Web Application Modeling Language

Ulrich Wolffgang and Herbert Kuchen

University of Münster, Department of Information Systems
Leonardo-Campus 3, 41819 Münster, Germany
{uwolffga,kuchen}@uni-muenster.de
http://www.wi.uni-muenster.de

Abstract. We present a new approach for modeling web applications based on a new notation called Web Application Modeling Language (WAML). As existing approaches it models the three relevant aspects of web applications, namely core application, presentation, and navigation, separately.
However in contrast to these approaches, WAML supports model-driven software development in a whole by offering (1) a lightweight meta model (2) that is EMF compliant in form of an (3) UML Profile using (4) activity diagrams to model navigational structures. In addition, (5) we have developed corresponding templates for the well-known MDA tool oAW, which allow to generate web applications from WAML models.
Using an example application, a Java EE based order system, we demonstrate that in case of pure CRUD applications 100 % of the code can be generated. In more specific cases, code can be inserted manually into protected regions.

Key words: WAML, MDD, UML, web engineering, meta model

1 Introduction

Web applications are not only becoming more and more widespread but also more complex. Thus, processes and technologies for web engineering are required for developing them easily and with high quality.

There exist several methods and languages for web engineering, which differ w.r.t. the level of abstraction and the notation. These approaches have in common, that they capture the hypertextual character of web applications. According to the separation of concerns, web applications are represented and structured by data, navigation, and presentation models [7]. Some prominent languages and approaches for web modeling are WebML [12], UWE [11], WebSA [13], WAE [4] and OO-H [8].

A very interesting approach for simplifying the development of software is the model-driven software development (MDSD) [9]. Roughly, the idea is to automatically transform the models of a system, possibly in several steps, to platform specific code. To apply MDSD, (1) a meta model is needed that (2) should be MOF or EMF compliant, so it can be referenced by (3) standardized transformation definitions specified in the common transformation languages MOF QVT or

Xpand and Xtend. Also it is useful to have (4) a meta model in form of an UML Profile, that offers (5) activity diagrams to model the navigational structure. The specification of the meta model as an UML Profile simplifies the application of MDSD for web development as most sophisticated and well-known CASE tools can be used.

The mentioned existing approaches do not meet all of these five requirements as none of them offers a lightweight UML Profile using activity diagrams for navigational diagrams. Also WAE and OO-H do not offer standardized transformation definitions based on the meta model. Thus, the aim of the present paper is to develop an UML-based notation called web application modeling language (WAML) for modeling web applications, which offers an alternative approach. Also in this paper, a set of transformations is provided, which supports the model-driven development of web applications by templates that generate complete CRUD-based web applications, i.e. web applications only providing operations for creating, reading, updating, and deleting data.

The rest of our paper is structured as follows. In section 2, the metamodel of WAML is introduced and the choice of the navigation diagrams is explained. In section 3, we show an example WAML model of a web application, namely an order management system. With the transformations defined in section 4 such models can be automatically transformed into source code. In the section 5, we conclude and point out future work.

2 WAML

The Web Application Modeling Language (WAML) is a language for modeling web applications which is suitable for model-driven software development. WAML models contain information on a level of abstraction, which serves on the one hand technical documentation purposes and on the other hand enables the automated generation of source code by simple transformation definitions.

For defining transformations, appropriate language structures are required, which can be expressed as metamodels. E.g. the concept *web page* has to exist as a language element in the metamodel in order to be referenced by transformation definitions in generator frameworks such as openArchitectureWare [5] or AndroMDA [1]. From model elements with the type *web page* e.g. files can then be created for the web pages of a web application.

WAML is based on UML, as the metamodel is specified as an *UML profile*. Profiles and *stereotypes* are part of the lightweight extension mechanism of UML and allow the enrichment of UML by more specific language elements [10]. In the following subsection, the metamodel of WAML is introduced in detail by giving a graphical overview of the language features.

2.1 Metamodel

The metamodel of WAML is divided into three layers by the profiles *content*, *control*, and *view*. By these profiles, the aspects (1) data management and business logic, (2) navigation and operational functionality, and (3) representation

are covered. In this way the separation of concerns is kept via the distinction of content, controller, navigation and presentation models.

The profiles of WAML are structured according to the model view controller architecture (MVC), which splits an application into three layers: its data model, the controller, and the view. The model contains the application data and business logic. The view presents the application data and business logic to the user, while the controller steers the views and processes user actions as well as entered data by calling the appropriate methods of the business logic in the model. A specific target platform for the generation of a web application from a WAML model is Java EE with JavaServer Faces (JSF) [6]. WAML is platform near, but not platform specific. Accordingly, the terminology of WAML is kept generic and it is not focussing on a specific platform. Other platforms such as .NET and Spring can be used, too, as they provide the required object-relational mapping and MVC-based frameworks.

WAML models have to be expressive enough to enable the completely automatic generation of web applications without manual adjustments. The required expressivity is described in the following subsections profile by profile.

Profile Content The profile *content* contains stereotypes for the modeling of entity classes and business classes, i.e. for the modeling of the backend of the web application. With Enterprise JavaBeans (EJB) entity classes are implemented by entities and business classes by session beans [2].

Entity classes are distinguished in WAML by the stereotype *ContentClass* and possess the *tagged value stringName* of type UML-Property. An attribute of the content class can be assigned to this and it is used for the naming of the content class in the running system, e.g. in a `toString` method. The `toString` method is used in the presentation layer e.g. in drop down boxes for the naming of the selectable elements.

By the stereotype *ContentKey*, attributes can be defined as IDs for their entity class. Only exactly one attribute per content class should possess this stereotype and have the type *UML-Integer*. Thus, it is guaranteed that IDs are generally numeric. This simplifies the transformation definitions, since no compound keys must be generated.

Each business classes is annotated by the stereotype *BusinessClass*. It can contain some EJB related tagged values which specify, (1) whether the class requires a state when being transformed to a session bean, (2) whether a session bean shall enable remote or local access only, and (3) whether the class possesses an entity manager. By the meta-association *contentClass*, a content class can be assigned to a business class and it is then primarily administered by the business class. Operations of business classes can be annotated by the stereotypes *LocalOperation* and *RemoteOperation* for declaring, whether they enable remote or local access. If an operation is marked as a *RemoteOperation*, its business class should be annotated accordingly. Standard operations for loading, deleting, saving, and listing of elements of the administered content class can be annotated by the stereotypes *bcLoad*, *bcDelete*, *bcSave* and *bcList*. Based on the signatures

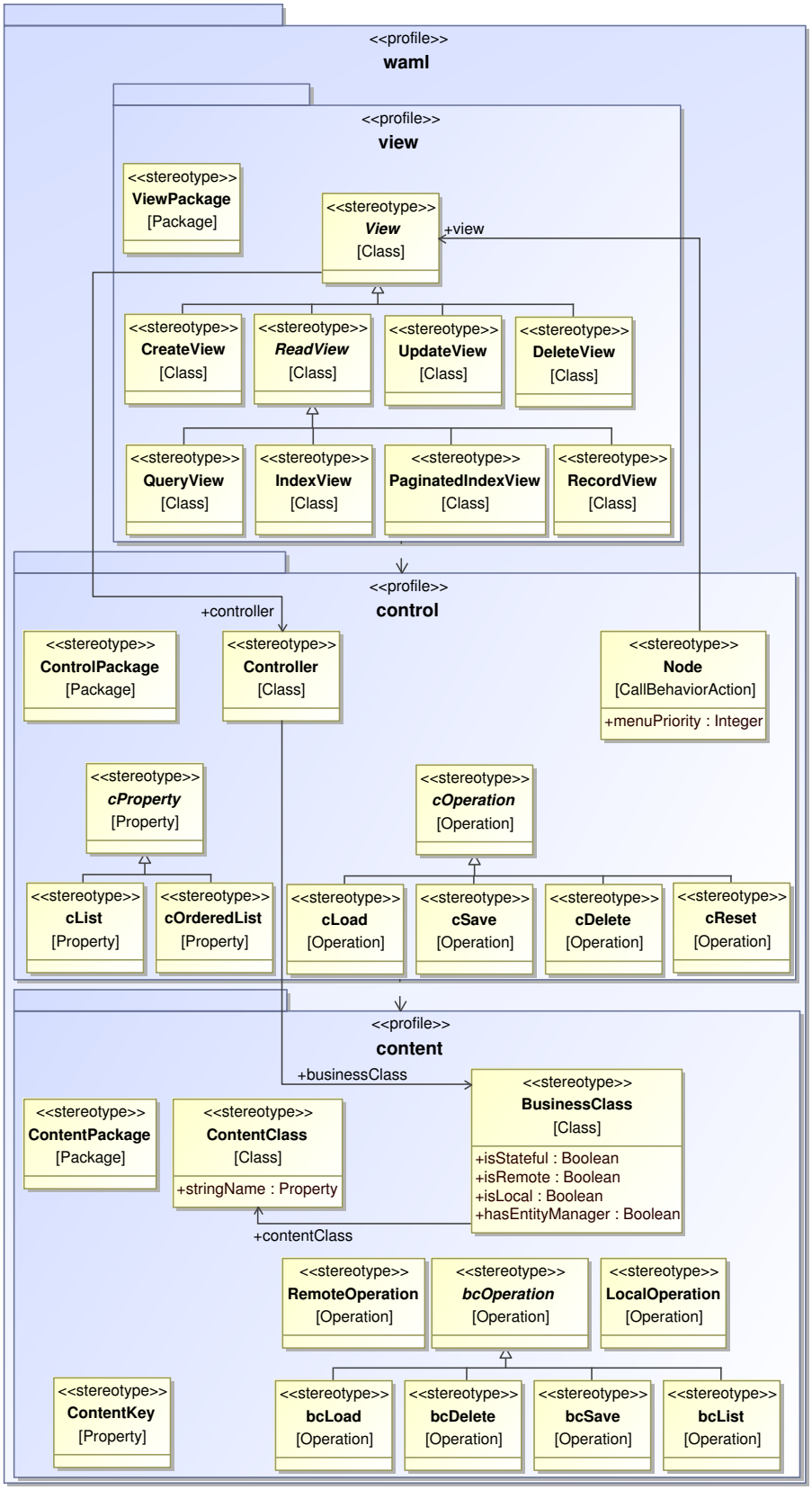

Fig. 1. Metamodel of WAML

and parameters of the business methods also the bodies of the operations can be generated completely automatically. The stereotypes correspond to the standard operations *create*, *read*, *update* and *delete* (CRUD) of data-oriented systems. The functionalities *create* and *update* are merged in the stereotype *bcSave*.

Content and business classes are aggregated in content packages, which are annotated by the stereotype *ContentPackage*. The modeling context of the content and business classes is specified by the declaration of UML packages as content packages. This specification of the context is also made when modeling with language elements from the profiles *control* and *view*. This has the advantage that the transformation definitions can be defined in such a way that depending on the context only elements of the respective profile are transformed. If a content package contains model elements with stereotypes of the view package, these are hence ignored by the transformation definitions. The different handling of profiles forces the modeler to separate concerns and hence to structure a model clearly.

Profile Control The profile *control* contains language elements for the modeling of navigational aspects and controllers. With the stereotype *Controller*, classes can be modeled, which offer the functionality of a graphical user interface by processing MVC model data. A primary business class is assigned to each controller with the meta-association *businessClass*. This class is administered by the controller. Since business classes possess content classes, each controller is also indirectly assigned to a content class.

Just as the operations of business classes, the operations of controllers are annotated by stereotypes, in order to enable the automatic generation of their bodies. These stereotypes, too, correspond to CRUD functionality. Stereotypes for CRUD operations of controllers are distinguished from those for business classes, because their operations differ by their return value. E.g. a bcLoad-Operation of a business class returns an entity, while the cLoad-Operation of a controller stores the requested entity in its state and returns no value. Operations, which reset the state of a controller, are annotated by the stereotype *cReset*. The state contains e.g. the value of the currently administered content class. This way, the user session on the web server can be kept clean.

Navigational structures can be modeled with the stereotype *Node*, by which actions in UML activity diagrams can be annotated. A node corresponds to a web page in the web application. It is enriched with representational functionality by an assigned view. Compared to the controller for operational functionality, a controller for managing the navigational flow does not have to be modeled explicitly, because this controller type is already made available by frameworks such as JavaServer Faces (JSF). Based on the activity diagram, a file must be generated for these frameworks, which configures the controller corresponding to the navigational structures of the web application.

Profile View The profile *view* contains stereotypes for the annotation of nodes with standardized presentation functionality. By the stereotype *View* and its

specializations, presentation elements such as links and tables are aggregated to units. This has the advantage that the models of the view keep abstract and meaningful.

As default views, the *CreateView*, *ReadView*, *UpdateView* and *DeleteView* are provided. They offer representational functionality for (1) the form-based input of single data records, (2) the output of one or more data records, (3) the form-based modification of single data records, and (4) the interactive deletion of single data records. The abstract view *ReadView* is specialized by the views *QueryView*, *IndexView*, *PaginatedIndexView*, and *RecordView*, each providing a different visualization and corresponding for data manipulation. *QueryView* enables a form-based search and output of data records. Both, *IndexView* and *PaginatedIndexView*, represent lists containing data records. The difference between them is, that the first one offers the presentation of a single list containing all records and the second one displays only a subset of all records, enabling a page by page traversal. With *RecordView*, the presentation of a single data record can be modeled. The hierarchy of views provides modularity and enables project-related (user-defined) extensions of the profile by more specific views. Since views are referenced by navigational nodes, they can be flexibly reused for several pages and nodes, respectively. Additionally, the navigation can developed independently of the presentation functionality, w.r.t. to time and responsible people.

2.2 Associations

In UML, references between model elements are expressed primarily by attributes and associations. This leads to the problem that in the transformation definitions attributes of language elements cannot be fixed independently of concrete models and thus cannot be referenced type safe by their name.

The profile mechanism of UML therefore provides tagged values and meta-associations for metamodeling. They can be assigned to language elements in language definitions and can be referenced type safe by their name in transformation definitions. Also the multiplicity of tagged values and meta-associations can be specified already in the metamodel. Compared to stereotyped associations, tagged values and metaassociations have the disadvantage that they are not shown graphically in the model by modeling tools. A difference between tagged values and meta-associations lies in their directionality. Tagged values are only unidirectional, while meta-associations can be both, unidirectional or bidirectional.

2.3 Modeling Navigational Aspects

A characteristic feature of WAML is the use of activity diagrams for the modeling of navigational structures. A diagram for this purpose has to fulfill two conditions: (1) it has to be a behavioral diagram that, for pragmatic reasons, is supported by usual modeling tools, and (2) it has to support the creation of directed graphs. Thus, state machines diagrams and activity diagrams remain

possible choices. While state machine diagrams focus on states and their changes, activity diagrams are intended for describing work flows. Since the latter fits better to our purpose, we have decided to choose activity diagrams. State machines would have been a valid alternative.

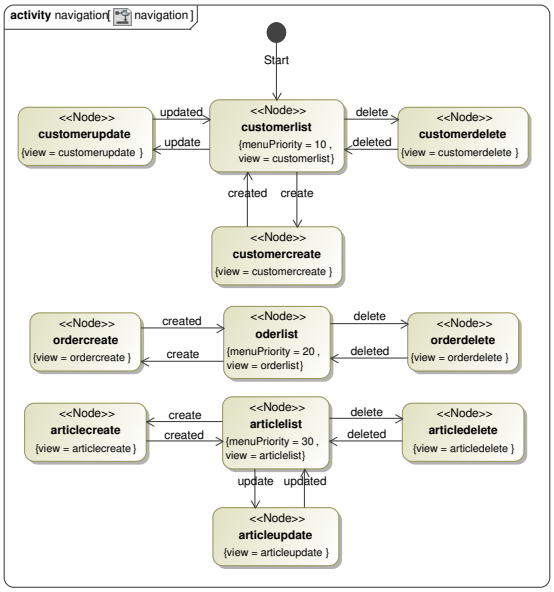

Fig. 2. Navigation diagram of the prototype

In Figure 2, a navigation model based on an activity diagram is shown, which describes the navigation of a simple web application for the processing of personal data. The nodes represent web pages as well as their interactional aspects and the edges define the links between these pages.

3 Example

In the following, the model-driven development of a prototypical web application is presented on the basis of WAML. We will consider an order management system as example.

The content model presented in Figure 3 consists of classes for customers, articles, and orders. In order to represent associations with different kinds of cardinalities, a unidirectional association with cardinality 1:* has been introduced between the classes `Customer` and `Order`, and an unidirectional association with cardinality *:* between classes `Article` and `Order`.

For the content classes, there are corresponding business classes, which contain stereotyped methods for the storing, loading, deleting, and listing of their

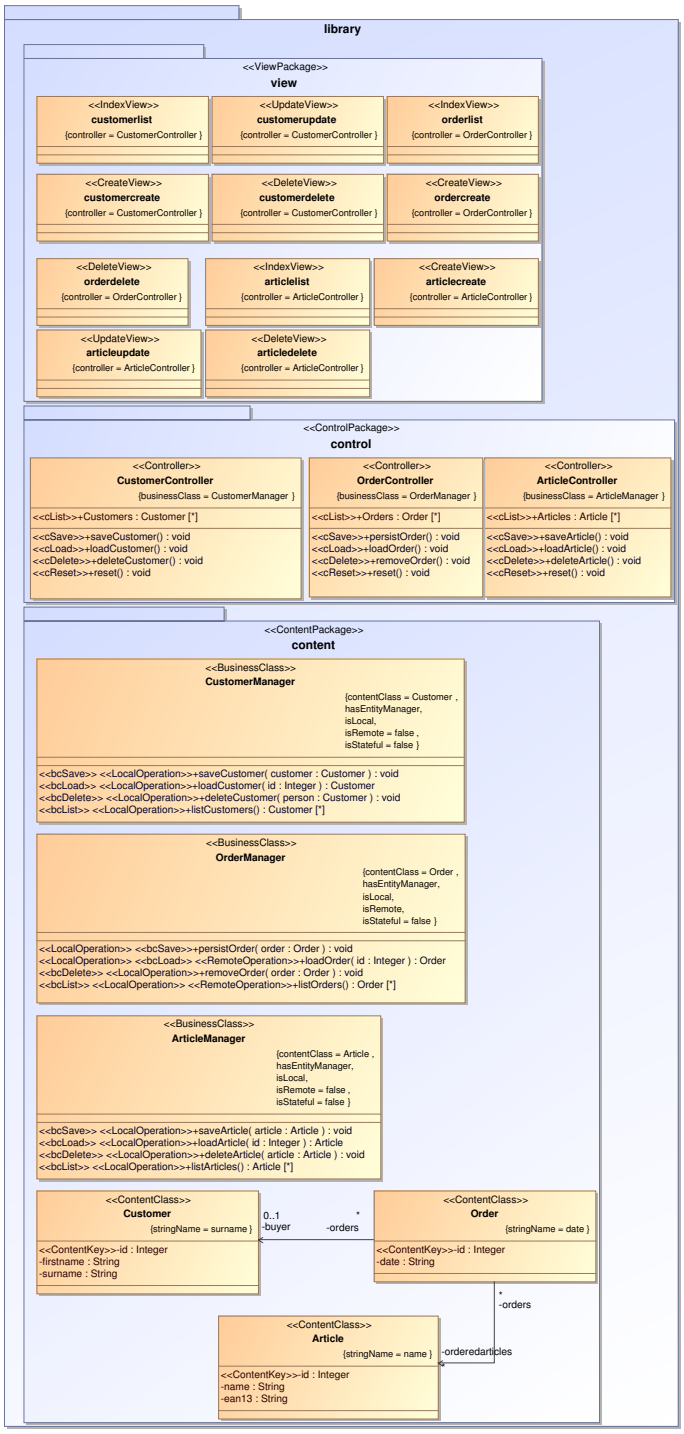

Fig. 3. Class diagram of the prototype

respective content classes. The dependency between a content class and its business class is expressed by the tagged value *contentClass*. For all business classes, it is configured that they reference an entity manager, with which the persistence layer of the web application can be accessed.

The package *control* covers classes for data controllers which, similarly to the business classes, contain methods for the storing, loading, deleting, listing, and resetting of their respective content classes. The controllers reference their corresponding business classes. In the spirit of the MVC pattern, the controllers supply data and methods of the business classes to the view classes.

The package *view* includes classes for presentational functionality such as the listing, input, and deletion of data records for a person. The views are parameterized with their corresponding controllers.

The navigation diagram shown in Figure 2 contains stereotyped actions, which are connected by edges and represent the navigational paths in the management system. The nodes *customerlist*, *orderlist* and *articlelist* are registered for the menu by their menu priority values. In principle, corresponding edges would have to connect each node with these three nodes, because with the menu these three nodes can be accessed from each node. In favor of the clarity of the model, we have omitted menu-based edges and chosen the alternative representation shown. The presentational functionality of the nodes is assigned to the views by means of parameters. For instance, the edge between the nodes *customerlist* and *customerdelete* expresses that in the visualization each row of the table containing customer data should have one link to another page for deleting the record. Thus the node for deletion is present in the context of the list node, and it parametrizes in this way the view of the list node.

Fig. 4. Generated page of node ordercreate

With these models and the following transformation definitions the system can be generated as a whole. The screenshot in figure 4 shows the look and feel of a page in the resulting application.

4 Transformation Definitions

For the generation of web applications from WAML models, a MDSD generator is needed, which creates source code by means of transformation definitions

from input models. With their template languages, generator frameworks such as openArchitectureWare and AndroMDA offer a high level of flexibility for the development and maintenance of transformation definitions. For the following transformation definitions, openArchitectureWare with its languages Xpand and Xtend is used.

Xpand is a template language, which makes it possible to query values from input models, concatenate these values with static text and write the results to files [5]. Templates consists of arbitrary text, which is enriched by Xpand expressions. During the generation process, these expressions are evaluated by the template engine and the results are then inserted at the appropriate places in the text [9].

In the following, some example templates and functions are described, which are specific for WAML as well as web applications beyond the usual generation of class bodies in CASE tools. As target frameworks, Java EE with EJB and JSF are used.

4.1 Root Template

The primary root template is initialized by the generator of openArchitectureWare with an UML model and expands the following secondary root templates for the model elements. The meta operation *this.ownedElement()* returns a list of the elements on the highest hierarchy level of the model such as classes and packages, but not the elements contained in these packages.

Xpand automatically selects and expands the appropriate template for a model element by type polymorphism. Therefore the model is scanned for activities, content packages, and control packages. If an activity is found, the appropriate root template for navigation models is expanded. Similarly for content packages and control packages the corresponding root template is opened. All remaining model elements with other types than the specified ones are caught by the last root template for generic UML elements and are not processed any further. This generic template is demanded by Xpand to ensure the processing of all types by inclusion-based polymorphism [3].

```
«DEFINE Root FOR uml::Model»
 «EXPAND Root FOREACH this.ownedElement»
«ENDDEFINE»
«DEFINE Root FOR uml::Package»
 «EXPAND Root FOREACH this.ownedElement»
«ENDDEFINE»
«DEFINE Root FOR content::ContentPackage»
 «EXPAND waml::content::Root::Root»
«ENDDEFINE»
«DEFINE Root FOR control::ControlPackage»
 «EXPAND waml::control::Root::Root»
«ENDDEFINE»
«DEFINE Root FOR uml::Activity»
 «EXPAND waml::control::Root::Root»
```

```
«ENDDEFINE»
«DEFINE Root FOR uml::Element»«ENDDEFINE»
```

4.2 Templates for Pages

The navigation structure is modeled in WAML by activity diagrams. For navigational nodes, a template is defined, which describes the essential structure of a web page. The content of a page is generated by the template of the view according to the value of the tagged value *view*. If the node does not reference a view, a protected region is inserted into the page, which can be filled manually with custom content.

```
«DEFINE Root FOR control::Node»
 «FILE getNodeJspxFilename(this)»
  «EXPAND Header::Header»
  <h1>«this.name»</h1>
  «IF this.view.length > 0»
    «EXPAND waml::view::Root::Root(this) FOR getView(this)»
  «ELSE»
   «PROTECT CSTART "<!-- " CEND " -->" ID getNodeId(this)»
   «ENDPROTECT»
  «ENDIF»
  «EXPAND Footer::Footer»
 «ENDFILE»
«ENDDEFINE»
```

As a special node is the initial node, represented by the UML element *InitialNode*. If the navigation model contains an initial node with an outgoing edge to another node, an *index.jsp* is generated for the initial node, which contains a redirection to the web page of the referenced node. The forwarding must not be made on the server-side with e.g. a *<jsp:forward>*, but the browser has to be redirected on the client-side by a HTTP code 301 in the header of the web page. This is important, because thereby the path of the target web page always stays correct. If the referenced web page contains relative links such as links to stylesheets, these could otherwise refer to wrong directories.

```
«DEFINE Root FOR uml::InitialNode»
 «FILE "index.jsp"»
  <% response.sendRedirect(".
    «getNodeIfaceId(this.outgoing.target.typeSelect(control::Node).
      first())» "); %>
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
  <html><body></body></html>
 «ENDFILE»
«ENDDEFINE»
```

4.3 Templates for the Menu

The template for the header of a page contains introductory HTML tags as well as JSP tags for JSF tag libraries. Additionally, the header template creates

the menu according to the following template listing. This template creates the menu as a list of menu elements, which are contained in a *JSF-Form* referencing the pages by *JSF-commandLinks*. The menu entries are constructed from those nodes, whose tagged value *menuPriority* has a numeric value.

```
«DEFINE Root FOR uml::Element»
 <h:form>
  <ul>
   «FOREACH menuEntryNodes(this.getModel()) AS Node»
    <li>
     <h:commandLink action="menu:«getNodeIfaceId(Node)»"
       value="«Node.name»"/>
    </li>
   «ENDFOREACH»
  </ul>
 </h:form>
«ENDDEFINE»
```

The menu elements can be sorted based on their numeric priority. This is a decentralized and a simple way for the generation of a one-dimensional linear menu.

```
List[control::Node] menuEntryNodes(uml::Model m) :
 m.allOwnedElements().typeSelect(control::Node).
   select(e|e.menuPriority > 0).sortBy(e|e.menuPriority);
```

4.4 Templates for Descriptors of the Navigation

The navigation is expressed in JSF by navigation rules, which are stored in the descriptor file *faces-config.xml*. In JSF, static as well as dynamic navigation structures can be created [6]. With static navigation, a click on links and form buttons results in opening the same web page. With dynamic navigation, the MVC controller chooses, which page to open depending on the result of an executed operation. If the user has, for instance, filled and submitted a web form, the controller can decide based on a method for input validation whether the received form data are inconsistent and it can possibly initiate a redirection to a page for the correction of the data.

Technically, dynamic navigation is realized by returning a result string from the validating method, which processes form data. The return string is assigned to an *outcome* in the *faces-config.xml*. For each source page, a *navigation rule* exists, in which for every outcome a target page is defined. Depending on the result string of the controller method, the corresponding navigation case is selected and the page indicated in the *faces-config.xml* is opened by the JSF controller.

The template for the generation of the *faces-config.xml* supports dynamic navigation structures, as the name of an edge is interpreted as the return string of the outcome. From the outgoing edges of a navigational node, the navigation cases are generated. A navigation rule is created for each node, which possesses outgoing nodes. As an outcome for each outgoing edge, a navigation case with the name of the edge is defined.

```
«DEFINE Root FOR uml::Model»
 «FILE "faces-config.xml"»
  <?xml version="1.0" encoding="UTF-8"?>
  ...
  <faces-config>
  «FOREACH this.allOwnedElements().
   typeSelect(control::Node) AS n»
   «IF n.outgoing.size > 0 ||
     menuEntryNodes(this.getModel()).size > 0»
    <navigation-rule>
     <from-view-id>«getNodeIfaceId(n)»</from-view-id>
     «FOREACH n.outgoing AS o»
      <navigation-case>
      <from-outcome>«o.name»</from-outcome>
      <to-view-id>«getNodeIfaceId(o.target)»</to-view-id>
     </navigation-case>
    «ENDFOREACH»
    «FOREACH menuEntryNodes(this.getModel()) AS MenuNode»
     <navigation-case>
      <from-outcome>menu:«getNodeIfaceId(MenuNode)»</from-outcome>
      <to-view-id>«getNodeIfaceId(MenuNode)»</to-view-id>
     </navigation-case>
    «ENDFOREACH»
    </navigation-rule>
   «ENDIF»
  «ENDFOREACH»
  </faces-config>
 «ENDFILE»
«ENDDEFINE»
```

If nodes are referenced by menu links, a navigation case must be created for each page and for each menu link, such that the menu links can be followed from each page. The outcomes are generated automatically by concatenating the prefix *menu:* with the name of the page targeted by the menu link.

4.5 Templates for Methods of Business Classes

Business classes can be implemented e.g. in EJB by session beans and can offer predefined CRUD methods, that can be automatically generated with their bodies by the following templates.

Methods for the loading data records are annotated in WAML by the stereotype *bcLoad*. By the lines 2 - 5 of the template below, a method signature of the form *public Person loadPerson(Integer id)* is constructed. In UML, a return type can be assigned to an operation [10]. Based on the return type and of the implicitly assumed method parameter *id*, a method body is generated, in which the entity manager returns the appropriate content class.

```
«DEFINE Operation FOR content::bcLoad»
 «getVisibility(this)» «getTypeName(this)» «this.name»
```

```
   («EXPAND Class::Parameter FOREACH getParameterList(this)
   SEPARATOR ",")
   { return this.em.find(«getTypeName(this)».class, id);}
«ENDDEFINE»
```

Methods for persisting content classes are annotated by the stereotype *bcSave* in WAML. The corresponding method body is generated in such a way that the entity manager persists the object given as the first parameter of the operation.

```
«DEFINE Operation FOR content::bcSave»
 «getVisibility(this)» «getTypeName(this)» «this.name»
 («EXPAND Class::Parameter FOREACH getParameterList(this)
   SEPARATOR ",")
 { em.merge(«getParameterList().first().name»);}
«ENDDEFINE»
```

The deletion method searches for a content class using a specified ID and requests from the entity manager the deletion of the content class.

```
«DEFINE Operation FOR content::bcDelete»
 «getVisibility(this)» «getTypeName(this)» «this.name»
 («EXPAND Class::Parameter FOREACH getParameterList(this)
   SEPARATOR ",")
 { this.em.remove(this.em.find(
   «getTypeName(getParameterList(this).first())».class,
   «getParameterList(this).first().name».getId()));}
«ENDDEFINE»
```

The listing method creates a query for the selection of the instances of a content class and returns them in a result list.

```
«DEFINE Operation FOR content::bcList»
 «getVisibility(this)» «getTypeName(this)» «this.name»
   («EXPAND Class::Parameter FOREACH getParameterList(this)
   SEPARATOR ",")
   { Query q = this.em.createQuery("FROM «this.type.name»");
   return («getTypeName(this)») q.getResultList(); }
«ENDDEFINE»
```

5 Conclusion and Future Work

We have introduced the web application modeling language WAML as an UML profile. Moreover, we have developed corresponding Xpand templates for the generator framework openArchitectureWare. These templates transform WAML models automatically to code for the target platform Java EE with EJB and JSF. Thus, WAML and the corresponding templates enable the model-driven development of web applications. The level of abstraction of WAML has been chosen in such a way that the models are sufficiently abstract, but nevertheless

allow a precise parameterization of the templates and an accurate technical documentation. Finally, we have shown based on an example application, a order management system, that the goal of the fully automated generation of standard functionality can be achieved. Also alternative systems such as a library management system could be generated with the same metamodel and templates. For more complicated business cases, hand written code can be inserted into protected regions.

The following extensions of our prototypical approach are reasonable. The generator can be enriched with more view templates that could allow a more precisely parametrization of the views. In particular, the view templates should be enhanced, so that the web application offers interaction by message boxes with the user when the referential integrity of data is concerned. On the server-side the templates should be enriched with transaction handling in order to manage collisions that can occur in multi-user applications. For ensuring consistent models constraints could be introduced, which allow a model validation before the execution of the generator.

References

1. AndroMDA, http://www.andromda.org
2. Burke, B., Monson-Haefel, R.: Enterprise JavaBeans 3, 5th Edition, O'Reilly Media (2006)
3. Cardelli, L., Wegner, P.: On Understanding Types, Data Abstraction, and Polymorphism. ACM Computing Surveys, Vol. 17, No. 4, pp. 471–522 (1985)
4. Conallen, J.: Building Web Applications with Uml, 2nd Edition. Addison-Wesley, MA (2002)
5. Efftinge, S., Friese, P., Haase, A., Kadura, C., Kolb, B., Moroff, D., Thoms, K., Völter, M.: openArchitectureWare User Guide, Version 4.2. http://www.eclipse.org/gmt/oaw/doc/4.2/openArchitecture (2007)
6. Geary, D.M., Horstmann, C.S.: Core JavaServer Faces, 2nd Edition. Prentice Hall, NJ (2007)
7. Kappel, G., Pröll, B., Reich, S.: Web Engineering: the Discipline of Systematic Development of Web Applications. Wiley, Heidelberg (2006)
8. Gómez, J., Cachero, C.: OO-H Method: Extending UML to Model Web Interfaces. Idea Group Publishing, pp. 144 – 173, (2003).
9. Stahl, T., Völter, M.: Model-driven Software Development: Technology, Engineering, Management. Wiley, Heidelberg (2006)
10. Object Management Group: Unified Modeling Language (UML), Version 2.1.2. http://www.uml.org/ (2008)
11. Koch, N. , Knapp, A., Zhang, G., Baumeister, H.: UML-based Web Engineering: An Approach based on Standards. In Olsina, L., Pastor, O., Rossi, G., Schwabe, D., ed., Web Engineering and Web Applications Design Methods, volume 12 of Human-Computer Interaction Series, chapter 7. Springer, (2007).
12. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., S. Comai, Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann, (2002).
13. Meliá, S., Gomez, J.: The WebSA Approach: Applying Model Driven Engineering to Web Applications. J. Web Engineering, 5(2), 2006.