Model-Driven Generation of Web Applications in UWE[1]

Andreas Kraus, Alexander Knapp, and Nora Koch

Ludwig-Maximilians-Universität München, Germany
{krausa, knapp, kochn}@pst.ifi.lmu.de

Abstract. Model-driven engineering (MDE) techniques address rapid changes in Web languages and platforms by lifting the abstraction level from code to models. On the one hand models are transformed for model elaboration and translation to code; on the other hand models can be executable. We demonstrate how both approaches are used in a complementary way in UML-based Web Engineering (UWE). Rule-based transformations written in ATL are defined for all model-to-model transitions, and model-to-code transformations pertaining to content, navigation and presentation. An UWE run-time environment allows for direct execution of UML activity models of business processes.

1 Introduction

Model-driven engineering (MDE) technologies offer one of the most promising approaches in software engineering to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively [18]. MDE advocates the use of models as the key artifacts in all phases of the software development process. Models are considered first class entities even replacing code as primary artifacts. Thus, developers can focus on the problem space (models) and not on the (platform specific) solution space [18]. The main objective is the separation of the functionality of a system from the implementation details following a vertical separation of concerns.

The area of Web Engineering, which is a relatively new direction of Software Engineering that addresses the development of Web systems [8], is as well focusing on the MDE paradigm. Most of the current Web Engineering approaches (WebML [5], OO-H [6], OOWS [20], UWE [9], WebSA [15]) already propose to build different views of Web systems following a horizontal separation of concerns.

The most well known approach to model driven engineering is the Model Driven Architecture (MDA) defined by the Object Management Group (OMG)[2]. Applications are modeled at a platform independent level and are transformed by model transformations to other models and (possibly several) platform specific implementations.

[1] This research has been partially supported by the project MAEWA "Model Driven Development of Web Applications" (WI841/7-1) of the Deutsche Forschungsgemeinschaft (DFG), Germany and the EC 6th Framework project SENSORIA "Software Engineering for Service-Oriented Overlay Computers" (IST 016004).

[2] Object Management Group (OMG). MDA Guide Version 1.0.1. omg/2003-06-01, http://www.omg.org/docs/omg/03-06-01.pdf

The development process of the UML-based Web Engineering (UWE [9]) approach is evolving from a manual process (based on the Unified Process [7]) through a semi-automatic model-driven process (based on different types of model transformations [9]) to a model-driven development process that can be traversed fully automatically in order to produce first versions of Web applications. A table of mapping rules is shown in [9]. The UWE approach uses recently emerged technologies: model transformation languages like ATL[3] and QVT[4]. In this paper we present a fully implemented version using the ATLAS environment and based on ATL transformations.

UWE applies the MDA pattern to the Web application domain from the top to the bottom, i.e. from analysis to the generated implementation ([9][12]). Model transformations play an important role at every stage of the development process. Transformations at the platform independent level support the systematic development of models, for instance in deriving a default presentation model from the navigation model. Then transformation rules that depend on a specific platform are used to translate the platform independent models describing the structural aspects of the Web application into models for the specific platform. Finally, these platform specific models are transformed (or serialized) to code by model-to-text transformations. In a complementary way platform independent models describing the business processes are executed by a virtual machine thus providing a seamless bridge between modeling and programming. The UWE approach is completely based on standards – in the same way as MDA – facilitating extensibility and reusability.

The remainder of this paper is structured as follows: First we give a brief overview on model-driven Web engineering. Next we present the UWE approach to MDWE and focus on the model-based generation of Web application using the model transformation language ATL and the run-time environment for executable business processes. We conclude with a discussion on related work, some remarks and an outline of our future plans on the use of model transformation languages.

2 Model-Driven Web Engineering

Model-driven Web Engineering (MDWE) is the application of the model-driven paradigm to the domain of Web software development where it is particularly helpful because of the continuous evolution of Web technologies and platforms. Different concerns of Web applications are captured by using separate models e.g. for the content, navigation, process and presentation concerns. These models are then integrated and transformed to code whereas code comprises Web pages, configuration data for Web frameworks as well as traditional program code.

2.1 Variants in Model-Driven Approaches

Model-driven Web development is an effort to raise the level of abstraction at which we develop Web software. MDWE processes can be achieved either by code genera-

[3] ATLAS Transformation Language and Tool, http://www.eclipse.org/m2m/atl/doc/

[4] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Final Adopted Specification. http://www.omg.org/ docs/ptc/05-11-01.pdf

tion from models or by constructing virtual machines that execute models directly. The translational approach is predominant and is supported by so called model transformations. An interpretational approach offers the benefits of early verification through simulation, the ability to separate implementation decisions from understanding of the problem, and the ability to execute the models directly and efficiently on a wide variety of platforms and architectures.

In addition, following the classification of McNeile there are two interpretations of the MDE vision named "elaborationist" and "translationist" approaches [14]. Following the "elaborationist" approach, the specification of the application is built up step by step by alternating automatic generation and manual elaboration steps on the way from computational independent model (CIM) to a platform independent model (PIM) to a platform specific model (PSM) to code. Today, most approaches based on MDA are "elaborationist" approaches, which have to deal with the problem of model and/or code synchronization. Some tools support the regeneration of the higher-level models from the lower level models, also called reengineering. In a "translationist" approach the platform independent design models of an application are automatically transformed to platform specific models, which are then automatically serialized to code. These PSMs and the generated code must not be modified by the developer because roundtrip engineering is neither necessary nor allowed. UWE is moving from an "elaborationist" to a "translationist" approach.

2.2 Role of Model Transformations

Transformations are vital for the success of a model-driven engineering approach for the development of software or in particular of Web applications. Transformations lift the purpose of models from documentation to first class artifacts of the development process. Based on the taxonomy of transformation approaches proposed by Czarnecki [4] we classify model transformation approaches in model-to-code and model-to-model approaches, which differ on providing or not a metamodel for the target programming language.

Model-to-model transformations translate between source and target models, which can be instances of the same or different metamodels supporting syntactic typing of variables and patterns. Two categories of rule-based transformations are distinguished: declarative and imperative and it is worth to mention two relevant subcategories of declarative transformations: graph transformation and relational approaches. Graph-transformation rules that consist in matching and replacing left-hand-side graph patterns with right-hand-side graph patterns. AGG[5] and VIATRA[6] are examples of tools supporting a graph model transformation approach. Relational approaches are based on mathematical relations. A relation is specified by defining constraints over the source and target elements of a transformation. A relation has no direction and cannot be executed. Relational approaches with executable semantics are implemented with logic programming using unification-based matching, search and backtracking. QVT and ATL support the relational approach and provide impera-

[5] Attributed Graph Grammar System, http://tfs.cs.tu-berlin.de/agg/

[6] VIATRA 2 Model Transformation Framework, http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/VIATRA2/index.html

tive constructs for the execution of the rules, i.e. they are hybrid approaches. A different approach is the declarative and functional transformation language XML for transforming XML documents. As MOF compliant models can be represented in the XML Metadata Interchange format (XMI), XSLT[7] could in principle be used for model-to-model transformations and transformed to code using languages such as MOFScript that generate text from MOF models[8]. This approach has scalability problems, thus it is not suited for complex transformations.

3 The UWE Approach to MDWE

The approach followed in UWE consists of decoupling the construction of models using a UML[9] profile, defining transformation rules using a model transformation language, and providing a run-time environment, respectively. Model type transformations map instances from a source metamodel (defining the source types) to instances of a target metamodel (defining the target types). Therefore, the MDE approach is based on metamodels and transformations. All metamodels share the same meta-metamodel (MOF). The proposed solution for the executable models is the use of a platform specific implementation of the platform independent abstract Web process engine. A transformation maps the process flow model to XML nodes, which represent the corresponding configuration of the runtime environment. This runtime process engine is part of the platform specific runtime environment.

There is no restriction on the employed modeling tool as long as it supports UML 2.0 profiles and stores models in the standardized model interchange format. We selected ATL as a Query/View/Transformation language and the ATLAS transformation engine. The Spring framework was selected as runtime environment engine.

3.1 UWE Process

Applying the MDA principles (see Sect. 2), the UWE approach proposes to build a set of CIMs, PIMs, and PSMs as results of the analysis, design and implementation phases of the model-driven process. The aim of the analysis phase is to gather a stable set of requirements. The functional requirements are captured by means of the requirements model. The requirements model comprises specialized use cases and a class model for the Web application. The design phase consists of constructing a series of models for the content, navigation, process, presentation and adaptivity aspects at a platform independent level. Transformations implement the systematic construction of dependent models by generating default models, which then can be refined by the designer. Finally, the design models are transformed to the platform specific implementation. This UWE core process is extended with the construction of a UML state machine – called "big picture" in our approach – that integrates the design models. The objective of the big picture model is the verification of the UWE

[7] W3C. XSL Transformations (XSLT) Version 1.0, www.w3.org/TR/xslt
[8] Eclipse subproject. MOFScript. http://www.eclipse.org/gmt/mofscript/
[9] OMG. Unified Modeling Language 2.0, 2005. http://www.omg.org/ docs/formal/05-06-04.pdf

models by the tool Hugo/RT, a UML model translator for model checking and theorem proving [10]. In addition, architectural features can be captured by a separate architecture model using the techniques of the WebSA (Web software architecture) approach [15] and further integrated to the so far built functional models.

In this work we focus on the UWE core process depicted in Fig. 1 as a stereotyped UML activity diagram. Models are represented with object nodes and transformations as stereotyped activities (special circular icon). A chain of transformations then defines the control flow.

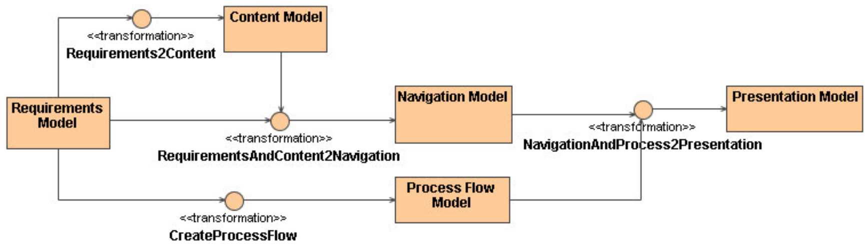


Fig. 1. UWE core process for CIM to PIM and PIM to PIM

3.2 UWE Metamodel

The metamodel is structured into packages, i.e. requirements, content, navigation, process, presentation and adaptation. It is defined as a "profileable" extension of the UML 2.0 metamodel providing a precise description of the concepts used to model Web applications and their semantics. We restrict ourselves to illustrate the approach by means of an excerpt of the presentation metamodel, for further details on the UWE metamodel see [11] and [12].

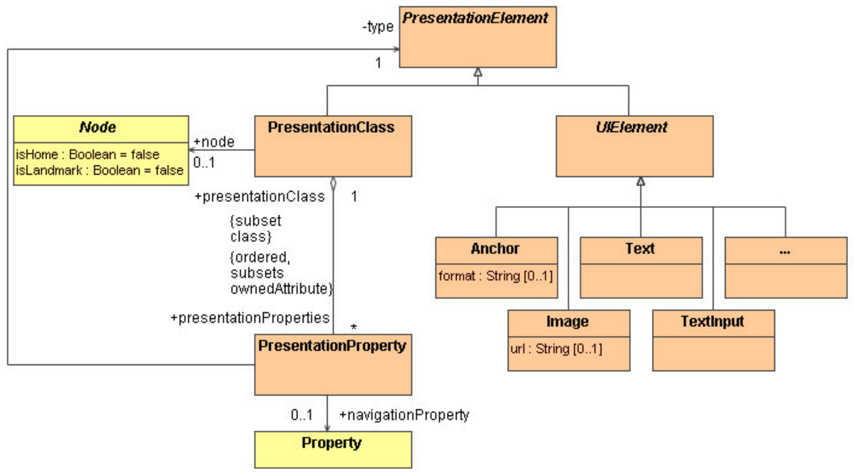


Fig. 2. UWE presentation metamodel

The presentation metamodel defines the modeling elements required to specify the layout for the underlying navigation and process models. A `PresentationClass` is a special class representing a Web page or part of it and is composed of user interface elements and other presentation classes. `UIElements` are classes that represent the user interface elements in a Web page. The presentation metamodel is depicted in Fig. 2. Anchors for example represent links in a Web page, and optionally a format expression may be defined for specification of the label that the anchor should have.

OCL class invariants are used to define the well-formedness rules for models, i.e. the static semantics of a model, to ensure that a model is well formed before executing a transformation.

3.3 Model to Model Transformations in UWE

Requirement specification is based on UML use cases for the definition of the functionality of a Web application. The design phase consists of constructing a series of models for the content, navigation, process and presentation aspects at a platform independent level. Transformations implement the systematic construction of dependent models by generating default models, which then have to be refined by the designer. The approach uses the ATL transformation language in all phases of the development process.

We illustrate our approach by means of a simple project management system, which is part of the GLOWA Danube system, an environmental decision support system for the water balance of the upper Danube basin. The project management system allows adding, removing, editing and viewing of two types of projects, user projects and validation projects. At CIM level the UWE profile provides different types of use cases for treating browsing and transactions (static and dynamic) functionality. UWE proposes the use of stereotypes «navigation» and «web process», to model navigation and process aspects, respectively. A content model of a Web application is automatically derived from the requirements model by applying a transformation `Requirements2Content`. The developer can refine the resulting default content model by adding additional classes, attributes, operations, associations etc. Such a refined content model is represented as a UML class diagram (see Fig. 3).

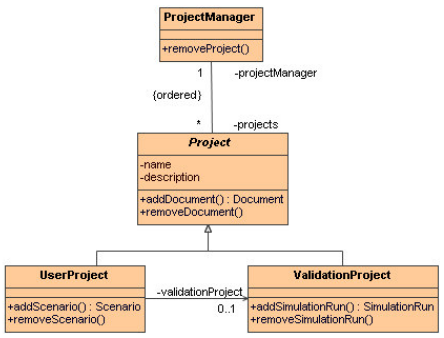


Fig. 3. Content model of project management system (simplified)

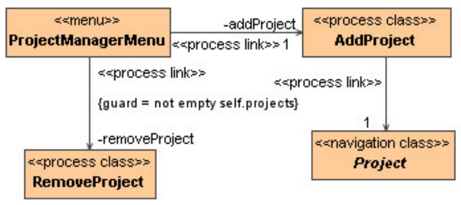


Fig. 4. Partial navigation model of project management system

The navigation model represents a static navigation view of the content and provides entry and exit points to Web processes. Nodes stereotyped as «navigation class» and «process class» like Project and RemoveProject (Fig. 4) represent information from the content model and use case model (requirements). They are generated by the rules `ContentClass2NavigationClass` and `ProcessIntegration` respectively. Nodes stereotyped as «menu» allow for the selection of a navigation path.

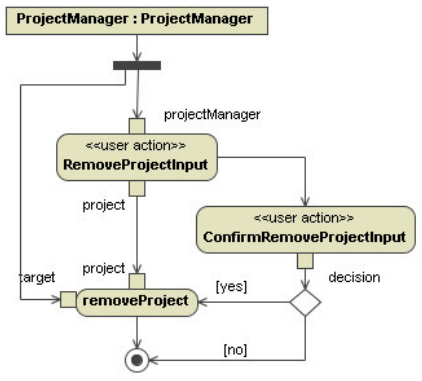


Fig. 5. Process flow for remove project

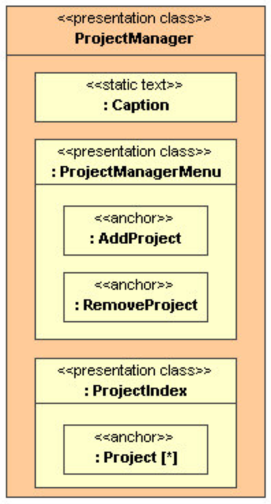


Fig. 6. Presentation model for project manager

Links («navigation link» and «process link») specify the navigation paths between nodes. The transformation rule `CreateProcessDataAndFlow` automatically generates the process data and a draft of the process flow for Web processes. Process flows are represented as UML activity diagram where «user action»s, such as RemoveProjectInput in Fig. 5, are distinguished to support a seamless application of transformation rules. The transformation `NavigationAndProcess2Presentation` automatically derives a presentation model from the navigation model and the process model. For each node in the navigation model and each process class that represents process data a presentation class is constructed and for each attribute a corresponding presentation property is created according to the type of a user interface element. For example a «text» stereotyped class is created for each attribute of type String and a «anchor» stereotyped class is created for an attribute of type URL. An excerpt of the results is shown in Fig. 6.

4 UWE-Based Generation of Web Applications

The main effort of an MDE approach to Web application generation is bridging the gap between the abstractions present in the design models (content, navigation, process, and presentation) and the targeted Web platform. In fact, this gap is quite narrow for UWE content and presentation models: The UML static structure used for describing content in UWE (if we may neglect more arcane modeling concepts like advanced templates, package merging and the like) has a rather direct counterpart in the backing data description techniques of current Web platforms, like relational database models, object-relational mappings, or Java beans. Similarly, an UWE presentation model, being again based on UML static structures, can be seen as a mild abstraction of Web page designs, using, e.g., plain HTML, Dynamic HTML, or Java Server Pages (JSPs). This abstraction gap is comparable for UWE navigation structures using only navigation nodes and access primitives. The situation, however, changes for UWE business process descriptions, which use a workflow notation. In particular, the token-based, Petri net-like interpretation of UML activities and their combination of control and data flow, which is especially well-suited for a declarative data transport, differs notably from the more traditional, control-centric programming language concepts supported by current Web platforms.

Thus, for generating a Web application from an UWE design model, we employ on the one hand a transformational and on the other hand an interpretational approach: Transformation rules are adequate for generating the data model and the presentation layer of a Web application from the UWE content, navigation structure and presentation models, where the differences in modeling and platform abstraction is low. The higher differences in abstraction and formalism apparent in the process models can be more easily overcome interpreting these executable models directly in a virtual machine. For concreteness, we describe the Web application generation process from UWE models by means of a single Web platform, the Spring framework[10], but, by exchanging the model transformations and adapting the virtual machine, the principal ideas could be easily transferred to other Web technologies like using simply Java Server Pages (JSPs) or, more heavy weightily, ASP.NET.

Spring is a multi-purpose framework based on the Java platform modularly integrating an MVC 2-based[11] Web framework with facilities for middleware access, persistence, and transaction management. Its decoupling of the model, view, and controller parts directly reflects the general structure of the UWE modeling approach and also supports the complementary use of transformation rules and an execution engine in Web application generation: Minimal requirements are imposed by the Spring framework on the model technology; in fact, any kind of Plain Old Java Ob-

[10] Spring Framework, http://www.springframework.org/

[11] Sun ONE Architecture Guide, 2002. http://www.sun.com/software/sunone/docs/arch/

jects (POJOs) can be used, and the access to the model from the view or the controller parts only relies on calling `get`- and `set`-methods. We illustrate the approach defining transformation rules from an UWE content model into Java beans. The view technology is separated from the model and the controller part by a configuration mechanism provided by Spring; thus technologies like JSPs, Tiles, or Java Server Faces can be employed. We define transformation rules from an UWE presentation model into JSPs. Finally, the controller part provides a hook for deploying a virtual machine for business process interpretation, as it can be customized through any Java class implementing a specific interface from the Spring framework. Configuration data for the virtual machine are generated from the UWE process and navigation model.

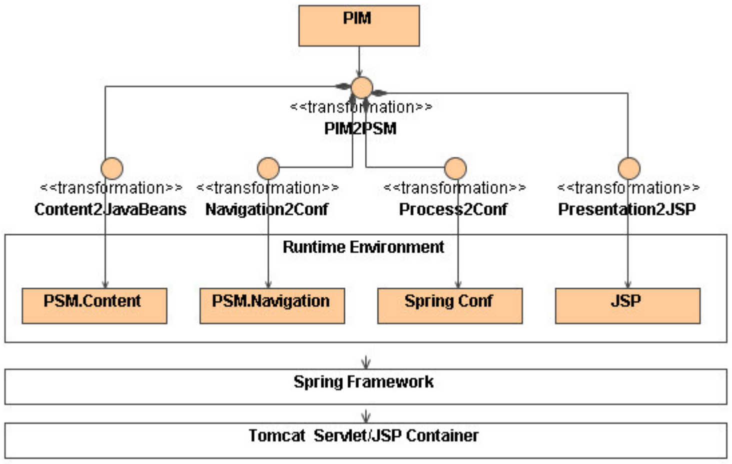


Fig. 7. Overview of platform specific implementation

An overview of the transformations and the execution engine we describe in the following is given in Fig. 7. The virtual machine for business process execution and the integration into navigation is wrapped into a runtime environment that is built on top of the Spring framework and also encapsulates the management of data and the handling of views.

4.1 Runtime Environment

The structure of the runtime environment is shown in Fig. 8. The Spring framework is configured to use a specific generic controller implementation named `MainController`. The controller has access to a set of objects with types generated from the content model and one designated root object as entrance point for the application. Model objects are accessed by their `get`- and `set`- methods and the operations defined in the content model. Additionally, the controller manages a set of `NavigationClassInfo` objects which contain information about the navigation structure regarding inheritance between navigation classes and are generated from the navigation model. A set of `ProcessActivity` objects generated from the process model represents the available Web processes; for each session at most one process can be active at a time.

Views, i.e. Web pages, are not explicitly managed by the runtime environment, only string identifiers are passed. The Spring framework is responsible for resolving these identifiers to actual Web pages that were generated from the presentation model.

The method `handleRequest` handles incoming Web requests by modifying the model and returning a corresponding view. When this method is called, it first checks if a process is active in the current session. If it is, then the execution is delegated to the process runtime environment as detailed in Sect. 4.3. If not, then the next object from the content model that should be presented to the user is resolved by its identifier. Finally, a view identifier is returned and the corresponding Web page is shown to the user.

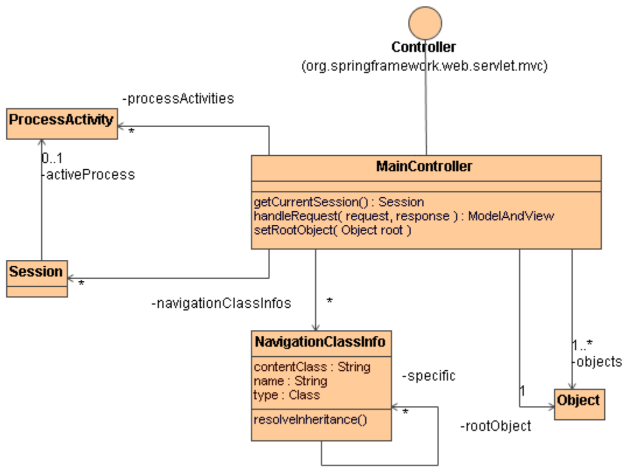


Fig. 8. Runtime environment

4.2 Content and Navigation

The transformation of the content model into Java beans is rather straightforward. The following example illustrates the outcome for the transformation rule applied to ProjectManager:

```
public class ProjectManager {
  private List<Project> projects;
  public List<Project> getProjects() {
    return projects;
  }
  public void setProjects(List<Project> projects) {
    this.projects = projects;
  }
  public void removeProject(Project project) {
    // to be implemented manually
  }
}
```

The navigation model does not have to be directly transformed into code because in the transformation of the presentation model the references to elements from the navigation model are resolved so that the generated pages directly access the content model. Nevertheless, a minimum knowledge about the navigation model is needed in the runtime environment to handle dynamic navigation. For instance, in the navigation model of Fig. 4 a process link leads from the process class AddProject to the abstract navigation class Project with the two navigation sub classes UserProject and ValidationProject. Thus, when following the link from AddProject to a created project then the presentation class for the navigation subclass for the dynamic content object type should be displayed.

4.3 Process

Because of the complex execution semantics of activities based on token flows we integrate a generic Web process engine into the platform specific runtime environment presented in Sect. 4.1. The basic structure of the Web process engine is given in Fig. 9. A process activity comprises a list of activity nodes and set of activity edges. Activity nodes can hold a token which is either a control token, indicating that a flow of control is currently at a specific node, or an object token which indicates that an object flow is at a specific node. Activity edges represent the possible flow of tokens from one activity node to another. Multiple tokens may be present at different activity nodes at a specific point in time. The method `acceptsToken` of an activity node or an activity edge is used to query if a specific token would currently be accepted which then could be received by the method `receiveToken`. An activity has an input parameter node and optionally an output parameter node which serve to hold input and output object tokens.

The control nodes supported by the process engine are decision and merge nodes, join and fork nodes, and final nodes. The object nodes supported are pins representing input and output of actions, activity parameter nodes for the input and output of process activities, central buffer nodes for intermediate buffering of object tokens, and datastore nodes representing a permanent buffer. The implementation of these nodes corresponds to the UML 2.0 specification.

Before starting the execution of a process activity it has to be initialized by calling the method `init`. This results in initializing all contained activity nodes and placing an object token in the input parameter node as illustrated by the following simplified Java code lines:

```
public void init(Object inputParameter) {
  // initialize all activity nodes
  for (ActivityNode n : activityNodes)
    n.init();
  // place new object token in input parameter node
  inputParameterNode.receiveToken(new ObjectToken(inputParameter));
  finished = false;
}
```

The complete execution of a process activity comprises the handling of user interactions, like RemoveProjectInput and ConfirmRemoveProjectInput in Fig. 5. Thus, when a process activity contains at least one user interaction then it cannot be executed completely in one step. The method `next` of a process activity is called from the runtime environment to execute the process activity until the next user interaction is encountered or the process activity has finished its execution. Moreover, either the next user interaction object to be presented to the user is returned; or in case the activity has finished with a return value, the output parameter object is shown. The following code lines give an outline to the implementation of the method `next`:

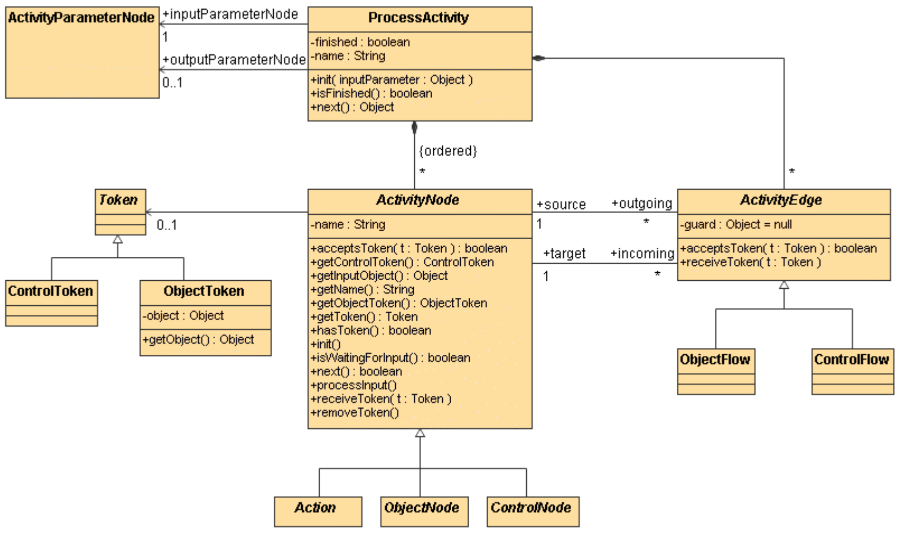


Fig. 9. Runtime process activity

```
public Object next() {
  // process input requested after last method call
  for (ActivityNode n : activityNodes) {
    if (n.isWaitingForInput()) {
      n.processInput();
      break;
    }
  }
  // token passing loop
  while (true) {
    for (ActivityNode n : activityNodes) {
      n.next();
      // return in case of waiting for user input
      if (n.isWaitingForInput())
        return n.getInputObject();
      else {
        // return if the output parameter node has an object token
        if (n == outputParameterNode && n.hasToken())
          return outputParameterNode.getObjectToken().getObject();
        // return in case of activity final node reached
        else
          if (n instanceof ActivityFinalNode && n.hasToken()) {
            return null;
        }
      }
    }
  }
}
```

First the method `processInput` of the first activity node that was waiting for input in the last step is called to process the user input that is now available in the user interaction object. Then all activity nodes are notified to execute their behavior by calling the method `next`. If a node then indicates that it is waiting for input the method returns with the user interaction object returned by this node. If a token arrives either at an activity output parameter node or at an activity final node the execution of the process activity terminates and the method returns.

4.4 Presentation

We outline the transformation from the presentation model to Java Server Pages. The metamodel of JSPs is shown in Fig. 10. For every user interface element of type *X* in the presentation metamodel a corresponding ATL transformation rule *X*`2JSP` is responsible for the transformation of user interface elements of the given type.

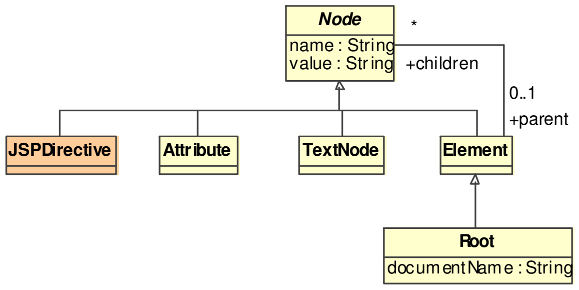


Fig. 10. JSP metamodel

Each presentation class is mapped to a root element that includes the outer structure of an HTML document. All mappings of user interface elements are included in the `body` tag.

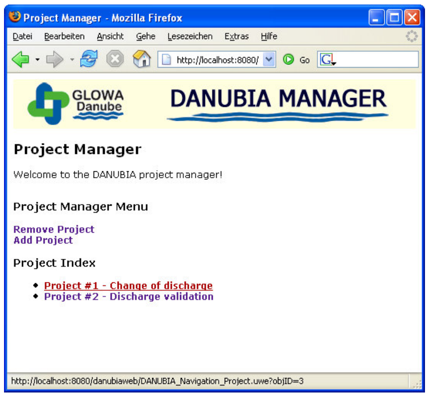
```
rule PresentationClass2JSP {
  from
    pc : UWE!PresentationClass
  to
    jsp : JSP!Root(documentName <- pc.name + '.jsp',
      children <- Sequence{ htmlNode }),
    htmlNode : JSP!Element(name <- 'html',
      children <- Sequence{ headNode, bodyNode }),
    headNode : JSP!Element(name <- 'head',
      children <- Sequence{ titleNode }),
    titleNode : JSP!Element(name <- 'title',
      children <- Sequence{ titleTextNode }),
    titleTextNode : JSP!TextNode(value <- pc.name),
    bodyNode : JSP!Element(name <- 'body',
      children <- Sequence{ pc.ownedAttribute->collect(p | p.type) })
}
```

Each user interface element of type form is mapped to an HTML `form` element. All contained user interface elements are placed inside the `form` element. The `action` attribute points to the target of the corresponding link associated to the form by using a relative page name with the suffix `.uwe`.

```
rule Form2JSP {
  from
```

```
      uie : UWE!Form
    to
      jsp : JSP!Element(name <- 'form',
        children <- Sequence{ actionAttr, uie.uiElements }),
      actionAttr : JSP!Attribute(name <- 'action',
        value <- uie.link.target.name + '.uwe')
  }
```

All remaining UI elements, like anchors, text, images are transformed similarly [12]. The JSP rendering of the presentation class ProjectManager of Fig. 6 is shown in Fig. **11**.


Project Manager - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://localhost:8080/
Go
GLOWA Danube
DANUBIA MANAGER

Project Manager

Welcome to the DANUBIA project manager!

Project Manager Menu

Remove Project
Add Project

Project Index

- Project #1 - Change of discharge
- Project #2 - Discharge validation

http://localhost:8080/danubiaweb/DANUBIA_Navigation_Project.uwe?objID=3

Fig. 11. Screenshot of JSP for the presentation class ProjectManager

5 Related Work

Other methodologies in the Web engineering domain are also introducing model-driven development techniques in their development processes. For example, the Web Software Architecture (WebSA [15]) approach complements other Web design methods, like OO-H and UWE, by providing an additional viewpoint for the architecture of a Web application. Transformations are specified in a proprietary transformation language called UML Profile for Transformations, which is based on QVT.

MIDAS [3] is another model driven approach for Web application development based on the MDA approach. For analysis and design it is based on the content, navigation and presentation models provided by UWE. In contrast to our approach, it relies on object-relational techniques for the implementation of the content aspect and on XML techniques for the implementation of the navigation and presentation aspects. A process aspect is not supported by MIDAS yet.

The Web Markup Language (WebML [5].) is a data-intensive approach that until now does use neither an explicit metamodel nor model transformation languages. The corresponding tool WebRatio internally uses a Document Type Definition (DTD) for storing WebML models and the XML transformation language XSLT for model-to-code transformation. WebML transformation rules are proprietary part of its CASE tool. Schauerhuber et al. present in [17] an approach to semi-automatically transform the WebML DTD specification to a MOF compatible metamodel.

A recent extension of the Object Oriented Web Solution (OOWS [19]) supports business processes by the inclusion of graphical user interfaces elements in their navigation model. The imperative features of QVT, i.e. operational mappings are used as transformation language. In [20] OOWS proposes the use of graph transformations to automate its CIM to PIM transformations. A similar approach is used in W2000 [1]. SHDM [13] and Hera [21]are both methods centered on the Semantic Web. HyperDE – a tool implementing SHDM – is based on Ruby on Rails extended by navigation primitives. Hera instead only applies model-driven engineering for the creation of a model for data integration.

Another interesting model driven approach stems from Muller et al. [16]. In contrast to this work, a heavyweight non-profilable metamodel is used for the hypertext model and the template-based presentation model, nevertheless UML is used for the business model. A language called Xion is used to express constraints and actions. A visual model driven tool called Netsilon supports the whole approach.

6 Conclusions and Future Work

We have presented an MDE approach to the generation of Web applications from UWE design models. On the one hand, model transformation rules in the transformation language ATL translate the UWE content and presentation models into Java beans and JSPs; on the other hand, a virtual machine built on top of the controller of the Spring framework executes the business processes integrated into the navigation structure. These are the first steps towards a "translationist" vision of transformations of platform independent models to platform specific models in UWE.

The combination of a translational and interpretational approach offers a high degree of flexibility to generate Web applications for a broad range of different target technologies. The approach presented in this work is further extended in [12], including a detailed description of computational independent models (CIMs) and platform independent models (PIMs) as well as transformations from CIM to PIM and PIM to PIM, which are also expressed as ATL transformation rules. The ATL transformations of this work are easily transferable to QVT.

Our future work will focus on applying the model transformation approach to other Web applications concerns, such as adaptivity and access control. An aspect-oriented modeling approach is used to model these concerns in UWE and still needs the definition of appropriate transformations. We also plan to analyze the applicability of an MDE approach to Web 2.0 features, e.g. Web services and Rich Internet Applications (RIAs) using AJAX technology in the model-driven development process of UWE.

References

[1] Luciano Baresi, Luca Mainetti. "Beyond Modeling Notations: Consistency and Adaptability of W2000 Models". Proc. ACM Symp. Applied Computing (SAC'05), Santa Fe, 2005.

[2] Michael Barth, Rolf Hennicker, Andreas Kraus, Matthias Ludwig. "DANUBIA: An Integrative Simulation System for Global Change Research in the Upper Danube Basin". Cybernetics and Systems, Vol. 35, No. 7-8, 2004, pp. 639-666.

[3] Paloma Cáceres, Valeria de Castro, Juan M. Vara, Esperanza Marcos. "Model transformation for Hypertext Modeling on Web Information Systems", Proc. ACM Symp. Applied Computing (SAC'06), Dijon, 2006.

[4] Krzysztof Czarnecki, Simon Helsen. "Classification of Model Transformation Approaches". Proc. OOPSLA'03 Wsh. Generative Techniques in the Context of Model-Driven Architecture, Anaheim, 2003.

[5] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera. "Designing Data-Intensive Web Applications". Morgan Kaufman, 2003.

[6] Jaime Gómez, Cristina Cachero. "OO-H: Extending UML to Model Web Interfaces". Information Modeling for Internet Applications. IGI Publishing, 2002.

[7] Ivar Jacobson, Grady Booch, Jim Rumbaugh. "The Unified Software Development Process". Addison Wesley, 1999.

[8] Gerti Kappel, Birgit Pröll, Siegfried Reich, Werner Retschizegger (eds.). "Web Engineering", John Wiley, 2006.

[9] Nora Koch. "Transformations Techniques in the Model-Driven Development Process of UWE". Proc. 2nd Wsh. Model-Driven Web Engineering (MDWE'06), Palo Alto, 2006.

[10] Alexander Knapp, Gefei Zhang. "Model Transformations for Integrating and Validating Web Application Models". Proc. Modellierung 2006 (MOD'06). LNI P-82, pp. 115-128, 2006.

[11] Andreas Kraus, Nora Koch. A Metamodel for UWE. Technical Report 0301, Ludwig-Maximilians-Universität München, Germany, 2003.

[12] Andreas Kraus. "Model Driven Software Engineering for Web Applications", PhD. Thesis, Ludwig-Maximilians-Universität München, Germany, 2007, to appear.

[13] Fernanda Lima, Daniel Schwabe. "Application Modeling for the Semantic Web". Proc. LA-Web 2003, Santiago, IEEE Press, pp. 93-103, 2003.

[14] Ashley McNeile. MDA: The Vision with the Hole? http://www.metamaxim.com/download/documents/MDAv1.pdf, 2003.

[15] Santiago Meliá, Jaime Gomez. "The WebSA Approach: Applying Model Driven Engineering to Web Applications". J. Web Engineering, 5(2), 2006.

[16] Pierre-Alain Muller, Philippe Studer, Frederic Fondement, Jean Bézivin. "Platform independent Web application modeling and development with Netsilon". Software & System Modeling, 4(4), 2005.

[17] Andrea Schauerhuber, Manuel Wimmer, Elisabeth Kapsammer. "Bridging existing Web Modeling Languages to Model-Driven Engineering: A Metamodel for WebML", In: Proc. 2nd Wsh. Model-Driven Web Engineering (MDWE'06), Palo Alto, 2006.

[18] Douglas Schmidt. "Model-Driven Engineering". IEEE Computer 39 (2), 2006.

[19] Victoria Torres, Vicente Pelechano, Pau Giner. "Generación de Aplicaciones Web basadas en Procesos de Negocio mediante Transformación de Modelos". Jornadas de Ingeniería de Software y Base de Datos (JISBD), XI, Barcelona, Spain, 2006.

[20] Pedro Valderas, Joan Fons, Vicente Pelechano. "From Web Requirements to Navigational Design – A Transformational Approach". Proc. 5th Int. Conf. Web Engineering (ICWE'05). LNCS 3579, 2005.

[21] Richard Vdovjak, Geert-Jan Houben. "A Model-Driven Approach for Designing Distributed Web Information Systems". Proc. 5th Int. Conf. Web Engineering (ICWE'05). LNCS 3579, 2005.