

# ArgoUWE: A CASE Tool for Web Applications

Alexander Knapp, Nora Koch, Flavia Moser and Gefei Zhang<sup>1</sup>

Ludwig-Maximilians-Universität München, Germany  
{knapp,kochn,moser,zhangg@informatik.uni-muenchen.de}

**Abstract.** The UWE methodology provides a systematic approach for the development of Web applications. UWE is based on a conservative extension of the UML and comprises the separate modeling of the conceptual, navigational and presentational aspects of Web applications. We present the CASE tool ArgoUWE to support the design phase of the UWE development process. It is implemented as a plugin module of the open source ArgoUML modeling tool. ArgoUWE fully integrates the UWE metamodel and provides an XMI extension. The construction process of Web applications is supported by incorporating the semi-automatic UWE development steps as well as the OCL well-formedness rules of the UWE metamodel that allow the designer to check the consistency of the UWE models during editing. ArgoUWE is part of the OpenUWE tool environment for model-driven generation of Web applications.

**Keywords.** CASE Tools for IS Design and Implementation, Web Design, Web Engineering, UML, OCL

## 1 Introduction

The Web Engineering field is rich in design methods supporting the complex task of designing Web applications. From our point of view the usability requirements to such methods are the following: to be based on standards, to define a process for the systematic development of Web applications and to provide tool support for the model-driven design and generation of Web applications. The well-known standard used for modeling is the Unified Modeling Language [UML 2003].

Most of the existing Web engineering methods fulfill some of these usability requirements, but not all of them. Interesting approaches for the systematic development supported by CASE-tools are those for the method OO-H process [Gomez et al. 2001] and for the modeling language WebML [Ceri et al. 2002]. Conallen [2003] proposes an extension of UML for a more architecture-oriented and implementation-based approach.

The main focus of our UML-based Web Engineering (UWE) methodology is to stick to the use of standards in the systematic design followed by a semi-automatic generation of Web applications fulfilling this way as close as possible the usability requirements we enumerated above. First, as indicated by its name, UWE is UML compliant. Second, UWE defines a systematic development process that can be performed semi-automatically. Third, the tool support is guaranteed by the OpenUWE model-driven development environment that comprises at the current implementation state two CASE tools: ArgoUWE to aid the design and UWEXML to generate Web applications automatically.

The focus of this work is the presentation of the tool ArgoUWE<sup>2</sup> describing the underlying UWE concepts, the functionality provided to the users of this tool and its architecture. The complete description of the UWE notation and the UWE process is not within the scope of this article, but can be found in [Koch & Kraus 2002]. The UWE methodology covers structure modeling as well as behavior modeling of Web applications. ArgoUWE, however, until now provides support to structural modeling only, therefore we limit ourselves to the presentation of these aspects.

---

<sup>1</sup> This work has been partially supported by the European Union within the IST project AGILE (IST-2001-32747), the DFG project InOpSys (WI 841/6-1) and the BMBF project MMISS (08NM070D).

<sup>2</sup> <http://www.pst.informatik.uni-muenchen.de/projekte/argouwe>

ArgoUWE is built as a conservative extension of a plugin module for ArgoUML<sup>3</sup>. The main advantage we see in an ArgoUML-based tool is the fact that it is an open source tool that provides a module plugin concept. To remark is that metamodeling plays a fundamental role in CASE tool construction and is also the core of the automatic generation. We have defined an easily extendible metamodel for the UWE methodology ([Zhang 2002], [Kraus & Koch 2003]) as a conservative extension of the UML metamodel (version 1.5). The goal to stay thereby compatible with the MOF interchange metamodel is to take advantage on the use of the corresponding XMI interchange format. The resulting UWE metamodel is *profileable*, which means that it is possible to map the metamodel to a UML profile. Moreover, it is easy to integrate into ArgoUML.

The reminder of this paper is structured as follows: Section 2 provides an overview of how the UWE methodology supports the development of Web applications with focus on the systematic design. Section 3 describes the UWE metamodel which is the basis of the ArgoUWE CASE tool. Section 4 and Section 5, which are the core of this work, present the functionality and the architecture of ArgoUWE, respectively. Finally, in the last section some concluding remarks and future work are outlined.

## 2 Developing Web Applications with UWE

The UML-based Web Engineering (UWE) is a software engineering approach for the development of Web applications that is continuously extended since 1999 [Baumeister et al. 1999; Koch & Kraus 2003]. UWE supports Web application development with special focus on systematization [Koch & Kraus 2002]. Being a software engineering approach it requires three pillars to be based on: a process, a notation and tool-support. The focus of this article is the tool-support, we restrict ourselves to give in this section a brief overview of the notation and the process

The UWE process is object-oriented, iterative and incremental. It is based on the Unified Software Development Process [Jacobson et al. 1999] and covers the whole life-cycle of Web applications focusing on design and automatic generation [Koch & Kraus 2003].

The UWE notation used for the analysis and design of Web applications is a *lightweight* UML profile [UML 2003] developed in various previous works. Such a profile is a UML extension based on the extension mechanisms defined by the UML itself, i.e. it only includes stereotypes, tagged values and constraints. These modeling elements are used in the design of the conceptual model, the navigation structure and the presentation aspects of Web applications, as it is shown in Section 2.1. The UWE methodology provides guidelines for the systematic and stepwise construction of models. The precision can be augmented by the definition of constraints in the Object Constraint Language (OCL [Warmer & Kleppe 1999]) of the UML. The core modeling activities are the requirements analysis, conceptual, navigation and presentation design.

The goal of a systematic design is to support the modeler with a clear design process defined step by step and with as many automatic generation steps as possible. Although some automation is possible, there are other steps that can not be automated because they are application specific or based on design decisions depending on the designers experience and knowledge. In the following we describe the UWE steps for developing design models of Web applications. These main steps – outlined in Sections 2.1 to 2.3 – are based on the clear separation of concerns by Web applications which are conceptual modeling, navigation modeling and presentation modeling.

The tool support that we achieve with OpenUWE – as already said in the introduction – is twofold. On the one hand, a CASE tool to support the design of Web applications using the UWE notation and methodology is realized by ArgoUWE. On the other hand, the semi-automatic generation of Web applications from the models (built with ArgoUWE or any other CASE tool that provides an XMI interface) is supported by UWEXML using a model-driven Code Generator for deployment to an

---

<sup>3</sup> <http://argouml.tigris.org>

XML publishing framework. A detailed description of UWEXML is not within the scope of this paper. For further details see Koch & Kraus [2003].

As a running example to illustrate how the main UWE design models are built, we use a Conference Review Management application – conference example for short. This application offers conference organizers, authors and reviewers information about the conference, submitted papers and corresponding reviews. The conference example application allows authors to submit papers and reviewers to provide an online evaluation of the papers.

## 2.1 Conceptual Modeling

UWE proposes the use of UML use cases and activity diagrams for capturing the requirements [Koch & Kraus 2002]. A conceptual model includes those objects needed to support the functionality the system will offer to the users. The conceptual design aims to build a conceptual model, which attempts to ignore as many of the navigation paths, presentation and interaction aspects as possible. These aspects are postponed to the steps of the navigation and presentation modeling. The main UML modeling elements used in the conceptual model are: class, association and package. These are represented graphically using the UML notation [UML 2003]. Figure 1 shows the conceptual model for the Conference example (upper left).

## 2.2 Navigation Modeling

Navigation design activities comprise the specification of which objects can be visited by navigation through the Web application and how these objects can be reached through access structures. UWE proposes a set of guidelines and semi-automatic mechanisms for modeling the navigation of an application [Koch & Kraus 2002]. Figure 1 (upper right) shows the result of the semi-automatic generation of the navigation model for the conference example. The navigation relevant classes of the conceptual model are transformed into navigation classes, the associations into navigation links, such as navigation class *Conference* and the navigation link between class *Conference* and class *Paper*. Instead, the navigation links for accepted and rejected papers as well as the navigation link between *Review* and *Paper* have been added manually.

The main modeling elements are the stereotyped class «navigation class» and the stereotyped association «direct navigability». These are the pendant to page (node) and link in the Web terminology. The access elements defined by UWE are indexes, guided tours, queries and menus. The stereotyped classes for the access elements are «index», «guided tour», «query» and «menu». All modeling elements and their corresponding stereotypes and associated icons are defined in Baumeister et al. [1999]. Figure 1 (lower right) shows the navigation model after been enriched with the access structures. The second navigation model can be generated automatically based on the first one using some default decisions. Afterwards, the designer can perform as many changes as considered necessary.

Note that only those classes of the conceptual model that are relevant for navigation are included in the navigation model, as shown in Figure 1. Although information of the omitted classes may be kept as attributes of other navigation classes (e.g. the newly introduced attribute *keyword* of the navigation class *Paper*); OCL constraints are used to express the relationship between conceptual classes and navigation classes or attributes of navigation classes.

## 2.3 Presentation Modeling

The presentation model describes where and how navigation objects and access primitives will be presented to the user. Presentation design supports the transformation of the navigation structure model in a set of models that show the static location of the objects visible to the user, i.e. a schematic representation of these objects (sketches of the pages). The production of sketches of this kind is often helpful in early discussions with the customer.

UWE proposes a set of stereotyped modeling elements to describe the abstract user interface, such as «text», «form», «button», «image», «audio», «anchor», «collection» and «anchored collection». The classes «collection» and «anchored collection» provide a convenient representation of frequently used composites. Anchor and form are the basic interactive elements. An anchor is always associated with a link for navigation. Through a form a user interacts with the Web application supplying information and triggering a submission event [Baumeister et al. 1999]. Figure 1 (lower left) depicts the presentation sketch of a publication.

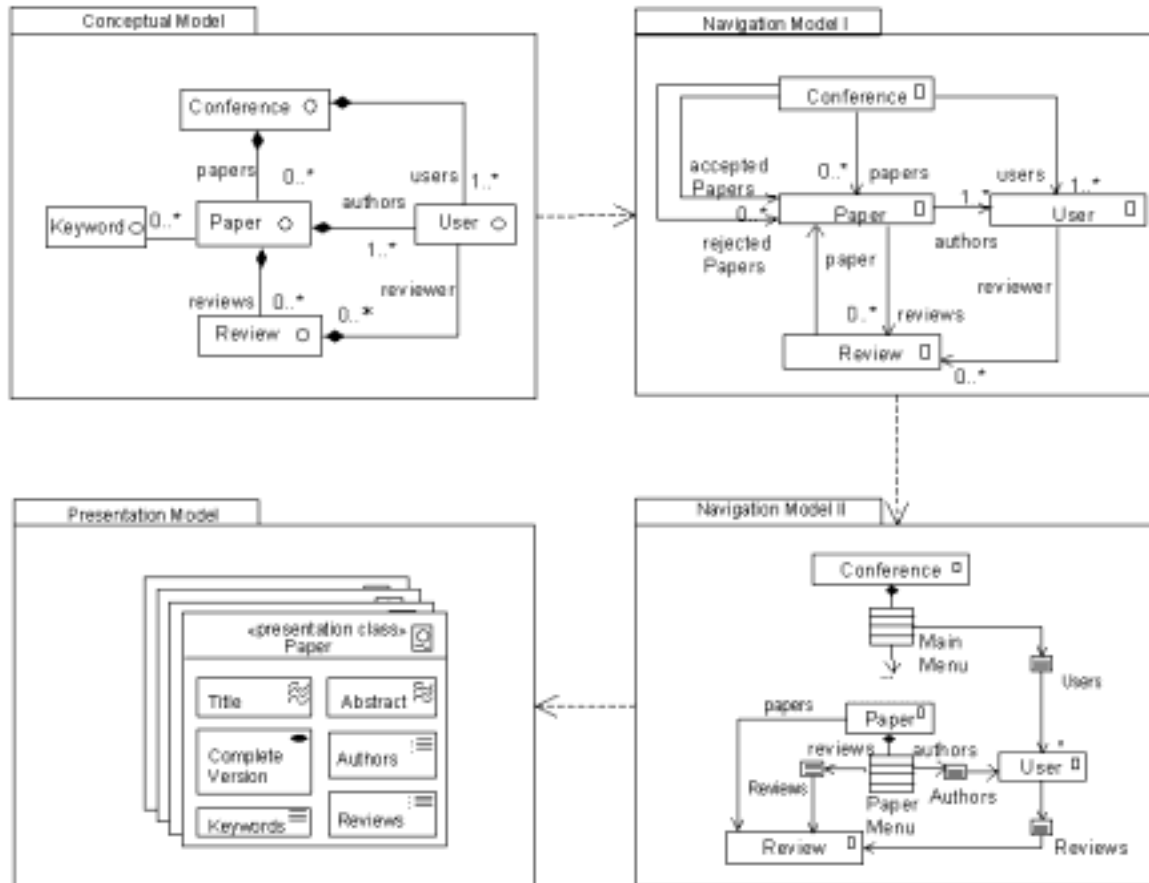


Figure 1 The UWE Design Process for the Conference Review Example

### 3 UWE Metamodel

The UWE metamodel is designed as a conservative extension of the UML metamodel (version 1.5). Conservative means that the modeling elements of the UML metamodel are not modified. Instead, all new modeling elements of the UWE metamodel are related by inheritance to at least one modeling element of the UML metamodel. We define for the new elements additional features and relationships. In addition, analogous to the well-formedness rules in the UML specification, we use OCL constraints to specify the additional static semantics of these new elements. We present here only an overview, for further details see Koch & Kraus [2003].

By staying thereby compatible with the MOF interchange metamodel we can take advantage of metamodeling tools that base on the corresponding XML interchange format XMI. The resulting UWE metamodel is profileable [Baresi et al. 2002], which means that it is possible to map the metamodel to a UML profile. Thus standard UML CASE tools with support for UML profiles or the UML extension mechanisms, i.e. stereotypes, tagged values and OCL constraints can be used to create the UWE models of Web applications. If technically possible these CASE tools can further be extended to support the UWE method. ArgoUWE presents an instance of such CASE tool support for UWE based on the UWE metamodel.

### 3.1 The UWE Package Structure

All UWE modeling elements are contained within one top-level package *UWE* which is added to the three UML top-level packages. The structure of the packages inside the UWE package depicted in Figure 2 is analogous to the UML top-level package structure (shown in grey). The package *Foundation* contains all basic static modeling elements, the package *Behavioral Elements* depends on it and contains all elements for behavioral modeling and finally the package *Model Management* which also depends on the *Foundation* package contains all elements to describe the models themselves specific to UWE. These UWE packages depend on the corresponding UML top-level packages. Note that the separation of concerns of Web applications is represented by the package structure of the UWE metamodel.

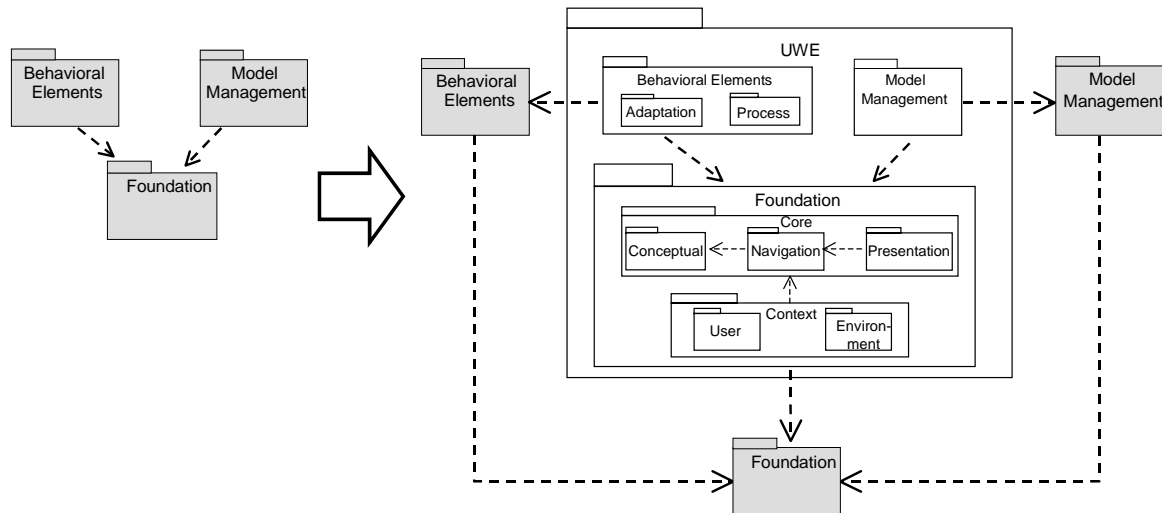


Figure 2 Embedding the UWE Metamodel into the UML Metamodel

The package *Foundation* contains all basic static modeling elements and is further structured in the *Core* and the *Context* packages (see Figure 2). The former contains packages for the core (static) modeling elements for the basic aspects of Web applications which are the conceptual, the navigation and the presentation aspects. The latter depends on the *Core* package and contains further sub-packages for modeling the user and the environment context.

The package *Behavioral Elements* depends on the *Foundation* package and consists of the two sub-packages *Process* and *Adaptation* that comprise modeling elements for the workflow and personalization aspects of a Web application, respectively. Finally, the package *Model Management* which also depends on the *Foundation* package contains all elements to describe the models specific to UWE, such as conceptual, navigation and presentation models.

The basic elements in navigation models are nodes and links. The corresponding modeling elements in the UWE metamodel are *NavigationNode* and *Link* (not to be confused with *Link* of the UML package *Common Behavior*) which are derived from the UML *Core* elements *Class* and *Association*, respectively. This navigation modeling elements are shown in Figure 3. The *NavigationNode* metaclass is abstract which means that only further specialized classes may be instantiated; furthermore it can be designated to be a node which is directly reachable from all other nodes of the application with the *isLandmark* attribute.

Figure 3 also shows the connection between navigation and conceptual objects. A *NavigationClass* is derived from the *ConceptualClass* at the association end with the role name *derivedFrom* – there can exist several navigation views on a conceptual class. The *NavigationClass* consists of *NavigationAttributes* (derived from the UML *Core* element *Attribute*) which are themselves derived from *ConceptualAttributes*. Figure 3 illustrates as well how access primitive classes, such as *Index*, are aggregated to navigation links with an association of type composition. Note that *Menu* is a specialization of class *NavigationNode*.

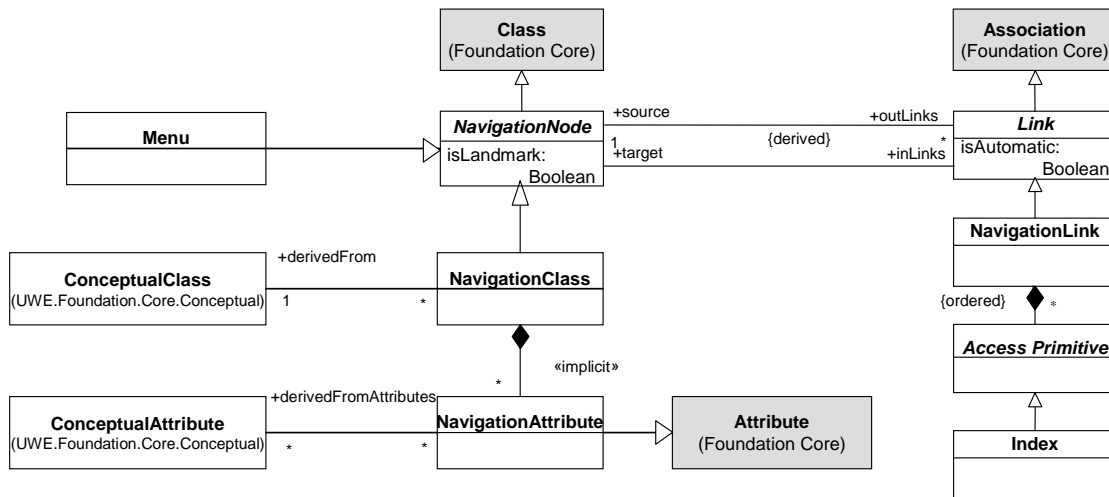


Figure 3 Connection between Navigation and Conceptual Package

### 3.2 Well-formedness Rules

Just like in the UML our UWE metamodel is subject to some well-formedness rules that we have formulated in OCL. The following are examples of the OCL constraints that are part of the UWE metamodel.

The first constraint expresses that a conceptual class contains only conceptual operations and/or conceptual attributes.

```

context ConceptualClass (1)
inv: self.feature->forAll(
    f | f.ocIsTypeOf(ConceptualOperation) or
    f.ocIsTypeOf(ConceptualAttribute))

```

The second constraint forbids the existence of “isolated” navigation nodes.

```

context NavigationNode (2)
inv: self.outLinks->size() + self.inLinks->size() > 0

```

The third constraint expresses that if a navigation node is landmarked, then it is directly reachable from every other navigation node.

```

context NavigationModel (3)
def: ownedNavigationNode : Bag(NavigationNode) =
    self.ownedElement->
        select (n | n.ocIsTypeOf(NavigationNode))->
            collect (n | n.ocAsType(NavigationNode))
inv: self.ownedNavigationNode ->
    forAll (n1 | n1.isLandmark implies
        self.ownedNavigationNode->
            forAll (n2 | n2.outLinks->exists (a | a.target = n1)))

```

## 4 Modeling with ArgoUWE

ArgoUWE is based on ArgoUML and makes use of the graphical user interface of ArgoUML. We introduce new types of diagrams to represent the new, UWE specific models. In these diagrams, users

can add, remove, copy and paste model elements as well as replace their figures and edit their properties just as they are used to in ArgoUML.

As shown in Figure 4, ArgoUWE inherits the four compartments of the ArgoUML *Project Browser*:

1. the *navigator pane*, in which all diagrams and model elements of the model are listed in a tree structure;
2. the *multieditor pane*, which is the main pane of ArgoUWE and where the diagrams are depicted and can be edited,
3. the *critique pane*, where a list of design critique issues are shown; and
4. the *detail pane*, where the attributes of the currently selected model element can be edited.

ArgoUWE exports user models using standard XMI. In the exported XMI UWE specific model elements are labeled with special tagged values.

In the following we illustrate how to use ArgoUWE to model Web applications by means of the Conference Review example (see Section 2).

#### 4.1 Starting with the Conceptual Model

In ArgoUWE a conceptual model is represented as a conventional UML class diagram. The standard procedure to model class diagrams is not changed. For the UWE process, however, the modeler can mark some conceptual classes as “navigation relevant”, i.e. these classes shall be connected to navigation classes that represent the nodes of the Web application structure.

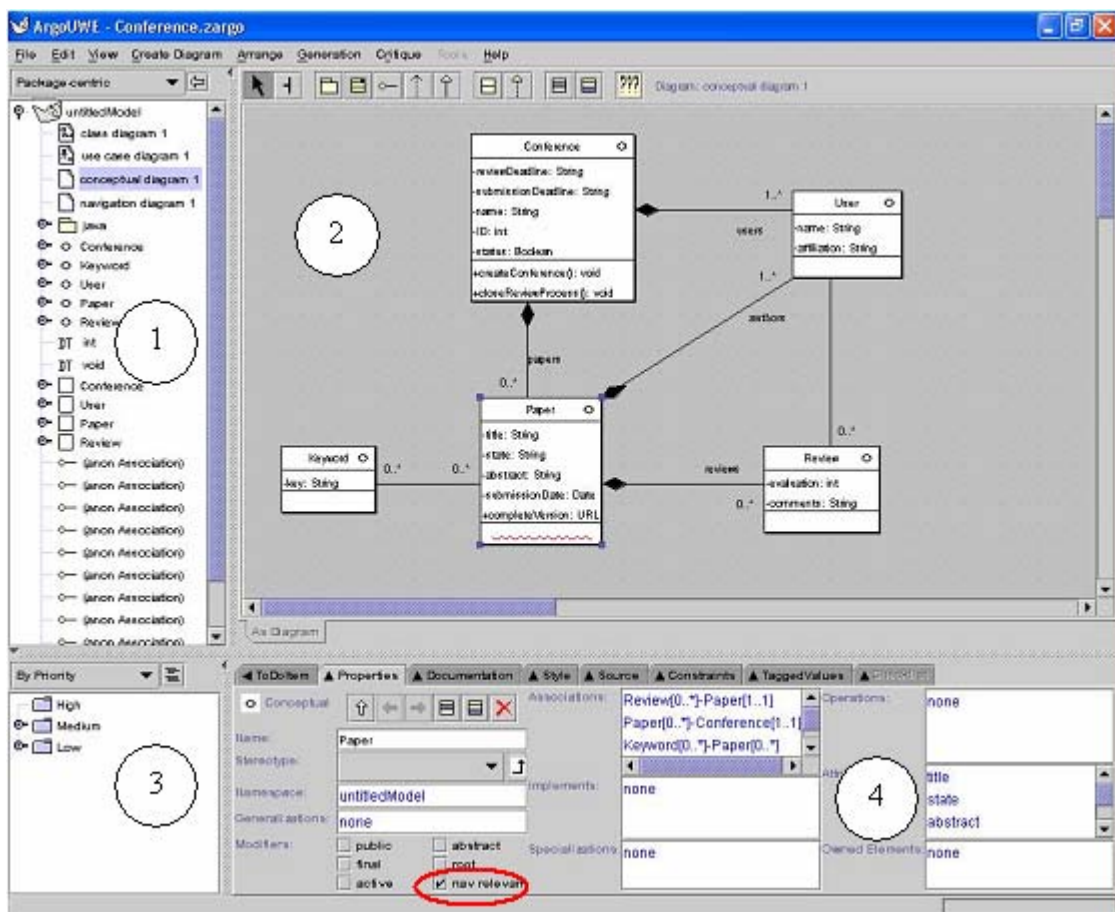


Figure 4 Conceptual Model for the Conference Example

Figure 4 shows the conceptual model of our Conference Review example. The conceptual class *Paper* has been identified as navigation relevant (see the encircled checkbox at the bottom of the screenshot).

## 4.2 Building the Navigation Model

In ArgoUWE a navigation model can be built semi-automatically based on the conceptual classes marked as navigation relevant by the modeler. When the designer selects Create Diagram | Navigation Diagram (in the menu line of the ArgoUML main window, see Figure 4), ArgoUWE copies all navigation relevant conceptual classes and all associations between them from the conceptual model to a new navigation model. This mechanism not only relieves the designer from copying conceptual classes one by one manually but also keeps the model consistent.

After automatic creation of the navigation diagram, the modeler can add some additional associations designating direct navigability from one navigation class to another. Figure 5 shows the result of this editing process. Instead of a single association between *Conference* and *Paper* the user now has separate links to an overview of all accepted papers and an overview of all rejected papers. The class *Keyword* has not been selected as navigation relevant and therefore no navigation class is created for it. Note that Figure 5 corresponds to Navigation Model I in Figure 1 (upper right).

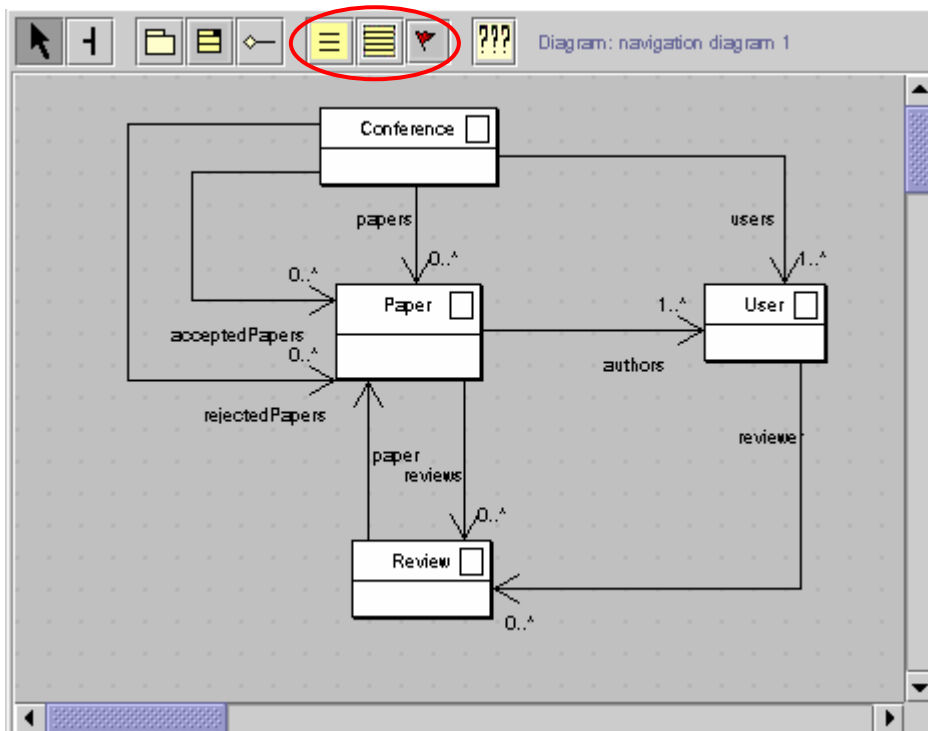


Figure 5 Navigation Model for the Conference Example (1)

The next step is to add menus between navigation classes and access primitives. The user can add indexes, menus, and associations to landmarked nodes automatically (encircled buttons from left to right):

- Indexes are added by the tool between two classes related by an association whenever the multiplicity on the target end is greater than one.
- Menus are added by the tool to every class that has more than a single outgoing association. Menus are included by composition.
- Associations to classes selected as landmarks are added by the tool at every navigation node from which the landmarked node cannot already be directly reached.

Additionally queries for explicit searches can be added by the modeler at any place he likes. The result of adding indexes, menus, queries and associations is shown in Figure 6, which corresponds to Figure 1 (lower right).

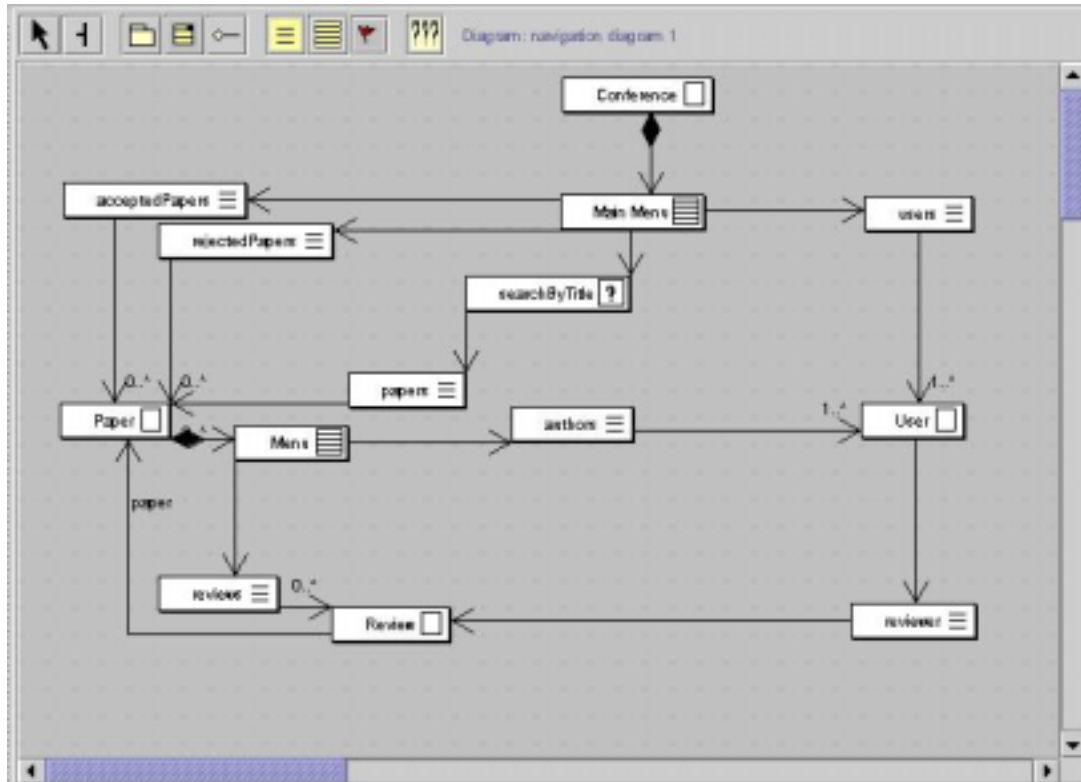


Figure 6 Navigation Model for the Conference Example (2)

### 4.3 Completing with the Presentation Model

The building process of a presentation model from the navigation model is similar to building a navigation model from the conceptual model and is triggered by choosing the menu item Create Diagram | Presentation Diagram. All navigation nodes are copied into the new class diagram for the presentation. Moreover, ArgoUWE creates for each attribute in a navigation class automatically a presentation element as well as an association of type composition to its owner presentation class.

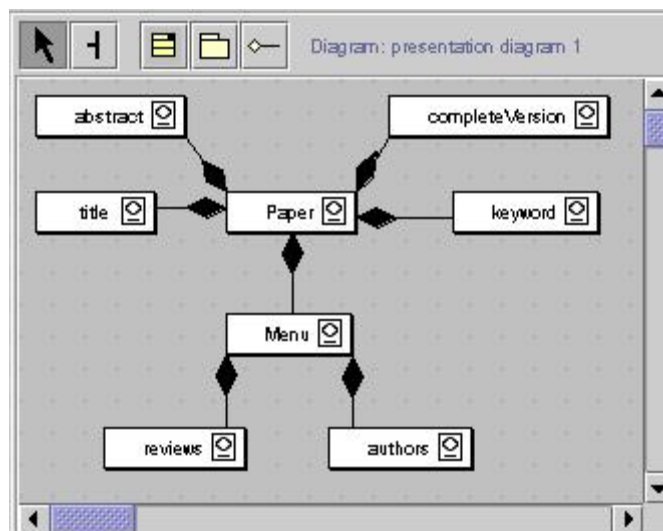


Figure 7 Partial Presentation Model for the Conference Example

Additionally, for each menu a presentation class is created and the composition between the menu and its owner navigation class is copied. A part of the created presentation diagram of our Conference example is shown in Figure 7. Note that in ArgoUWE the nested representation as shown in Figure 1 (lower left) of composition is currently not possible.

#### 4.4 Consistency Checking

ArgoUWE helps the modeler keep the models consistent. Some of the constraints of the UWE metamodel (see Section 3) are constantly enforced by ArgoUWE. For instance, in the conceptual diagram, since only conceptual operations and conceptual attributes are available, the user cannot create any operation and any attribute but the conceptual ones, and thus constraint (1) in Section 3.2 can never be violated. On the other hand, there are also constraints that simply cannot be enforced continually during modeling. For example, at the moment where a new navigation node (e.g. a navigation class) is created, ArgoUWE must accept for a while that it is neither landmarked nor directly reachable from all other navigation nodes and therefore constraint (3) in Section 3.2 is violated. However, as soon as the user triggers ArgoUWE's consistency check of the current models by pressing the "???"-button (encircled in Figure 8), the user will be warned about this violation.

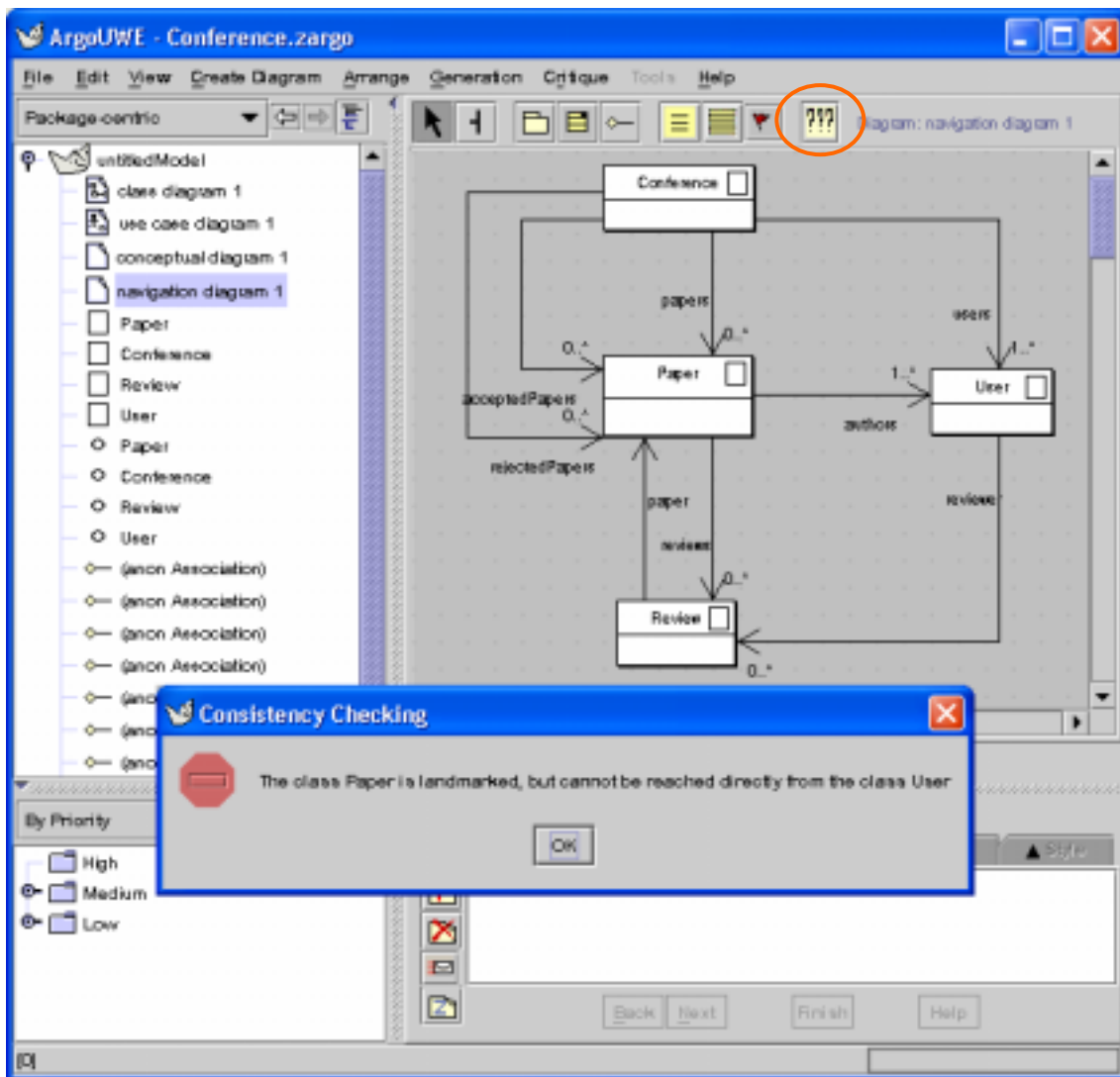


Figure 8 Constraint Violation

If we landmark the navigation class *Paper*, the navigation model shown in Figure 5 would be inconsistent because *Paper* cannot be reached from the class *User* directly. A warning of that constraint violation is shown in Figure 8.

## 5 Architecture of ArgoUWE

The ArgoUWE tool is implemented as a plugin into the open-source UML modeling tool ArgoUML (version 0.10), both written in Java. ArgoUML provides a suitable basis for an extension with UWE tool support by being based on a flexible UML metamodel library (NSUML<sup>4</sup>, version 1.3/0.4.20) and a general graph editing framework (GEF<sup>5</sup>, version 0.95), as well as featuring an extendable module architecture. Not only these feature characteristics but also the fact that ArgoUML is an open-source tool with an active developer community lead us to favoring ArgoUML as development basis over other, commercial tools — although the open source of ArgoUML has sometimes to outweigh its rather poor documentation. However, tools like Rational Rose™<sup>6</sup> or Gentleware’s Poseidon™<sup>7</sup> would also afford the necessary extension prerequisites, perhaps with the exception of metamodel extensions.

### 5.1 ArgoUWE Metamodel

The “Novosoft UML library” (NSUML), on which ArgoUML is based, not only provides a library for working with UML 1.3 models in terms of Java objects, but also contains an XML-based generator for arbitrarily changed and extended (UML) metamodels. As UWE uses additional modeling concepts targeted onto Web applications, ArgoUWE uses NSUML to generate an extended UML/UWE metamodel that again allows the programmer to handle UWE entities in a seamless and straightforward manner. In particular, we chose a “heavyweight extension” for the physical metamodel that is generated by NSUML. Alternatively, we could employ the UWE lightweight UML profile directly. However, stereotyping and tagging is not compatible with the concept of overloading in object-oriented programming. For the current ArgoUML versions, the adaptations of the UML metamodel merely consist of extending the NSUML generator resource files by the UWE metaclasses *ConceptualClass*, *NavigationClass*, *PresentationClass*, etc.

However, the more recent versions of the Novosoft UML library, starting with 1.4/0.13, replace the proprietary XML metamodel description by a standardized “Meta Object Facility” (MOF) input. The libraries generated with NSUML 1.4/0.13 (for UML 1.4) are code-incompatible with the NSUML 1.3/0.4.20 (for UML 1.3) libraries and thus cannot be used in ArgoUML directly yet.

### 5.2 Plugin Architecture

In ArgoUML as of version 0.10, the model is encapsulated in an instance of the (extended) NSUML library class `ru.novosoft.uml.model_management.MModel`. Thus, manipulations of the model have a single access point, all effects of model manipulations are disseminated to other components by a general observer mechanism following the Model-View-Controller paradigm. Figure 9 summarizes the general structure of the ArgoUML model, view, and control devices as used in ArgoUWE.

The UWE diagram kinds: conceptual diagram, navigation diagram, and presentation diagram are straightforwardly supported by introducing new subclasses of `org.argouml.uml.diagram.ui.UMLDiagram`, more specifically the common superclass `org.argouml.uml.diagram.ui.UWEDiagram`. A `UMLDiagram` captures the graphical presentation and manipulation of model elements. It is based on a bridge, the

---

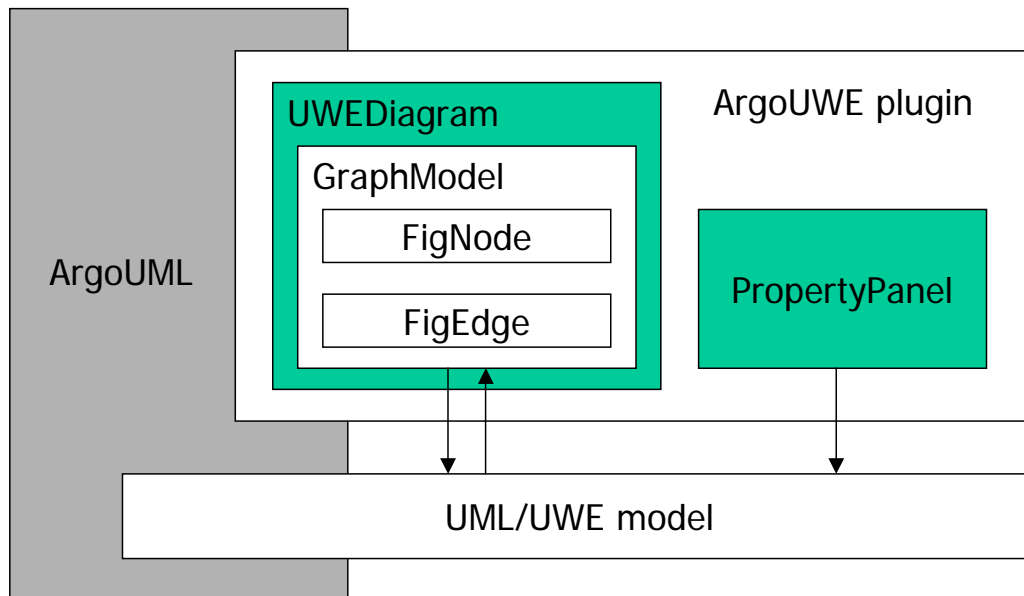
<sup>4</sup> <http://nsuml.tigris.org>

<sup>5</sup> <http://gef.tigris.org>

<sup>6</sup> <http://www.rational.com>

<sup>7</sup> <http://www.gentleware.com>

graph model, between the model realized by NSUML library classes and the graphical presentation using the “Graph Editing Framework” (GEF) library classes. The bridges for UWE are all derived from `org.argouml.uml.diagram.static_structure.ClassDiagramGraphModel`. Each UWE model element is linked to a figure node (`org.tigris.gef.presentation.FigNode`) or figure edge (`org.tigris.gef.presentation.FigEdge`) of the GEF library. In addition to manipulating the model elements graphically, they can also be changed and edited by using property panels that are implemented as subclasses of `org.argouml.uml.ui.PropPanel`. ArgoUWE adds property panels for conceptual classes, navigation classes, etc., which are installed on the ArgoUML platform automatically with the diagram counterparts by reflection.



**Figure 9 Overview of the ArgoUML plugin architecture**

The UWE model consistency checks and the semi-automatic editing functionalities induced by the UWE method (see Section 4) are triggered in a `UWEDialog` (see Section 3). In particular, well-formedness of a UWE model is not enforced for every model change; during editing some inconsistencies may be inevitable or at least tolerable.

The UWE extensions are packaged in a plugin module. The original ArgoUML user interface is extended by a class implementing `org.argouml.application.api.PluggableMenu`, registering the new diagram types and their support. This extension, when put into the extension directory (`./ext`) of ArgoUML, is loaded automatically on ArgoUML start-up. However, it must be noted that the UWE extension of ArgoUML is not completely orthogonal to ArgoUML as the underlying metamodel has been changed. Nevertheless, packaging the UWE extensions into a plugin module paves the way for growing the extensions towards the development of new ArgoUML versions.

## 6 Related Work

Many methods for the development of Web applications have been proposed since the middle of the nineties. An excellent overview is presented in Schwabe [2001] where the most relevant methods, such as OOHD [Rossi et al. 2001], OO-H [Gomez et al. 2001], WSDM [De Troyer et al. 2001], W2000 [Garzotto et al. 2001] and UWE [Koch et al. 2001] are described on the basis of a same case study. Only some of them have implemented a CASE-tool supporting the systematic development.

The most advanced tool-support is offered for the method OO-H the modeling language WebML. VisualWADE is the tool supporting the OO-H method that includes a set of model compilers to provide automatic code generation capabilities and rapid prototyping. In contrast to our ArgoUWE, it uses the UML [UML 2003] only in the first phase of the development process. WebRatio is based on

the proprietary Web modeling language WebML and supports code generation technology built on XSL [Ceri et al. 2002]. This approach differs from ours as it does not perform a clear separation of the navigation and presentation aspects. A more architecture-oriented approach is proposed by Conallen [2003]. It extends the UML to support the design of Web applications focusing on current technological aspects of the implementation and is based on the generic RUP development process. The notation is supported by the Rational Rose™ tool, but conversely to ArgoUWE it neither supports the systematic development process nor guides the developer through the process.

## 7 Conclusions and Future Work

We have presented the CASE tool ArgoUWE that we have developed for the computer aided design of Web applications using the UWE methodology. ArgoUWE is built as a flexible extension of ArgoUML due to the plugin architecture facilities provided by the ArgoUML tool (version 0.10). We stress that the core of the CASE tool is the underlying UWE metamodel defined as a conservative extension of the UML metamodel.

We outlined in this work the basic ideas behind the UWE methodology and how ArgoUWE is integrated in the OpenUWE tool suite environment to achieve a model-driven generation of such Web applications. In addition we presented a running example to show how the tool supports the design of the three main UWE models: conceptual, navigation, and presentation models in a semi-automatic process where user and tool-activities are interleaved.

We are currently working on minor improvements of the usability of ArgoUWE and the migration to the latest version of ArgoUML. We plan to include the UWE well-formedness rules into the design critique mechanism provided by ArgoUML. This model checking mechanism allows the continuous verification of the rules in contrast to the current checking process that is explicitly triggered by the modeler. Further, we will include better support for iterative and incremental modeling. Finally, we will improve ArgoUWE with the new modeling elements incorporated in UWE as well as additional well-formedness rules needed in the design of personalized and business process guided Web applications.

## References

- Luciano Baresi, Franca Garzotto, Monica Maritati (2002). W2000 as a MOF Metamodel. In: Proc. 6<sup>th</sup> World Multiconf. Systemics, Cybernetics & Informatics, Web Engineering Track.
- Hubert Baumeister, Nora Koch, Luis Mandel (1999). Towards a UML Extension for Hypermedia Design. In: Robert France, Bernhard Rumpe (eds.), Proc. 2<sup>nd</sup> Int. Conf. UML (UML'99). Lect. Notes Comp. Sci. 1723. Springer.
- Stefano Ceri, Pietro Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera (2002). Designing Data-Intensive Web Applications. Morgan-Kaufmann.
- Jim Conallen (2003). Building Web Applications with UML. Addison-Wesley. 2<sup>nd</sup> edition.
- Jaime Gómez, Cristina Cachero, Oscar Pastor (2001). On Conceptual Modeling of Device-Independent Web Applications: Towards a Web-Engineering Approach. IEEE Multimedia 8(2), pp. 26–39.
- Ivar Jacobson, Grady Booch, James Rumbaugh (1999). The Unified Software Development Process. Addison-Wesley.
- Nora Koch, Andreas Kraus (2002). The Expressive Power of UML-based Web Engineering. In: Daniel Schwabe, Oscar Pastor, Gustavo Rossi, Luis Olsina (eds.), Proc. 2<sup>nd</sup> Int. Wsh. Web-Oriented Software Technology (IWOOST'02), CYTED.
- Nora Koch, Andreas Kraus (2003). Towards a Common Metamodel for Web Applications. In: Proc. 3<sup>rd</sup> Int. Conf. Web Engineering (ICWE'03). Lect. Notes Comp. Sci. Springer. To appear.
- Daniel Schwabe (2001). A Conference Review System. In: 1<sup>st</sup> Workshop on Web-oriented Software Technology, Valencia, <http://www.dsic.upv.es/~west2001>.

UML (2003). OMG Unified Modeling Language Specification, version 1.5. Object Management Group.  
<http://www.omg.org/cgi-bin/doc?formal/03-03-01>.

Jos Warmer, Anneke Kleppe (1999). The Object Constraint Language. Addison-Wesley.

Gefei Zhang (2002). CASE Support for Modeling Web Applications. Diploma thesis, Ludwig-Maximilians-Universität München.