XGuide – Concurrent Web Engineering with Contracts

Clemens Kerer and Engin Kirda

Vienna University of Technology, Distributed Systems Group
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
{C.Kerer,E.Kirda}@infosys.tuwien.ac.at

Abstract. XGuide introduces the concept of contracts to the Web engineering domain. Contracts define the characteristics of Web components and pages. They serve as specifications for pages, enforce strict separation of concerns, are composable and enable concurrent implementation. As a result, time-to-market is shortened and support for maintenance and evolution scenarios is provided. The XGuide development process iteratively extends and refines conceptual models towards concrete implementations and thus bridges the gap between the conceptual and implementation space.

1 Introduction

Various methods were proposed to support Web engineering [1] processes—often influenced by and building on results from other fields of computer science. Existing approaches include database-centric, graph-based, logic-based, object-oriented and component-based methods. Most approaches share several key requirements and concepts. The ability to model a Web-based system on a conceptual level that is subsequently transformed into a concrete implementation is one. Dividing the system into smaller modules and components that can be reused and composed to form larger components is another. Additionally, the principle of *separation-of-concerns* is identified as being crucial and is supported to various degrees.

The goal of the XGuide approach is to propose precise specifications and design-by-contract [2] for Web pages and Web components (page fragments). We call these specifications *Web contracts*. Web contracts serve as the basis for the composition and integration of Web components into Web pages. Furthermore, they support the separation of design concerns by clearly specifying the concerns (and their dependencies) involved in the creation of a page or component. Such concerns include the content management, layout design and application logic programming. After the contract design, XGuide includes a step-wise refinement of these design artifacts towards a concrete implementation. Based on the concern and dependency specifications in the contracts, XGuide strives to reduce the overall time-to-market by supporting a concurrent implementation phase. A prototype development environment called *XSuite* supports and automates the XGuide method.

2 Contract-Based Development with XGuide

The XGuide development process is structured into seven phases: requirements analysis, feasibility decision, conceptual modeling and design, implementation, testing, deploy-

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 88–92, 2004.

ment, and evolution and maintenance. For space reasons, we focus on the design phase in this paper. A complete discussion of the XGuide method can be found in [3].

In the design phase, a conceptual model of the Web site is created. This model consists of pages, components, component references and application logic processes. To illustrate the approach, we use a snippet from the Vienna International Festival (VIF, `http://www.festwochen.at`) 2003 case study. The VIF is an annual cultural festival in Vienna featuring operas, theatre performances, concerts, exhibition and other cultural events. The VIF Web site not only offers information about the programme, venues, press releases, etc. but also provides a shopping cart application for ordering tickets online. Figure 1 displays a simplified version of the conceptual model for the shopping cart of the VIF Web site.

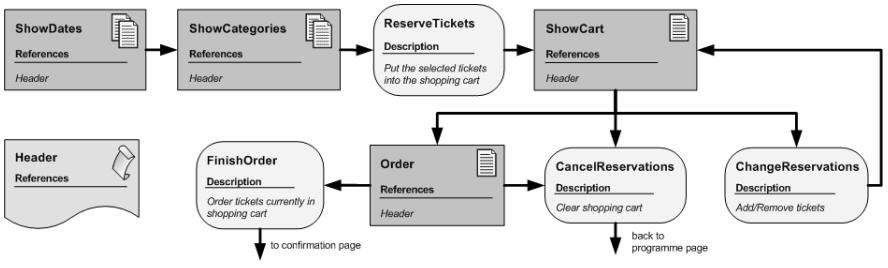

Fig. 1. A snippet from the component web for the VIF shopping cart application.

A *page* (depicted as a rectangle in the model) denotes a unit that will be presented to the user in her browser. From the user's point of view, all interactions are with pages; the internal structure of pages (i.e., its decomposition in components) is not visible. *Components* (depicted as rectangles with curved lower borders) represent page fragments that are reused on several pages. Typical examples for components are page headers, footers and global access structures such as hierarchical navigation menus. A component can be included in another component or page by referencing it in the *References* section of the component or page. For the purpose of this discussion, we only defined a single component *Header* in Figure 1 that is included in all pages. Finally, *application logic processes* (rectangles with rounded corners) represent the back-end processing (if any) that takes place on a server round-trip (e.g., when moving from one page to another by following a link or submitting a form).

So far, we did not distinguish between static and dynamically-generated pages or any information flow between them. The content of dynamic pages usually depends on some input values provided by the framework or the user. Customized navigation bars, for example, take a page identifier as input and render the currently viewed entry in the menu differently than the others. In our example, the *ShowDates* page requires an event identifier as input to determine which event's dates it should present. In XGuide, we denote the set of arguments a page requires to be created at the server side as the page's *input interface*.

After the input interfaces for all components and pages, we can define the corresponding *output interfaces*. An output interface is the opposite concept and describes what values a page, a component or an application logic process provides. Pages and components can provide values using embedded forms where the user can enter information and submit it. Since a page can contain multiple independent forms, a page can have multiple output interfaces as opposed to a single input interface. Similarly, application logic processes can offer multiple output interfaces based on values created, calculated or retrieved at runtime.

With the interfaces for all components and pages, the conceptual model (called *XGuide sitemap*) is complete. It serves as a basis for the detailed page and component specifications. The next section discusses Web contracts, their internal structuring into concerns, and how they can be composed.

3 Web Contracts and Contract Composition

A Web contract is a design-level specification of one or multiple Web pages or components that is structured into separate concerns. XGuide currently supports concerns for the content (i.e., information that is offered to the user), the graphical appearance (i.e., the layout – the formatting information with which the content is formatted for presentation), and the application logic (i.e., the functionality that is necessary for providing the dynamic interaction with users) of a page. Each concern has its own (XML-based) specification vocabulary; the contract itself acts as a container for the concerns.

```
<xcontract version="1.0">
 <concern docElement="webpage" type="Content"> <!-- XML schema --> </concern>
 <concern type="AppLogic"> <!-- XML interface definition --> </concern>

 <compositionreferences>
  <reference version="1.0" to="header.contract">
   <composition type="Content">
    <operator elementName="webpage" position="beginning" />
   </composition>
   <composition type="AppLogic">
    <in>
     <param-ref name="page_id">
      <operator type="omit" value="/webpage/@id" />
     </param-ref>
    </in>
   </composition>
  </reference>
 </compositionreferences>
</xcontract>
```

Fig. 2. The contract for the programme page of the VIF Web site.

The *content* concern specifies the structure and type of the page's content using an XML schema. This concern is used by the content manager to prepare the content for

the page and by the layout designer to apply style information to the content elements. The *application logic* concern, on the other hand, captures the input and output interface requirements of the page and is used by the programmer. The contract shown in Figure 2 defines the two concerns in lines 2 and 3 and would include an XML schema and an XML-based interface definition in the respective `<concern>` elements.

Contracts also form the foundation of XGuide's concurrent implementation phase. They capture all information necessary to implement the various aspects of the page independently of each other, i.e., the content manager, layout designer and programmer can work in parallel based on the information provided in the contract.

In the example in Figure 1, all pages reference the header component and thus their contracts need to be composed with the header component's contract. In Figure 2 the composition information is represented by the `<compositionreferences>` element (lines 5-18). Contract composition works on a per concern basis. Consequently, each concern defines its own composition operator.

3.1 Content Composition

Since the content concern uses an XML schema to define the structure and the data types used for content modeling, composing content concerns can rely on schema composition. To keep the operation simple, we currently restrict content composition to extending an existing element definition either by inserting a new element at the beginning or appending it at the end of an existing element's content model.

Figure 2 shows the programme page's contract that references the header component's contract. The content composition operator (lines 7-9) defines that the `webpage` element should be extended at the beginning with the referenced header.

3.2 Application Logic Composition

The composition of application logic concerns requires a more sophisticated composition operation. We distinguish between composition of input interfaces and composition of output interfaces. Since output interfaces represent Web forms that are embedded in a page, composition of forms is cumulative. This means that if we reference a component that contains a form, i.e., has an output interface, we add a new output interface to the referencing component.

For input interfaces, the situation is more complex. Since Web components do not exist on the client-side (because they are represented as HTML), the task is to create an input interface for the whole page (for page creation at the server side) that subsumes the input requirements of all components on the page. As a consequence, the composition operation does not deal with the whole input interface but defines a composition operator for every parameter in the input interface of a referenced component. We distinguish three cases when composing a component input value c_i of a component C with the input interface of a page P. If c_i is not related to any value in P's input interface, it is added to the new input interface (*composition-by-addition*). If the input value c_i happens to be already represented in P's input interface (because the page also needs this value as input), the composition operation does not need to modify the input interface but merely adds a mapping from the existing value to c_i (*composition-by-unification*).

Finally, c_i can be directly provided by the content of the embedding page P. In this case, c_i is mapped to an XPath expression identifying the value for c_i in the content of P (*composition-by-omission*). Figure 2 gives an example for this case (see lines 10-16). The value for the *page_id* input parameter of the header component is provided by evaluating the */webpage/@id* XPath expression on the embedding page. Thus though the header component requires a *page_id* input parameter, it does not appear in the page's input interface. For more information on the XGuide contract calculus see [3].

In addition to the currently supported content and application logic concerns, we are working on two additional concern plug-ins capturing the specification information for meta-data and access control to page content based on user identity. Each concern plug-in again consists of an XML language to express the specification information and the corresponding composition operators. Given the contracts for all pages, developers can start implementing the specified concerns. Because of the strict separation of concerns in the contracts, they can even do so in parallel and assemble the concern implementations at the end.

To support the development process, we provide a prototype tool called *XSuite* that supports the XGuide method. XSuite is an Eclipse plug-in and supports visual modeling via Microsoft Visio. It provides dialogs and wizards to create contracts for all artifacts in the XGuide sitemap, to define pages based on the created contracts, to implement these pages in the MyXML [4] implementation technology (other implementation technologies are supported via a plug-in mechanism), and to test and deploy the implemented pages.

4 Conclusion

This paper describes how we apply the concept of design-by-contract to the field of Web engineering and proposes the use of specifications (Web contracts). It demonstrates the use of contracts in the XGuide approach and shows how they influence the composition of Web components and facilitate a parallel implementation. A simple example from deploying XGuide in the VIF 2003 case study is shown. The strict separation of concerns adds additional benefits through an increased reuse potential and eased maintenance.

References

1. Ginige, A., Murugesan, S.: Web Engineering: An Introduction. IEEE Multimedia, Special Issue on Web Engineering **8** (March 2001) 14–18
2. Meyer, B.: Applying "Design By Contract". IEEE Computer **25** (1992) 40–51
3. Kerer, C.: XGuide - Concurrent Web Development with Contracts. PhD thesis, Technical University of Vienna (2003) `http://www.infosys.tuwien.ac.at/Staff/ck/downloads/xguide.pdf`.
4. Kerer, C., Kirda, E.: Layout, Content and Logic Separation in Web Engineering. In: 9th International World Wide Web Conference, 3rd Web Engineering Workshop, Amsterdam, Netherlands, May 2000. Number 2016 in Lecture Notes in Computer Science, Springer Verlag (2001) 135–147