Engineering Web Applications with Fine- and Large-Grained Web Components

Hans Albrecht Schmid
University of Applied Sciences Konstanz
Brauneggerstr. 55
D - 78462 Konstanz
xx49-07531-206631
schmidha@fh-konstanz.de

Bertram Dieterich
University of Applied Sciences Konstanz
Brauneggerstr. 55
D - 78462 Konstanz
xx49-07531-43390
bertram@fh-konstanz

ABSTRACT
Web engineering concepts and tools have often not kept step with the dramatic increase in the power and complexity of Web applications. This paper focuses on problems caused by the introduction of business processes and their integration with navigation in Web applications. A phase-lag mismatch between the HTTP-oriented structure of the Web support software and the logical structure of processes and activities makes, besides other reasons, the development quite complex, and composition and reuse difficult, if not impossible.

As a remedy, we propose to use modern component technology on the J2EE Web tier to supply reusable components, which follow the logical process and activity structure. WACoF activity components are provided as fine-grained components isomorphic to business processes and activities. Whereas WebCoF Web components are customizable off-the-shelf large-grained components with cross-domain or domain-specific functionality, which integrate activity components and business components. A Web application is built by composing Web components in a configuration program.

Categories and Subject Descriptors
D.2.2 [Design Tools and Techniques]: Object-Oriented res. Component-Oriented Design Methods

General Terms
Design, Languages

Keywords
Components, Business Processes, Web Applications, Component-Oriented Web Engineering, Web tier

1. INTRODUCTION
The power and complexity of Web applications has dramatically increased during the last few years. “The Web is increasingly doing things that we’ve typically seen in enterprise applications” [6]: in combination with navigation, it is used to execute business processes, leading a customer through a set of related activities in the B2C (Business to Customer) area; and embodying often sophisticated interaction patterns among different participants in the B2B (Business to Business) area.

However, the Web engineering concepts and techniques used to build Web applications have not kept step. On the modeling and conceptual design level, a document-centric and navigation-oriented approach (compare e.g. [17] [2]) is often applied also to business process design. Resulting problems and a solution to them are presented in [16].

On the concrete design and implementation level, the situation is similar. Though Java has encapsulated the low-level HTTP processing and character stream handling in a very elegant and efficient way (in objects like servlets and Java ServerPages), it has preserved the basic HTTP structures. Since there is, as we will show, a phase-lag mismatch between HTTP-based support structures and the logical process and activity structure, the former do not conform well to the use of modern software engineering technologies. “Many current web technologies lend themselves to - or even encourage - bad practices.” [8].

As a remedy, different frameworks like Struts [4], Expresso [11], and others, all based on the Java model-2 architecture or a similar one like the revised MVC architecture [8], have been introduced. But our experience is that MVC-based frameworks are not too well suited to realize business processes and activities, since they let the HTTP structure shine through (as we show in section 3) and do not allow for a composition of business activities and processes. In addition, you have to program “by hand” each navigation step when you integrate navigation.

As a solution to many of these problems, we propose to introduce business activities and processes as first-class citizens in Web engineering, both on the modeling and conceptual design level (see [16]), and on the concrete design and implementation level, which is the topic of our paper. We use a modern component technology [18] to build Web applications on the base of reusable components, which are isomorphic to the logical process and activity structure.

The WACoF Web Activity Component Framework provides fine-grained activity components on the J2EE web tier. On top of those, the WebCoF Web application Component Framework integrates activity components and business components in large-grained Web components with cross-domain or domain-specific functionality. A Web application is composed from Web components in a Java configuration program.

Thus, you may follow a seamless approach, beginning with the modeling of modern Web applications with embedded business
processes, over their conceptual and concrete design to their implementation with components.

The structure of the paper is as follows: section 2 analyzes the characteristics of business processes and activities and compares them to those of servlets or ASPs. Section 3 gives an overview on related frameworks and their suitability to realize business processes. We introduce in section 4 activity components that are oriented towards the logical structure of business processes and activities. Section 5 shows how to compose activity components to more powerful and reusable units, as e.g. business processes with different kinds of control flow. Our experiences in building Web applications with them are described in section 6. Section 7 presents large-grained Web components with cross-domain or domain-specific functionality that integrate several activity and business components, and section 8 shows how to build Web applications from them.

2. WEB ENGINEERING OF PROCESSES

In many Web applications a user has to carry out a business process, that is a predefined set of activities in a predefined sequence. We should speak more accurately of a "Geschäftsverfahren" (German), business procedure [13], or process thread [1]. Its definition is that it is performed by one person in one place at one time; it is triggered by an external event and produces some result, which adds measurable value to the business. In contrast, a business process often involves multiple users and multiple activities that take place over different time-slices [1]. But since the terms "business procedure" or "process thread" are not widely accepted, we will use the term business process in the following when no confusion can arise.

In this section, we consider the characteristics of activities and business processes and compare them to those of servlets.

2.1 Business Process Analysis

As an running example, we use a business process providing an offer for a car policy, invoked from the Web portal of an insurance company. The business process, which offers you a car insurance policy with rates specific for the car type, make and owner characteristics, leads you through the set of activities shown in the UML use case diagram, Figure 1 left. First, you determine or enter the car type identification number; then, the code of your residence area; the risks covered by the policy; and the payment options. With all the information entered correctly, the cost of the policy is calculated and you get a policy offering.

![Figure 1: Use case/activity diagram for car policy offering](image)

Figure 1: Use case/activity diagram for car policy offering

The UML activity diagram, Figure 1 right, shows that the car policy offering process carries out the described activities in a two-way sequence. It does not terminate an activity when you go on to the next activity, but suspends it. Similar to a wizard process, it allows you by pressing an application-provided back-button to go back to and resume suspended activities, change their input, and advance again. In that way, you may try out which influences changed conditions have on the cost of the policy.

A business process like car policy offering drives the user through its activities: it defines the set of activities to be executed, and the control flow among them.

2.2 Phase-Lag Mismatch

When we analyze what happens in an activity like car type identification (see small box middle left in Figure 2), we see that it presents first an output page with an HTML form, which may contain previously entered car type data. After the user has entered new car type data or edited the displayed data and sent the HTML form off, the activity gets the data, processes them, and checks if the car type data are entered correctly. If so, the activity will complete its processing and the activity to be performed next is started. Otherwise, the activity repeats the output-input cycle until the entered data are correct and consistent, or the user breaks off and cancels the processing.

As the example shows, an activity follows usually the output-input scheme; it may perform one or several output-input cycles. In contrast, servlets, ASPs and similar software follows the HTTP request-response scheme. When a servlet is activated (by a call to its doGet- or doPost-method), it gets the request that contains the HTML form parameters of the input Web page. It processes this input and then produces an output page as a response.

![Figure 2: Phase-lag mismatch: activities and servlets](image)

Figure 2: Phase-lag mismatch: activities and servlets

Consequently, the logical activity structure and the physical servlet structure do not match. Figure 2 shows the mismatch among two activities from the "policy offering" business process, and the servlets (drawn as dashed boxes) implementing them. There is a phase-lag mismatch between activities and servlets: a servlet includes the second half of the preceding activity, the decision making which activity to execute next, and the first half of the succeeding activity. Servlets are not well adapted to the logical application structure, but conform well to efficiency requirements since a servlet is not active during the thinking time of the user.

2.3 Global Business Objects

With servlets, references to business objects are often kept as a kind of global variables in the session or a similar context. One
reason is the phase-lag mismatch, which often makes it necessary that different servlets access the same business object.

From a software engineering standpoint, the reference to a business object like cartypes should be local to and encapsulated in the activity using the object (see Figure 2).

2.4 Software Engineering Problems

Building Web applications with business processes directly on the client tier, activity components on the Web tier, and business objects/components on the business tier

consider the (Expresso) state machine model (see Figure 3) of the policy offering process.

Figure 3: Expresso State machine model of modified car policy offering process.

Though one may deal with these and related problems by factoring the processing out, one should better introduce an abstract representation which avoids that and other problems from the beginning. Another problem with controller-based architectures is that activities do not exist neither as concepts nor as software deliverables, and thus cannot be composed to larger units.

In general, no other component-oriented approaches on the Web tier are known to the author.

Several Web engineering workshops like that held at the ICSE give an good overview over different areas of Web engineering. The IEEE Software issue “Engineering Internet Software” addresses specific issues regarding the software engineering of Web applications.

4. ACTIVITY COMPONENTS

To provide a base for Web application engineering that does not cause the described problems, we introduce fine-grained activity components with the Web Activity Component Framework WACoF. The WACoF architecture is partially derived from that of the business process component framework BPCF for business applications.

Consider the (Expresso) state machine model (see Figure 3) of the policy offering, with the control flow changed in the way that in the second process step, either the covered risks or the residence area selection activity is to be executed. A state corresponds to the point of time a Web page is presented to the user. A transition represents the reaction on a user event. A transition processes the request, i.e. the user input made at the last state, and provides as response the Web page corresponding to the new state.

If several transitions emanate from a state like cartype identification, the problem is that each of these transitions has to process the input related to the cartype identification page. If it depends on the input made on a Web page (and not on the button pressed, i.e. user event) one of several transitions is selected, that can only be handled by introducing a single, invisible successor state which makes the selection among these transitions. Moreover, if two states, like covered risks or residence area selection, have a common successor state like payment options, the processing required to present the payment options page has to be done with each of these two transitions.

Figure 4: Overall architecture with JSP pages on the client tier, activity components on the Web tier, and business objects/components on the business tier

The WACoF overall architecture (see Figure 4) follows the J2EE architecture and a generalized MVC architecture with regard to the separation of responsibilities, similar to the revised MVC. But WACoF provides activity components instead of...
controllers on the J2EE Web tier, with associated JSP pages on the J2EE client tier. Activity components collaborate with business objects or components on the J2EE business tier.

Activity components on the J2EE Web tier are responsible for providing the output Web pages and processing of user input, performing the application logic, and controlling the processing flow among the activities of a business process. Activity components embody and encapsulate the basic "mechanics" that transform HTTP-oriented structures and operations into structures and operations that conform to modern software-engineering principles. Therefore, activity components are programmed by "action"-oriented interfaces containing e.g. a start and several buttonPressed operations.

4.1 Phase-Lag Transformation

An activity component has, instead of a doGet/doPost operation, an incoming ActivityIF interface that defines activity-related operations following the output-input scheme: after a start operation, a (possibly repeated) cycle consisting of a getOutputPage operation followed by a processInput operation, is called. The operation getOutputPage gets the Web page (area) to be presented to the user, and processInput returns the user-input HTML form parameters for processing.

![Figure 5: Servlet and JSP servlet transforming doPost calls into ActivityIF operations calls](image)

A root servlet component, which is (derived from) a servlet, compensates for the phase-lag between a servlet and an activity by transforming the operation calls. It transforms an incoming doPost (or doGet) method call from an application server into two calls of ActivityIF operations (consider the root servlet and activity instances of the UML sequence diagram, Figure 5): it calls in the first doPost-call, after the initialization with start, the getOutputPage operation; and in the next doPost-call, the processInput operation. Then, the root servlet component calls the getOutputPage operation of the next output-input cycle, etc.

4.2 JSP Output Page

An activity component might produce its output page in the form of a HTTP response character stream. However, it is more attractive and more common in real-world applications to use Java Server Pages (JSP). The advantage of JSP pages is that a graphics designer designs a fixed HTML page, and a Web programmer provides its dynamical content by inserting, within JSP tags, Java code that gets the content data from business objects or components. The Java code is processed before the output page is sent off to the browser.

Therefore, activity components provide the output page in the form of an associated JSP page. This causes a change of the control flow: an activity does not return a JSP page as a result of getOutputPage, since a JSP page is produced by a special JSP page servlet. As a consequence, the root servlet hosting the activity components must pass control (by a JSP-forward command) to the JSP Servlet (see Figure 5). The JSP Servlet calls the operation getOutputPage of the currently active activity (stored in the servlet request environment) to get the JSP output page to be displayed (see Figure 5).

4.3 Reuse of Activity Standard Behavior

A Web application developer does not need to be aware of the described control flow mechanics among an activity component, the root servlet component, the JSP servlet etc. All common behavior is provided by the component framework. This includes, besides the control flow mechanics, the user input analysis. For example, the activities of the car policy offering process have a next-, back- and cancel button. When an activity would process the user input directly, it would have to analyze the form parameters of the response parameter in its processInput method to find out which button was pressed. This standard behavior is programmed repeatedly with many state-of-the-art Web applications.

![Figure 6: Activity component with user-event adapter](image)

For reuse of standard behavior, we provide an user event adapter for an activity component (see Figure 6). It processes the operation processInput, defined in its incoming interface, analyzes the parameters of the HTML form to find out which button was pressed or which link clicked, and calls the matching hook operation of its outgoing UserEvent (and ChildActivityListener) interface. The UserEvent interface defines hook methods for all standard user events with operations like nextPressed, backPressed and cancelPressed.

As a consequence, a developer of an application activity component implements (see Figure 7 for car type identification) - in the start method the initializations required, like creating or accessing business objects or retrieving information from a database, for display on the output page.

- in the hook methods of the UserEvent interface the reactions on user events returned from the browser, like on the buttons possibly pressed or links clicked at.
Composed Activity Component as defined by the UML diagrams of component structure of a business process is isomorphic to that of its high-level model. An important characteristic of our approach is that the sequence of a composed activity component has been provided, several children may be started and active at a time. A parent regains control when a child notifies the parent about the termination of its execution. This means that maximally one child is active at a given time. A single-active-child composed activity may activate one child after another one. Thus, the parent activity defines the sequence (the control flow) in which its child activities are executed, and may delegate work to them. In infrequent cases, for which a special multiple-active-child composed activity component has been provided, several children may be started and active at a time.

We may implement a business process as a composed activity component that has subprocesses or activities as children. For example, the car policy offering process is implemented by a car policy composed activity that is composed with the child activity components car type identification, residence area selection, and so on (see Figure 8). It controls the processing flow among the child activities so that they may be executed in a two-way sequence.

From a component framework point of view, a composed activity component has two child-related interfaces, which a basic activity component does not have. It invokes via the ActivityIF as an outgoing interface the services of child activities, and receives callback events like childActivityCommitDone via the incoming ChildActivityListener event interface from the child (see Figure 9, left). We extend the UML interface notation by double-lined arrows for event interfaces.

From a supporting class framework point of view (see Figure 9, right), the Activity and Composed Activity class play a role in two patterns. Together with the ActivityIF interface, they form a composite pattern [5]. Following this pattern, a parent component forwards calls to the ActivityIF of its child components. Together with the ChildActivityListener interface, Activity and Composed Activity form an Observer pattern in the Java-specific listener-form. Following that pattern, a child component notifies its parent component about ChildActivityListener events like childActivityCommitDone.
An activity is responsible for processing the input in the associated JSP page area. This is handled by the user event adapter of a composed activity: it checks if some user input has been made on the associated page area. If so, it calls a hook-method of the UserEvent interface. Otherwise, it forwards the processInput call to its active child (see Figure 9 left: arrow from UserEventAdapter to outgoing ActivityIF).

### 5.3 Suspending Activity Execution

We allow to suspend and resume a business activity and a process for different reasons: In a wizard-like process, you may leave an activity, going to the previous or subsequent one, and return later to it in the state, in which you left it. Or in a business process like the check-out process of a Web shop, you may want to carry out some navigation to get more information on the items you are going to order. Or you may want to execute several processes in parallel.

An activity may finish its processing for several reasons:
- ✓ it may terminate if its work is completed,
- ✓ it may be aborted if the user cancels it,
- ✓ it may be suspended due to a user input.

Each activity has a state. The state of an activity indicates the current point of control and the current control situation (it may be active or suspended) and includes the entities that the activity uses. The state of a parent activity contains information on the child activities, which are currently active and on those previously executed and terminated or suspended.

When an activity is suspended, it conserves its state, and returns control to the parent activity. This keeps the child as suspended in its state. When a parent activity resumes a child activity, the child activity just goes on with its execution at the suspended state.

### 5.4 Composing Business Processes

You may use composed activities to realize and implement business processes with different kinds of control flow among the child activities.

A one-way sequential composed activity executes a set of child activities in a fixed sequence from the first to the last child (see UML activity diagram, Figure 11 a). A simplified car policy offering process without back buttons would be an example. Each child activity is completed before the next one is started.

A wizard-like (or two-way sequential) business process, like a car policy offering as described in section 2, executes a set of activities in a fixed sequence, but a user may go from an activity either to the next or to the previous one (see Figure 11 b). A child activity is suspended when a user leaves it by pressing a next- or back-button, but not completed. Thus, the data entered are saved so that a user has not to re-enter the data if he goes later back to the child activity and resumes it.

### 5.5 Summary

Activity components on the J2EE Web tier collaborate with business objects from the J2EE business tier, though we did not relate to the business objects in our discussion and examples.

With composed activity components, we introduce on the J2EE Web tier implicitly two different granularities of components [9]. When we reuse a composed activity like a car policy offering, we reuse implicitly its complete activity subtree. With large-grained Web comps, the subtree will be reused explicitly.

On a first glance, the nested execution of activities may seem to be the same as the nesting of procedures in a procedure call hierarchy. But the difference is that activities are first-class objects [13], have a state that may be conserved and have nested page areas associated.

### 6. APPLICATION EXPERIENCES

After the completion of the first version of the WACoF activity component and class framework, we have used it very successfully in university projects and in cooperation with the industry, to build different Web applications. Our estimate is that about half the development time was required compared to the direct use of servlets and JSP. E.g., in the CRe@M project (described below) the activity components (incl. JSPs) were implemented by three (part-time) persons in each ½ person-weeks, after the user requirements, in particular the user-friendly administration of a hierarchical technical installation had been clarified. Another project, a co-travelling electronic agency was
done by two different groups, one using WACoF, the other one Expresso. The one using WACoF required about half the time of the other group, though that may be attributable also to other influence factors.

6.1 CRe@M
CRe@M (Customer Relationship Management) is a software for customer relationship management for small enterprises and customers with a technical installation, which won a technology transfer price at the Intertech technology and trade fair in St.Gallen, Switzerland, November 2001. Its main strength is the flexible and user-friendly description of a technical installation that consists of a hierarchy of technical components, with all repair and maintenance actions. Our first concrete application area was customer relationship management for small companies building, repairing and maintaining heating installations.

![Figure 12: CRe@M activity component tree](image)

The CRe@M home page provides a user-friendly customer search with different search criteria like name or place of residence. Once you have selected a customer, you may carry out different business processes. The activity tree (see Figure 12) reflects that structure: The home page as root of the tree is realized by the "landmarks" and the child "customer list" composed activity. It has as grandchildren composed activities like "new customer", "customer details", "edit skeleton", etc. that represent these business processes.

6.2 Experience Summary
Four different aspects contribute to the very successful use of WACoF:

- The activity component structure is isomorphic to the structure of the business process and activities. This makes its design, understanding and implementation fast and easy.

- Well-defined and clear interfaces define the boundaries between the different components so that a separate implementation and fast integration nearly without problems is possible.

- Standard processing is embodied in the activity component class framework so that it does not need to be programmed repeatedly.

- You may reuse activity components of different granularities, simple ones and composed ones, together with the associated page areas (which may be exchanged when required). For example, in CRe@M we could re-use the complex composed activities and the JSP pages (after a small modification) related to the hierarchical technical installation in five different business processes, namely building a installation skeleton, modifying a skeleton, building an installation from scratch, building an installation from a skeleton, and modifying the installation structure and description after technical changes.

7. LARGE-GRAINED COMPONENTS
When we developed different Web applications with WACoF, we made the experience that we implemented several times similar functionality in a similar way. To be able to do reuse on a larger scale, we introduced large-grained Web components with the Web Component Framework WebCoF.

A Web component consists of one or several activity and business components, spanning itself over two J2EE tiers, the Web tier and the business tier (see Figure 13). Together with the associated JSP page, it spans over three tiers. For example, a search component embodies:

- in activity components on the Web tier the control of the processing flow among the different search steps and the input/output and application logic required in each search step,

- in EJB business components on the business tier the querying of business data bases,

- in associated JSP pages on the client tier the search masks, index display masks, etc.

The current WebCoF version uses entity and session Enterprise JavaBeans (EJBs) as business components. Session beans are used, in particular, for performance reasons, when a set of entities is handled and little information about each entity is required.

A Web component has an activity component (see Activity Component 1 in Figure 13) as a facade [5], via which its execution is started. On the other hand, if a Web component
delegates work to another Web component, it does that also via an activity component, which is a kind of reverse facade (see Activity Component 2 in Figure 13) funneling the outgoing calls. Thus, a Web component provides:

- towards its parent, the incoming ActivityIF and outgoing ChildActivityListener interface as "parent" ports;
- if it has children, towards them, reversely, the outgoing ActivityIF and incoming ChildActivityListener interface as "child" ports.

A Web component is customized via operations of its customization interface, which is associated to the ActivityIF of the parent ports. Its facade activity component passes the customization parameters to the other internal components as required.

7.1 WebCoF Web Component Framework

The WebCoF component framework provides a number of "off-the-shelf" reusable cross-domain Web components: structural components allow to structure a Web application by providing e.g. composite or portal Web pages or landmarks; cross-domain functionality components embody e.g. different kinds of searches and navigation. We provide so far no domain-specific business process components, since they are more individual and less reusable. But an organization should implement its specific processes in the form of Web components for reuse in different Web applications.

You build a Web application as a tree of Web components, similarly as you build a tree of fine-grained activity components with WACoF. Two Web components are connected by plugging the parent ports of the child component in the child ports of the parent component. Behind the cover, the facade activity of the child Web component is plugged into the reverse facade activity of the parent Web component.

In the following sections, we will give an overview over a few structural and cross-domain functionality components.

7.2 Structural Components

Structural components like landmarks, composite pages and portal pages are used to structure Web applications.

Let us describe a landmark component in some detail. You customize it with its child Web components, to which links are displayed and provides them for access by the JSP page. If a user clicks at a landmark, the landmark component terminates the execution of the currently processed child Web component and starts the execution of the selected Web component.

A landmark Web component consists of a landmark composed activity component, which forms both the facade and the reverse facade of the Web component. Only 150 lines of code, mainly in the linkClicked hook method, sufficed to develop a landmark component from a composed activity component (with an user event adapter).

7.3 Navigation Components

WebCoF provides navigation components, one for fixed page navigation and another one for dynamic page navigation, so that navigation does not need to be programmed in specific activities of Web components. You provide only the pages over which a user may navigate. They are normal HTML, XML or JSP pages that are stored together in files or subdirectories of a directory, which represents a navigation space. The JSP file of a dynamic Web page contains a skeleton of information, which needs to be complemented with information contained in the object or set of objects to be displayed. The links on these pages are in a special format:

A static link, which gives a reference to a Web page, consists of a URI to the servlet that is running the Web application, and a parameter that gives the path from the directory root to the selected page.

A dynamic links consists of a page reference, like a static link, and additional parameters that describe which object(s) are to be displayed, giving the object class and the instance or list of instances. Since we work with EJB entity beans, the object class is given by the JNDI name and the EJB home name; the instance is given by the finder method name, and the object identification to be supplied as a parameter of the finder method.

We use a component also for fixed page navigation though we could use just the browser. The reason is that the Web application should not loose control during navigation. In addition, we integrate in this way navigation better in the application, e.g., when landmarks are to be shown on top of all Web pages during navigation.

A Web application may contain several navigation components with differing navigation spaces if required. A navigation component is customized with the directory name of its navigation space, and with the name of a default Web page that is displayed first. At invocation, a parameter with the Web page to be displayed first may be passed, overwriting the default page.

Both kind of navigation components consist of one navigation activity component, which analyzes (in the linkClicked method) the link and returns the page to be displayed. The dynamic navigation activity finds, in addition, the entity bean or beans to be displayed and provides them for access by the JSP page.

7.4 Search Components

WebCoF currently provides three off-the-shelf search components for databases search: taxonomic search, filter search, and guided filter search. They differ with regard to the knowledge a user must have in advance about the objects he is looking for, and if he must type in values or may just click at values offered.

Taxonomic search shows navigational characteristics: a user does not need to enter values, but can select an item from a (sub-) category list, in which a previously selected category item has been refined. Several levels of a classification hierarchy may be stepped through before eventually objects are associated to a subcategory. Criteria-based filter search is like query by example, where a user enters values for given different search criteria. Keyword-based filter search is similar except for the user entering keywords. Guided filter search is similar both to taxonomic and filter search in combining the characteristics of filter search with those of navigation.

General characteristics of search components

The WebCoF search components are user-friendly processes with a common structure and a control flow that allows query refinement, iteration through found items, or item reselection.
from the index list. In contrast, many Web applications offer only a simple form of search without these features.

![Diagram](image)

**Figure 14: Filter search component with search activity, search session bean and embedded index component**

All search components are composed in the same way from large- and fine-grained components (see Figure 14): A search activity component, which is the facade [5], has as child a large-grained index component, which has an index activity as facade. The search activity uses the services of a fine-grained search session bean to retrieve the result items from the database, it puts them in a result object, and it collaborates with the index component to display the index and result items on a Web page if required. The index component has a fine-grained display item activity component as a child.

To avoid the well-known “index creation from persistence layer entity instances” performance problem, we use a search session bean that searches the index items directly in the database so that an optimal search performance is achieved. Only when an item is selected from the index, the search component creates an entity bean for further processing.

For each kind of search, we provide a special search session bean and a special search activity component, which inherits the common responsibilities from a search activity base class in the activity class framework. All other components and objects are the same in the different kinds of search components.

You customize a search component with the following general customization options:

- The control flow option allows or disallows query refinement, index item iteration etc.

- The input mode option controls if the search component presents a Web page with a search mask and fetches from it the user input, or if the search input values are passed as a parameter. For example, the input for the bestseller index on the home page is not entered by the user, but passed from the home page component as a parameter.

- The output mode option controls if the search result is presented from the search component on an index page, or if it is not presented, but may be obtained via the getResult operation.

- The Display Item activity displays an item selected from the index.

In addition to these general customization options, there are specific options for the specific kinds of search.

**Filter search component**

The filter search bean of a filter search component retrieves the result items from one database table, which contains as columns all attributes which are search criteria or presented as attributes of the result items. You may select freely table and column names as customization parameters of the filter search component. Its facade, the filter search activity component, passes the database-related customization parameters to the filter search session bean, which configures itself for the subsequent search operations.

The fine-grained filter search activity component controls the processing with regard to displaying and input processing of Web pages, passing the search input values to the filter search session bean, and processing the result list.

**Index component**

An index is a large-grained component, which is used as a sub-component by a search component. It may be used also independently to show a fixed, predefined list of items. It contains an index activity component which embodies the iteration through found items or item reselection from the index list, and starts the Display Item activity when an item from the index is selected.

8. **BUILDING A WEB APPLICATION**

We use a Web bookshop as an example to present a seamless approach to the engineering of a Web application that covers the conceptual design, concrete design, and the implementation. The bookshop has a home page with landmarks and different areas, from which a user may

- start a search for a book
- navigate through a taxonomic hierarchy to find books
- select a book from a list of bestsellers
- look at the shopping cart
- start the check-out process.

When you have found and selected a book, you may navigate from it to its author, etc.

8.1 **Navigation and Process Design**

Since conceptual and navigation design is not the subject of this paper and presented in [16], we just show the navigation and process design (see Figure 15) in form of a UML state diagram, which shows the interaction states of the Web shop. Processes like filter search and check out form a compound state, which has the interaction states of the activities that form the process as substates.
8.2 Web Component Selection and Design

When you analyze the navigation and process design diagram, you find out that you may build the bookshop to a large part from off-the-shelf Web components (see Figure 16). The home page, built from a combined landmark and composite page component, forms the root of the Web component tree. Its child Web components are:

- a filter book search; it is a filter search component with a search mask, which is customized for a user entering the author, title, etc., as search criteria, and with regard to the relational tables and columns (containing the book information) to be searched.

- a book classification; it is a taxonomic search component, which is customized to display the multilevel book-classification hierarchy, by passing a kind of parts-explosion relational table for book-classification, and the books relational table on a lowest classification level, as customization parameter. The advantage of using that component is that the classification hierarchy may be modified just by modifying the database content without reprogramming or re-customizing the Web component and without providing new JSP Web pages.

- the bestsellers; it is a filter search component, which is customized with parameters to work with fixed search criteria for bestsellers on the book relational table. The search, started implicitly by the composite page component, finds the bestsellers by searching for the ten books with the highest sales numbers, and presents them in an index.

- a custom-developed shopping-cart component

- a custom-developed check-out process component.

A different instance of the dynamic navigation component is attached as a child each to both filter search components, the taxonomic search component and the shopping-cart component. Each instance of the dynamic navigation component is initialized from its parent component with the book to be displayed, and allows navigation from a book to its author, etc.

8.3 Web Application Configuration

When the component design of a Web application is completed, you compose it from standard components and individually developed components by "glue-ing" them together with Java configuration code, as part of an initialization method of the root servlet of the application. This code creates the component instances, customizes them as required, and builds the application tree. For an application the size of the Web shop example, the configuration code has the length of about one to two Java source pages.

The Java compiler performs during the compilation of the configuration program the type checking: the matching of the outgoing and incoming interfaces of the children and parent ports, and the checking if the customization operations are called correctly.

Writing the Java configuration code is no programming in the sense of GUI or application programming; with a good knowledge of the component customization operations it is quite easy. However, it requires programming knowledge. Therefore, our objective is to get rid of programming and replace in the next WebCoF version the Java program glue by a different kind of glue, the natural candidate being XML.

8.4 Application Experiences

After the completion of the first version of the WebCoF framework, we have used it very successfully in university projects and in cooperation with the industry, to build prototypes of different Web applications: a bookshop somewhat more complex than the described one, two real-world shops, a wine shop and a milk product shop, and another version of the co-travelling electronic agency. We could reuse the off-the-shelf Web components intensively in these Web applications, so that only the customer-specific business processes had to be developed individually. Web components are built easily and with a relatively small programming effort, since they use activity components as a base. Thus, the construction of the prototypes was very fast.

9. CONCLUSIONS

We have shown that a modern software technology, component-oriented programming, may be used very successfully for the Web engineering of complex Web applications, which integrate navigation, search and business processes. The proposed approach allows a seamless Web engineering extending from the conceptual design over the concrete design to the implementation.
On the J2EE web tier, the WACoF component framework provides fine-grained activity components which
- hide the basic HTTP-oriented structures that shine through servlets and many Java model-2 based frameworks
- are isomorphic to the activity and process structure, as defined by a high-level model (e.g. UML use case and activity diagrams)
- provide for the reuse of Web application standard behavior
- allow for component reuse and composition in a tree structure. A tree lends itself to realize Web application structures like landmarks, or business processes.

The component and associated class framework WACoF was successfully used to implement several small to medium-sized Web applications. Developers with a knowledge of object-orientation, but no previous knowledge of servlets, JSP, and related aspects could implement a complex Web application like CRE@M in a short time.

The WebCoF Web component framework with "off-the-shelve" cross-domain Web components is built on top of WACoF. A large-grained Web component consists of one or several activity and business components, spanning itself over two J2EE tiers, the Web tier and the business tier.

With Web components, you may follow a seamless component-oriented approach to Web application engineering from the conceptual design, over the concrete design, to the implementation. Our first experiences with the approach are very favorable.

10. REFERENCES