Simplifying a Web Application’s Architecture – the DaVinci Framework

A. Langegger, J. Palkoska, and R. Wagner

Abstract. The World Wide Web has undergone a rapid transition from the originally static hypertext to an ubiquitous hypermedia system. Today, the web is often used as a basis for distributed applications. That’s why many research work has already been done about the modeling and engineering process for web applications. Many of the published concepts try to merge hypertext modeling with traditional techniques from the software engineering domain. Unfortunately, those concepts which capture all facets of the web’s architecture become rather bulky and are eventually not applicable for serious development. The missing adoption in industry indicates this. It seems that web engineering based on the blank architecture of the WWW is no promising approach. Moreover, there is a need for architectural frameworks. Such a framework should address both, the modeling process and the implementation.

DaVinci1 is the prototype of a proposed framework. It is a modeling framework as well as a framework supporting the implementation of web applications. For the modeling process UML is used. The architecture is based on the Model View Controller type 2 pattern. The major features of DaVinci are the hierarchical GUI tree and a special URL scheme to facilitate the interaction process.

1. Introduction

In 1989, when Tim Berners-Lee and his colleagues were working on the first drafts for a distributed hypertext system, they probably had no idea what the World Wide web would look like two decades later. While the number of registered domain names and accessible resources on the web was increasing rapidly, also the variety of programming languages and technologies for web page generation grew. Additionally, the latter one provoked an increase in the range of possibilities and applications on the web. Today the WWW is not merely an huge information system consisting of uncountable documents and other kinds of media, it can also host distributed applications providing concurrent access to users all around the world. Where browsers are acting as platform-neutral containers for the applications’ graphical user interfaces, the Hypertext Transfer Protocol provides a simple but effective communication layer. Moreover the web is becoming more and more ubiquitous and is already accessible from various kinds of mobile devices.

Another noticable fact is, that on the one hand hypermedia resources where more and more asking for application logic and on the other hand, traditional software approached the hypermedia domain

1DaVinci has been released for PHP under the GPL at SourceForge: http://phpdavinci.sourceforge.net. In the meantime an extensible Content Management System and several database-driven websites have been developed with DaVinci.
to benefit by its features mentioned before. For example, today’s business software products often integrate a web-based front-end to the main application server (e.g. mySAP Enterprise Portal or Oracle PeopleSoft’s Supply Chain Management software).

2. Related work – modeling for web applications

With the growing importance of web-based applications, the relevance of decent modeling techniques for such applications increased too. Having sound models can also provide the possibility to build CASE tools for (semi-)automatic generation of web applications [7], [13], [5]. In fact there exist numerous scientific publications, a journal\(^2\) and also some books (e.g. [12], [17], and [4] about Web Engineering and modeling for web applications. Many scientific concepts try to combine already existing models for hypermedia design (that is for example the Dexter model [9], HDM [8] or RMM [11]) with traditional Software Engineering techniques (where ER-models and UML are leading concepts). For instance OOHDM [15] and Baumeister et al. [1] try to combine hypermedia modeling with object-oriented software design methods. By contrast, the concept proposed by Jim Conallen [4] strongly originates from the Software Engineering domain and tries to cover the whole modeling process for web applications with UML and a proper Web Application Extension. Finally, the Web Modeling Language (WebML\(^3\)) [2] rather descends from the hypermedia modeling domain and provides a quite extensive common set of patterns and models for database-driven websites.

Beyond the bottom line some of the existing concepts seem to be rather complete and applicable to a wide range of different scenarios (especially WebML and Jim Conallen’s concepts). Except WebML, scientific concepts hardly provide development frameworks (common architecture and APIs) or even CASE tools which integrate the investigated concepts in practice. The founders of WebML, Piero Fraternali and Stefano Ceri, developed a tool\(^4\) which seems to be rather powerful when building database-driven web applications. It provides a CASE tool for building sites based on fundamental components (entities, relationships, pages, layouts, etc.) at a very high abstraction layer. The aim of the DaVinci project explicitly was to provide a modeling and development framework for web applications for developers. This means, that the source code is still managed by the developers themselves and CASE tools should only help where they are suitable.

2.1. A definition of terms

Since there exist quite divergent definitions for the terms web application and framework for web applications, a short definition has to be done first.

Where for some people interaction is the only difference between a website and a web application [10], a more severely definition is probably given by Finkelstein et al. in [6]. Their definition has also been adopted for this work. In our context, a web application is not just a set of web pages, rather it’s a peace of software with application logic and state. "[...] this implies that it enforces the notion of a session, which differentiates it from the ordinary request-response web paradigm [6, Introduction]." Note that, according to this definition a search engine’s front-end and even a webservice are not denoted to be web applications. These are just web-based services, but no applications.

Concerning the term framework for web applications, a quite appropriate definition can be found

\(^2\)Journal of Web Engineering, see http://www.rintonpress.com/journals/jweonline.html
\(^3\)see http://www.webml.org
\(^4\)WebRatio, see http://www.webratio.com
in [16]: "A Web-tier application framework [...] providing common application functionality such as dispatching requests, invoking model methods, and selecting and assembling views. Framework classes and interfaces are structural; they are like the load-bearing elements of a building, forming the application’s underpinnings." Thus, the framework must at least define a common scheme for user interaction and the displaying of content and navigational widgets (the GUI). With the generalisation of the interaction and view scheme the developer can concentrate on the actual application and doesn’t need to model and implement the fundamental functionality. A framework for web application should also provide common data management functionality (e.g. JDO, EJB when using Java) and could provide rights management within the session handling routines. Additionally, if we talk about web engineering, a web application framework must also provide methodologies for the modeling of applications based on the framework. This allows the development of CASE tools which are great assistance for a rapid development process.

3. Characteristics of the DaVinci Framework

Like most of the existing web application frameworks, DaVinci is also fundamentally based on the Model View Controller (MVC) Model 2 pattern [16, Ch. 11]. There are three base components which can be assembled when modeling a DaVinci application: models, views and actions. Of course, beside these function blocks optional user libraries can be imported. A software API based on the architectural framework provides all necessary functionality to combine the basis components to a web application. Each application also asks for a deployment descriptor which can specify common resources.

The DaVinci framework is neither solely a modeling framework nor an architectural framework – it addresses both requirements in the web engineering process. A significant feature is the pre-defined hierarchical GUI-tree that defines the complete set of views and even the possible interaction spots on each view in the application at once. A single webpage is composed when rendering the current state of the view tree. This concept will be outlined in more detail in section 4.3.

DaVinci also provides a database abstraction layer with pooling of multiple persistent connections. There is an additional building block, the repository. A repository can be seen as a bean container which facilitates loading and storing of objects stored in a database. To some certain extent it works like a container managing Entity Beans [14] but it’s much simpler. A special repository type can be used to transparently manage hierarchical trees (self-referencing database tables).

3.1. HTTP requests are function calls

Concerning the interaction via HTTP the architecture of DaVinci uses a special URL scheme. Any incoming request is processed by a central entry point (MVC Model 2). The path info\(^5\) of an URL is used to refer to a specific action to be executed on the server\(^6\). Various parameters required by an action can follow in the URL separated by slashes. This can be compared to a traditional function call, which is simply executed over the web\(^7\). The function call returns the response of the HTTP request, which is an HTML document, an image, or various data of any other MIME type.

\(^5\)The path info is an additional peace of information inside an URL coming after the path to the actual resource.
\(^6\)User rights and role management could be integrated at this point, allowing or refusing the execution of specific actions.
\(^7\)A webservice is very like a function call executed over the web. In the case of DaVinci every request is a function call to the application logic.
This request scheme allows HTTP GET requests without using CGI url-encoding, which can be troublesome with search engines. Another common issue when developing dynamic webpages is, that parameters which are transmitted to the server (either with GET or POST) always come as string values (e.g. the selections of multi-list boxes have to be de-serialized each time). That’s why DaVinci actions classes specify all required and optional incoming parameters and also the types they should be casted to. According to this definition, each parameter value submitted is automatically translated by the framework and will be available for further processing in its correct data type. Basically a DaVinci action can define two kinds of parameters: path info parameters and POST parameters. The values of HTML form elements can of course be sent using POST.

For example the request "GET /davinci.php/showPage/3129 / HTTP1.1" will execute the action "showPage". Because the action class "showPage" defines the incoming path parameter "pageId" of type "integer", the framework will declare an integer variable "pageId" and assign the value "3129".

3.2. The general interaction process of a web application

Figure 1 shows the interaction process of a web application in general. A web application always consists of a server-side application logic\(^8\) and a graphical (or textual) user interaction. The GUI provides various interaction spots like links and HTML forms which will invoke further HTTP requests. A request can change the application logic and also the state of the GUI, which can also be seen as part of the application logic. At the end of a request the GUI is rendered again and sent back as response to the client. While a change in the application logic can simply mean "select a different document" and the GUI can be seen as "document rendered as HTML output", a change could also be the invocation of a webservice on the server while the GUI would simply show the SOAP response without any interaction spots. The former case would be similar to a traditional HTTP request that served static resources.

Basically, a web-based application is a client-server application with two special aspects: the client-side (web browser mostly acts as a thin client) need not to be deployed by the application developer\(^9\) and the communication inbetween is fundamentally stateless \[^3\]. Thus, session handling is an essential part in a web application framework.

4. Modeling with DaVinci

To demonstrate the framework by example, the following scenario is defined. We want to build a little web application that could be configured as the personal starting page in one’s favourite web browser. The application should allow to store personal links (bookmarks) and addresses in a hierarchical folder structure. Furthermore the application should start with a configurable homepage whose URL is stored among the other links. The application should provide some navigation so that the user can switch between "homepage", "links" and "addressbook" (which we will further call "modules").

We will not outline the complete web engineering process including planning, workflow and team management, testing, deployment, etc. here. Many concepts mentioned in section 2. have drawn

\(^8\)The server-side part will always be the core of the application, although application logic can be moved onto the client-side too (e.g. using Scripts, ActiveX, Applets, etc.). Client-side behaviour mostly addresses minor and support functionality (e.g. advanced GUI widgets, DHTML).

\(^9\)Nevertheless, some limitations and compatibility issues between different browser software have to be taken into account during development.
particular attention to this. Instead we will show the most important steps involved when modeling with DaVinci – building and combining models, actions, views and also repositories.

When modeling with DaVinci we suggest the following workflow:

1. analysis of requirements (use cases)
2. finding required entities and modeling repositories (class diagrams)
3. modeling the application logic (using various kinds of UML diagrams)
4. building the GUI tree (class diagram, user-centric approach [4])
5. finding the required interaction spots (class diagram of the GUI tree)
6. generating the facets with CASE tools
7. build and deploy all components

When modeling the application logic (using DaVinci models and imported libraries) dynamic diagram types like activity and sequence diagrams can also be helpful. They can be applied like when modeling for traditional software. We will not cover step 1 (since the scenario is already given and quite simple), step 6 and 7.

4.1. Entities and repositories

To be able to store links and addresses we will need two self-referencing entities, respectively two repositories (figure 2). Each repository in the DaVinci framework has a data source name (DSN) similar to JDBC or ODBC and a unique handle. The handle is used to fetch a reference to a specific repository from the RepositoryManager which is a singleton class managing all the repositories. The repository layout specifies the fields of the records. There exist type mappings for each database supported by the framework (currently MySQL only).

Repositories can even be dynamically created and maintained using the manager API.
4.2. Application logic

The fact, that we use a pre-defined GUI tree eliminates the complexity of the hypermedia paradigm to some certain extent. A web application in the context of DaVinci is more like an application that’s based on the web than a set of web pages that are generated dynamically. Thus, the application logic can be modeled like for traditional software which is a great advantage in contrast to other approaches.

The main building blocks of the application logic are models. They can aggregate repositories, modify the GUI tree and link to additional libraries. The public functions of a model class are invoked by various actions (see later in section 4.4.). Since models account for the application’s state and persistent data they are also responsible for serving data for views. That’s why each view node in the GUI tree can refer to a model class that’s responsible for serving data. The model therefore must implement the interface function "getViewData(String viewHandle)" which is supposed to return appropriate data for the requesting view node.

In our example we have two models so that we can split all the logic for the bookmark and the addressbook component (figure 3). The models provide various public functions for creating, editing and deleting links respectively addresses. Each model aggregates the corresponding repository (of the type SimpleTreeRepository) for the hierarchical storage of links and addresses.

4.3. Modeling the GUI tree

Traditionally the GUI has been regarded separately from the display of content. But if we look at windowing toolkits today and also at web pages, the GUI fuses with the display of content. Thus, the graphical user interface does both, present content and provide interaction spots inside parts of the content. Hyperlinks are interaction spots inside documents on the web. If we think of a GUI shown in a window frame or on the full screen, there are areas with information only and areas which act as interaction spots (a navigation, menu or as already explained, concerning web applications: hyperlinks and HTML forms).

We can describe each GUI-model by hierarchical trees with "view nodes" even including dynamics.
Figure 3. models for the demo application

and behaviour. The demonstrated application’s GUI can be outlined using a simple flat view like in figure 4. It’s useful to abstract the flat views and omit details. The complete view hierarchy of big GUIs would be too complex to show in one sketch. Therefore figure 4 doesn’t show the sub-trees of the three modules. The flat view also shows the location of the views elements in the window. In the middle of the window we have the page’s body. The content of the body can either be the homepage, the addressbook or the links repository. The interaction spots provided by the navigation on the top will invoke a change in the GUI tree and switch the page body accordingly.

Figure 4. draft of the GUI, flat

The corresponding hierarchical GUI tree is modeled using UML class diagrams in figure 5. The framework uses XML for the GUI tree configuration.
Figure 5. GUI tree including interaction spots (functions)
dynamical aspect can be achieved by special "switch nodes". A switch node can switch between various underlaying sub-trees in the GUI model and perform a change in the GUI when the tree is rendered at the end of each request. In fact, this GUI tree model has been derived from the scenegraph model, which is used in computer graphics (e.g. AutoCAD, Java3D API).

Each session of the web application starts with an initial configuration of the complete GUI tree\textsuperscript{12}. All switch nodes are set to an initial underlaying sub-tree (parameter "currentView", see figure 5). During the session the GUI tree can be altered through changing switch nodes. A very important advantage is, that navigation paths in the application are kept very short. That's because deeper switch nodes are not changed when a node at a higher level in the tree is changed. Changing back the higher switch node will exactly bring up the previous view state again.

Each view node in figure 5 has an associated file (parameter "file") that will be parsed and included into the response's output if it's visible in the current tree state. Some view nodes also have a "model" parameter. These views will request data from the specified model. For example the view node "main-frame.body.homepage"\textsuperscript{13} has the model "Links" associated. Before the view is parsed the framework invokes the function "getViewData(String viewId)" of the model. As a consequence inside the view file (that could be a JSP or PHP script) the data is available for output. This strategy is required to achieve the strict separation of application logic and presentation. So it's possible to change all the view files and provide different output channels for various devices (e.g. PDAs or cell phones) without changing any other part of the web application.

4.4. Adding interaction spots

To complete the GUI model we now insert interaction spots into the GUI tree. This is done by adding operations to all the classes in the UML model in figure 5. An operation stands for an interaction spot and will later refer to DaVinci actions. Some actions require parameters, for instance "SetModule(String handle)" in the view "mainframe.body.homepage"\textsuperscript{13} has the model "Links" associated. Before the view is parsed the framework invokes the function "getViewData(String viewId)" of the model. As a consequence inside the view file (that could be a JSP or PHP script) the data is available for output. This strategy is required to achieve the strict separation of application logic and presentation. So it's possible to change all the view files and provide different output channels for various devices (e.g. PDAs or cell phones) without changing any other part of the web application.

All interaction spots point to URLs using the DaVinci URL scheme (as explained in section 3.1.). Each URL further refers to an action class. For instance, the interaction spot "SetModule(String handle)" could cause the HTTP request "GET /davinci.php/SetModule/links / HTTP1.1". The DaVinci framework will then type-cast and translate all path info parameters (only "handle" in this case, which has the value of "links") and POST parameters (none here). Next, the framework invokes the interface function "execute()" of the action class "SetModule". In the example given, the action will change the switch node "mainframe.body" of the GUI tree to "links". When the execute() function exits, the framework will render the GUI tree and send the generated output to the client. The source code of the action class discussed is printed in listing 1. Note that an action class not necessarily calls model functions. In this case only the GUI is changed.

To show again how a request will be processed by the framework, figure 6 shows an UML sequence diagram of the user interaction. Since the prototype was written in PHP – which is a scripting language

\textsuperscript{12} Combined with user profiles a user could therefore recall a previously saved state of the GUI tree or personalize his GUI of the web application.

\textsuperscript{13} To refer to a view node we start from the top and use dots between the levels.”
**class** `SetModule` **extends** `AbstractAction`

```php
/** inside the constr. All path info and POST parameters are defined */
function SetModule () {
    $this->_addPathParamDef('module_id', DATATYPE_VARCHAR, 'homepage');
}

/** exec by the framework, returns true on success, false otherwise */
function execute () {
    $mod = $this->parameters['module_id'];
    if ($mod == 'homepage' || $mod == 'addressbook' || $mod == 'links') {
        ViewGenerator::setSwitch('mainframe.body', $mod);
        return true;
    }
    else return false;
}
```

Listing 1. action class SetModule

-- there are several drawbacks that have to be mentioned. The most important one is, that PHP doesn’t allow to keep objects in memory across subsequent requests\(^{14}\). All session information is serialized and stored in files at the end of a request. At the beginning of each request, some resources have to be re-initned. This applies to all the Manager classes (see figure 6) and also to the ConfigFileParser. Nevertheless the handling of requests is extremely fast.

\(^{14}\)Of course there are approaches to get rid of this shortage. See the SRM project for example: http://www.vl-srm.net

---

**Figure 6. sequence diagram of the user interaction**

#### 5. Conclusion and future work

The GUI tree concept seems to be a very good approach to escape from the hypermedia complexity. When building web applications the focus should be the application logic and not the possibilities of hypermedia. When using DaVinci all views must be pre-defined as nodes of the GUI tree. This heavily differs from the hypermedia paradigm where an arbitrary number of additional pages could be linked into the document structure afterwards. But since the GUI of an application should be well-formed anyway, any view elements must be properly defined in the GUI tree.

The modeling possibilities are sufficient already, but will be extended in future. Once the framework
is stable and powerful enough CASE tools can be written to burst production times. Concerning the architectural framework, the developer can already concentrate on the GUI and the business logic and doesn’t have to implement basic functionality like the handling of HTTP requests, sessions and fetching parameters for instance.

The DaVinci project has already been started in 2003 by Andreas Langegger in the context of his diploma thesis (special thanks to Rimbert Rudisch for his great support!). Since the prototype was implemented with PHP, there are some drawbacks. One could also argue that PHP is too weak for business applications. This is right to some certain extent. However, for many of the requirements of web applications it’s adequate, fast and cheap and it also provides numerous options for application integration. Otherwise the implementation for Java has already been considered too. In future the DaVinci core will be ported to PHP5 first and some improvements are waiting to be done as well. Since PHP5 really lacks of well-designed application frameworks, this task has highest priority.

Finally figure 7 shows a screenshot of a DaVinci-based Content Management System (CMS). For instance, the website of the Samariterbund Feldkirchen in Upper Austria is thoroughly based on DaVinci and uses the DaVinci CMS in the backend. The CMS provides various plug-ins like a MediaSuite and a NewsletterSuite.

Figure 7. DaVinci-based Content Management System

References


