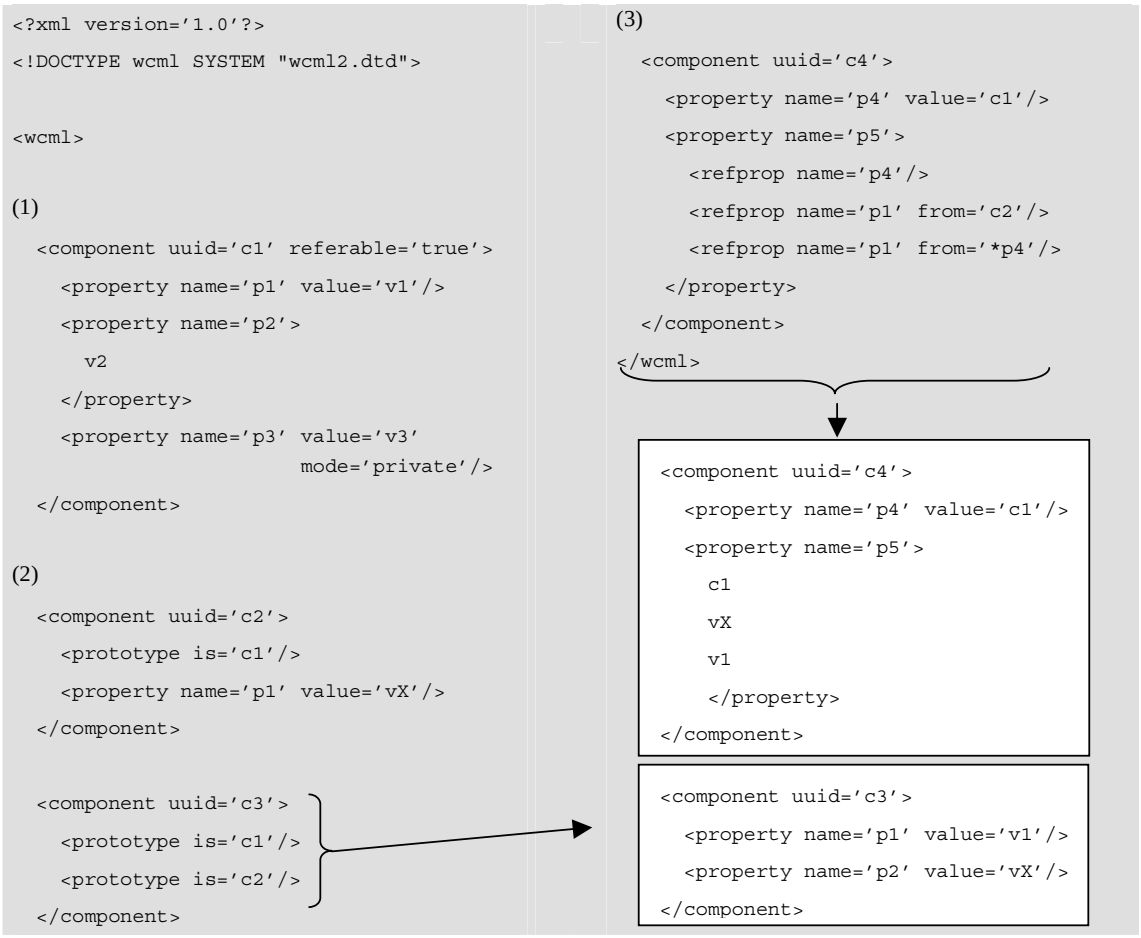

Figure 1: A simple WCML example

man-readable identifiers like "component1". A component must have at least one property and can have any number of prototypes (see below). Furthermore, it can be specified whether the component itself should be referable in the generated code. If this switch is set to true an HTML anchor tag is created by the WCML compiler.

Properties are simple (name, value)-pairs, whereas two notations are possible as shown in Figure 1 (1). Furthermore, a property can be defined in different modes, as there are:

- public
  This is the standard setting. A public property is visible and referable everywhere.
- private
  A private property is only valid within the component where it has been defined.

Figure 1 (1) shows examples for the above.

3.3 Prototypes and Inheritance

Since WebComposition uses the prototype-instance-model [13] for modelling inheritance, every component can be used as a prototype by other components. An inheritance tree is implicitly built using the appropriate prototype-statements by the compiler. This facilitates the code and provides a simple but powerful means for reuse and code sharing.

The WebComposition model supports multiple inheritance that allows a component to have more than one parent. Since a component inherits all properties of its ancestors and because properties can have the same names in different ancestors, a mechanism to avoid ambiguities is required. In WCML we use ordered multiple inheritance, which means that in case of conflicts the value is used that belongs to the last component that has been inherited from.

Inherited properties can be redefined, which will overwrite the original value of the property in order to specialise components for different tasks. Figure 1 (2) shows these concepts along with the component derived from the inheritance tree by the compiler.

3.4 References and Linking

Properties can be referred to within any other property, i.e. the reference to the property name is replaced with the value of the referred property. A reference can be further qualified with the *from* attribute in order to refer to a property from a particular component.

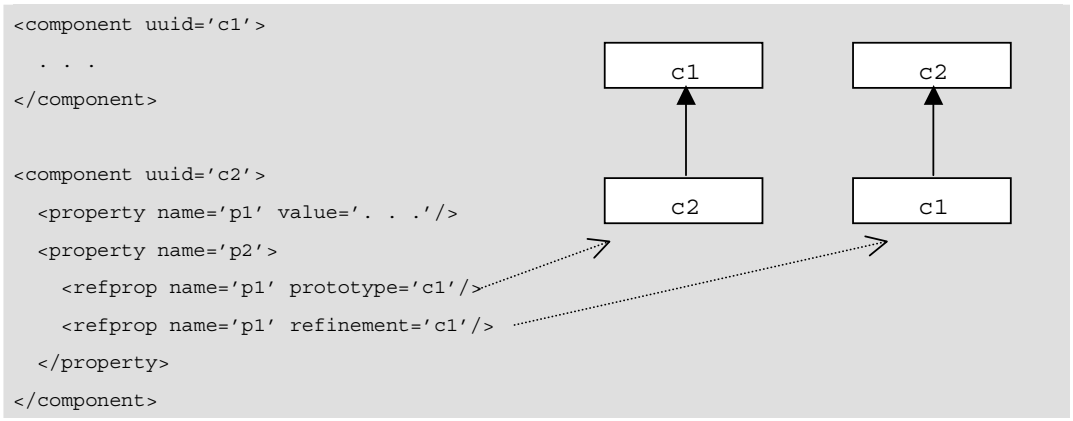

Figure 2: Parameterised property reference and the respective inheritance tree

Furthermore, property references can be indirect in the sense that the *from* attribute does not point to the component containing the referred property but to another (local) property (marked with an asterisk – similar to C). The local property then contains the UUID denoting the actual component that includes the desired property. This supports abstraction and thus facilitates the definition of reusable implementation patterns. Again, Figure 1 shows an example for these notions (see (3)).

By using the object-oriented capabilities of WCML it is possible to parameterise property references. For this reason, a further component stated in the *prototype* or *refinement* attribute of the reference tag is temporarily inserted into the inheritance tree. The value of the desired property then is derived from the modified tree, which is discarded after use. In this scenario, the properties of the further component act as arguments – this provides an efficient technique to build generic components that can be fully parameterised in order to fulfil special tasks. Figure 2 shows an example together with the respective temporary inheritance trees for the usage of the *prototype* and the *refinement* attribute. The difference between both is the place where the "argument component" is inserted into the tree: components called with *prototype* are inserted above the referred component; components called using *refinement* are put below the referred component.

Relationships between components on a conceptual level can be defined using a special variety of references, called "link properties". The compiler maps link properties to the corresponding hypertext links. The link property is identified by the attribute *to* denoting the referred component. For example, `<PROPERTY name="link" to="c1.c3"/>` will be resolved to the value `uuid.html#c1.c3`. This property defines a link to the anchor of component `c3`, which resides (may be not only) in the content of component `c1`, as shown in Figure 3. The necessary HTML tags with their attributes must be provided by the referring component, as the target language is not determined.

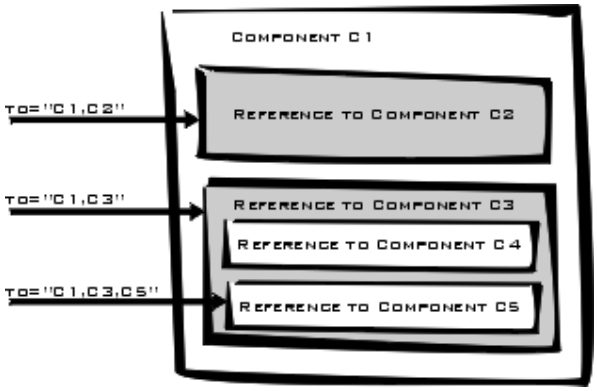

Figure 3: Hyperlink-support - Resolution of the Link-property

The benefit of modelling the hypertext links in this way is the possibility to define the links outside the components and thus to redefine a navigation structure or even define multiple navigation structures for the same components without modifying any components.

3.5 Factories: Getting productive

A wide-spread class of Web applications are Web-based information systems, designed to provide access to more or less extensive information usually stored in a database system. Conventional applications therefore require some mechanism to retrieve the data from the database; some well-known solutions include cgi-based database integration, server-side scripting for dynamic page generation or client-side approaches like JDBC. In any case, a great number of information pages is generated, which usually look all the same but contain different data.

When these Web information systems are regarded from a Web-Composition point of view, it becomes clear that there are two possibilities how to implement such systems:

- A special component is required, which implements one of the database integration approaches mentioned above. This solution is quite efficient and does not cause any special problems for the developer – the required code is simply encapsulated by the component. However, this component will probably contain code that is specific to the target system, depending both on the system software and the database interface.

- For each information page one component is created. The advantage of this approach is that platform-independence is preserved.

Nevertheless, the latter strategy requires some mechanism to create the information components automatically, based on given data. In WCML, the notion of a factory provides such a mechanism.

In general, a factory is a special component that contains the required information for producing any number of components according to a production schema. The main elements of a factory are:

- the data source
- a query expression to select data sets from the source
- a template defining how the "products" of the factory should look like

The data source is stated as a URL addressing an XML-file that contains the data. Using this approach, data can be stored in the file system, but also can be dynamically derived from a database using a simple mapping server. A subset of this data can be selected by stating an XQL (XML Query Language) expression, which is evaluated by the compiler. The results are taken dataset by dataset and inserted into the template, so that a new component is created for every dataset. The UUIDs of these components are generated by the compiler using well-know algorithms for this purpose.

The following section focuses on how WCML code is analysed and translated by the WCML compiler.

4. PROCESSING WCML

A WCML document is processed by the WCML compiler mapping the described components of the Virtual Component Store to the Web implementation model respectively the target language. The WCML compiler is implemented in Java and as DCOM Component using an arbitrary XML parser with DOM and XQL-support. Thus, the remaining tasks for the WCML compiler is to accomplish the presentation operation for the components and to resolve the different properties including the links.

In Figure 4 the integration of the WCML compiler with the existing system is depicted.

The main goal to provide a possibility that enables Web engineers to reuse object-oriented design and code or develop code for reuse is accomplished by using XML. The components are provided by WCML documents (XML-based description) accessible through the file system, a database system, or for distributed development support through a Web server. In the following step, the WCML document is parsed by the WCML parser respectively an XML parser.

After the parsing process of the components succeeded, the compiler analyses the composition of the components. Therefore, the properties with their references and strings are processed to generate the presentation output. The analysis step takes the WCML descriptions for function calls, hyperlinking, polymorphism, aggregation and specialization into account and evaluates the enrolled component by enumerating all affected name-value pairs on demand.

In the final step, the compiler creates the target resource given by the UUID of the component, if not otherwise specified by the filename and directory properties. The content of the component is then passed through to the file respectively to the Web-server, if the compiler serves as Web-server extension, servlet or is called as CGI-application.

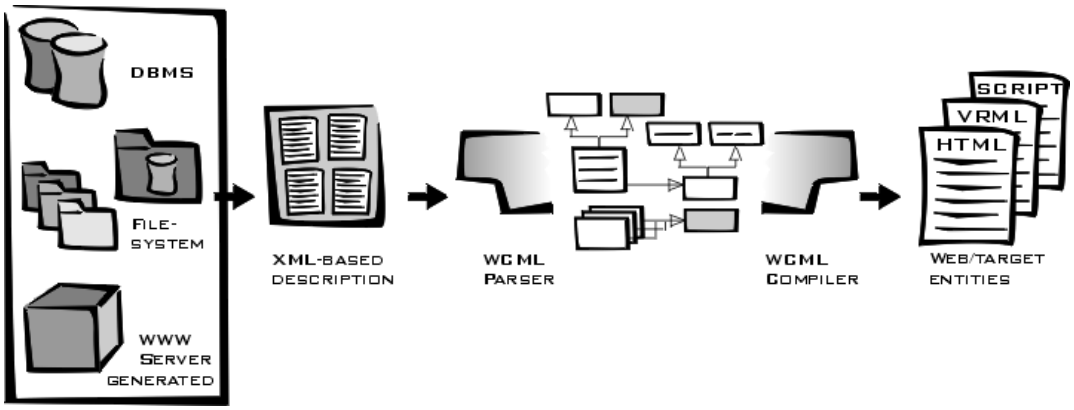

Figure 4: Integration and Processing of WCML

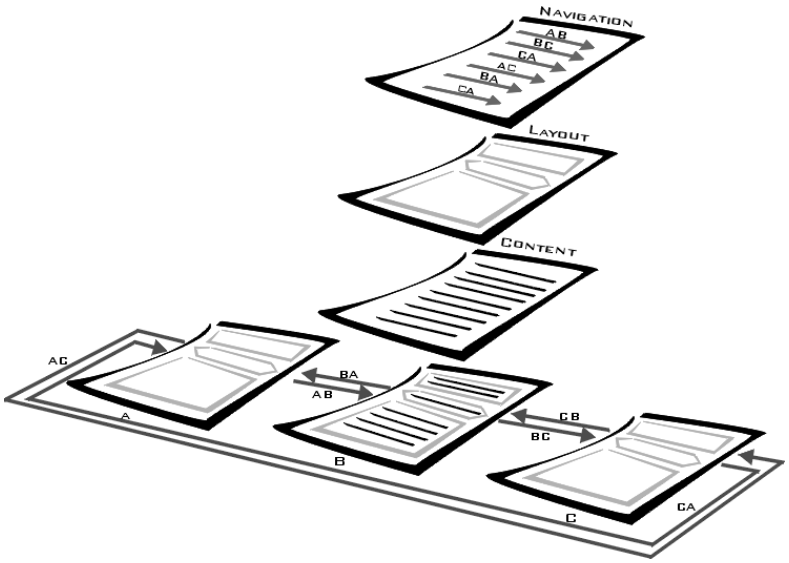

Figure 5: Decorator use for Navigation and Layout

5. APPLICATION DEVELOPMENT WITH WCML

As demonstration, we present a small WCML application in which one information component is reused for the generation of different information pages. This is done by falling back upon a well-know design pattern – the decorator [5], a flexible alternative to subclassing for extending functionality. In the Web environment the notion of a decorator can be adopted to easily create different views of one information item. Figure 5 shows how decorators can be used to specify navigation and layout of a Web page.

The following paragraphs show step-by-step how to reuse code for the generation of different documents in WCML. The results of these code fragments are shown in Figure 6. For the sake of clarity some special tags, which are necessary to identify HTML code, have been omitted here. Instead, HTML code is printed in bold.

First, we have to state the document prologue and open the WCML document:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE wcml SYSTEM "wcml2.dtd">
<wcml>
```

Then, a component representing the information itself is defined:

```
<component uuid='Chapter1'>
  <property name='title' value='Introduction'/>
  <property name='content'>
    The development of applications for the
    World Wide Web . . .
    . . .
  </property>
</component>
```

Each decorator must be described in an extra component:

```
<component uuid='HTMLDecorator1'>
  <property name='content'>
    <H1>
    <refprop name='title'
             from='*information.component'/>
    </H1>
    <refprop name='content'
             from='*information.component'/>
  </property>
</component>


<component uuid='HTMLDecorator2'>
  <property name='content'>
    <B>Title of document:</B>
    <refprop name='title'
             from='*information.component'/>
    <HR>
    <B>Content:</B><BR>
    <refprop name='content'
             from='*information.component'/>
  </property>
</component>
```

These components display the information contained in the given information component in two different ways. In order to generate the "decorated information" a further component must be defined, which sets `information.component` to the appropriate value:

```
<component uuid='Chapter1StyleHTML1'>
  <prototype is='HTMLDecorator1'/>
  <property name='information.component'
```

```
            value='Chapter1'/>
</component>
```

To create a differently decorated Web page, another component is defined:

```
<component uuid='Chapter1StyleHTML2'>
  <prototype is='HTMLDecorator2'/>
  <property name='information.component'
            value='Chapter1'/>
</component>
```

Other information components can be displayed using the same layouts by simply changing the value of the `information.component` property. This technique itensifies and simplifies the reuse of both the information and the decorator component. Both information and layout are kept in one single location, and changes are easily promoted throughout the Web site by the reuse and inheritance mechanisms of WCML.

Furthermore, code of other target languages can be created by just defining an appropriate decorator and the respective output component:

```
<component uuid='LaTeXDecorator'>
  <property name='content'>
    \section{<refprop name='title'
                from='*information.component'/>}
    <refprop name='content'
             from='*information.component'/>
  </property>
</component>
```

```
<component uuid='Chapter1StyleLaTeX'>
  <prototype is='DecoratorLaTeX'/>
  <property name='information.component'
            value='Chapter1'/>
</component>
```

Finally, the WCML document must be closed:

```
</wcml>
```

Figures 6 shows screenshots of the three representations of `Chapter1` produced with the different decorators. The upper left screenshot depicts `Chapter1StyleHTML1`, the screenshot on the right side shows `Chapter1StyleHTML2`. At the bottom, `Chapter1StyleLaTeX` is depicted.

6. CONCLUSION

Web development suffers from the coarse-grained Web implementation model that makes it hard to map fine-grained design entities to the implementation model. Due to the tremendous progress in Web technology and the Web model itself, it is difficult to reuse code and design for cost reduction and quality improvements. In this paper, we introduced the WebComposition Markup Language that enables the object-oriented specification of Web content. In the WebComposition model, a site with its different entities is composed in terms of components of arbitrary granularity. Components can capture the entities that are basic units for more complex patterns, but that are hidden in resources in a standard Web implementation. One instance of such patterns is views on the content that can be modelled following the idea of the decorator design pattern. We propose that using WCML, which is an application of the XML, enables the development of reusable hyper-

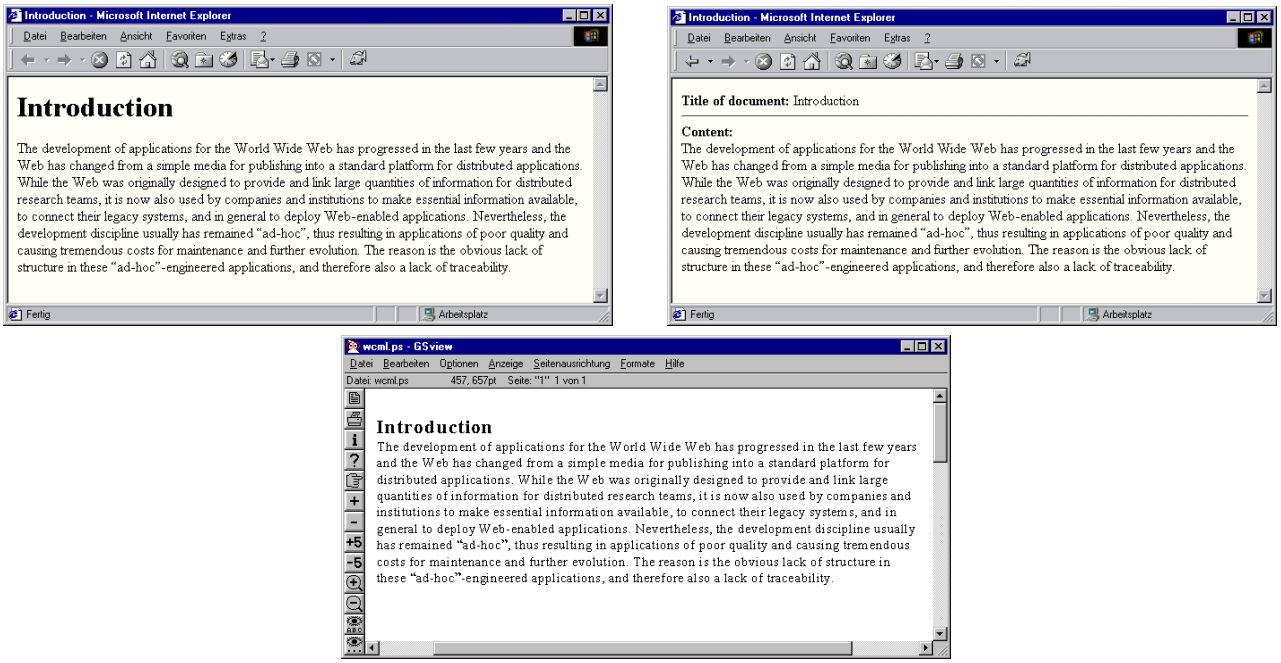

Figure 6: Chapter1 decorated with HTMLDecorator1 (top left), HTMLDecorator2 (top right) and LaTeXDecorator (bottom)

media structures. In addition, by using XML for describing the components we can fall back on existing XML-Editors or easily develop custom editors using the DTD of the WCML and an existing XML-Parser.

Further work on the support for reuse in Web Engineering aims at the development of an open WCML component repository. The repository is being built using hypermedia technologies to simplify the component access. We also investigate on design pattern for hypermedia applications and their integration in the repository.

7. REFERENCES

[1] S. W. Ambler, *Process Patterns - Building Large-Scale Systems Using Object Tecchnology*. New York, NY: Cambridge University Press, 1998.

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

[3] F. Coda, C. Ghezzi, G. Vigna, and F. Garzotto, "Towards a Software Engineering Approach to Web Site Development," 9th International Workshop on Software Specification and Design (IWSSD), Ise-shima, Japan, 1998.

[4] M. Gaedke, M. Beigl, H.-W. Gellersen, and C. Segor, "Web Content Delivery to Heterogeneous Mobile Platforms," *Lecture Notes in Computer Science (LNCS)*, vol. 1552, 1998.

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

[6] 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 (1997), pp. 1429-1437, 1997.

[7] T. Isakowitz, E. A. Stohr, and P. Balasubramaninan, "RMM: A Methodology for Structured Hypermedia Design," *Communications of the ACM*, vol. 38, No. 8, pp. 34-44, 1995.

[8] A. Kristensen, "Tempate resolution in XML/HTML," *Computer Networks and ISDN Systems*, vol. 30 (1998), pp. 239-249, 1998.

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

[10] J. S. Poulin, *Measuring software reuse : principles, practices, and economic models*. Reading, Mass.: Addison-Wesley, 1997.

[11] D. Schwabe and G. Rossi, "An Object Oriented Approach to Web-Based Applications Design," *TAPOS - Theory and Practice of Object Systems*, vol. 4, pp. 207-225, 1998.

[12] I. Sommerville, *Software Engineering*. London ; Reading, Mass.: Addison-Wesley Pub. Co., 1982.

[13] D. Ungar and R. B. Smith, "Self: The Power of Simplicity," OOPSLA '87, 1987.

[14] World Wide Web Consortium, "Extensible Markup Language (XML) Specification: W3C Working Draft 21 Apr 1996," : http://www.w3.org/, 1999.