flashWeb: Graphical Modeling of Web Applications for Data Management

Mihály Jakob **Oliver Schiller** **Holger Schwarz** **Fabian Kaiser**

Institute of Parallel and Distributed Systems
Universität Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
Email: {mihaly.jakob, oliver.schiller, holger.schwarz, fabian.kaiser}@ipvs.uni-stuttgart.de

Abstract

This paper presents flashWeb, a Computer-Aided Web Engineering (CAWE) tool for the model-driven development of web applications that focus on data management. Present-day web applications, like online auction systems or enterprise web portals require comprehensive data access, data processing and data manipulation capabilities. However, existing web application development approaches treat data management operations as second-class citizens. They integrate data operations into existing models or derive them as a by-product of business processes. We argue that data management is an important part of the application logic hence we capture operations with an additional Operation Model. We show that the explicit modeling of operations provides many benefits that distinguish our solution from other approaches. We present the flashWeb development process utilizing a graphical notation for the models in use, a CAWE tool that supports the creation of the graphical models and a code generator that creates ready-to-run web applications.

Keywords: Model-driven web engineering, Graphical web application modeling, CAWE tool, Web application generation, Object-orientation

1 Introduction

The development of data-intensive web applications has been topic to many research activities in the field of web application engineering over the last decade. Traditionally, the primary focus of data-intensive web applications is on the presentation of large amounts of static data in a hyperlinked manner. In addition to that, present-day web applications offer a wide range of interactive features allowing users to modify content that is stored by the web application and to create personalized views or navigation patterns. E-commerce web sites are omnipresent examples for this category of web applications. An online auction system like eBay allows users to create accounts and auction items, place bids or provide feedback on completed transactions thereby creating new content and associations between existing content objects. All users of the system are able to access content objects through searching, diverse indexes and listings that are properly filtered and ordered according to different criteria. As present-day web applications rely heavily on different kinds of data management operations, we need an approach that facilitates the development of web applications providing strong support for reusable business logic components.

In this paper, we present flashWeb, a CAWE tool for the model-driven development of web applications that support data management operations. We show that our solution utilizing the Operation Model and the novel graphical connections between models allows the rapid development of such web applications. Figure 1 depicts an overview of the flashWeb development process.

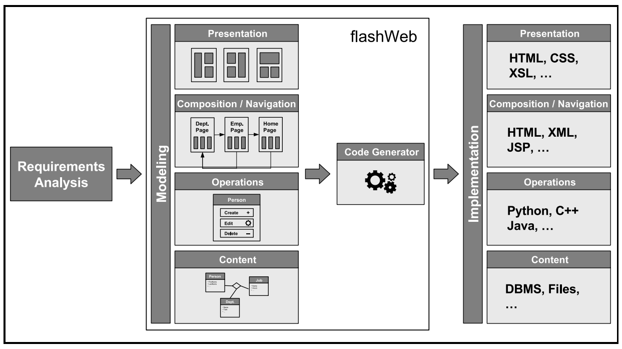

Figure 1: flashWeb Development Process

After the requirements analysis phase, the developer uses four models to define the web application. The Content Model captures content objects and their relationships. The Operation Model defines data management operations that provide sophisticated access to the web application's content. The Composition/Navigation Model describes the composition of the user interface and the navigation structure of the web application. Finally, the Presentation Model defines the positioning and the visual appearance of user interface components. As the next step, the models are used to generate the implementation of the web application. The flashWeb CAWE tool supports the creation of the graphical models, the code generation step as well as the deployment of source code into an appropriate run time environment for the web application.

In Section 2, we compare our approach to existing web application development methods. Upfront, we point out the following differentiating characteristics:

- An Operation Model graphically defines operations that provide versatile access to the web application's content. Operations are defined only once and can be directly used by elements of the Composition/Navigation Model.
- Models are connected in a graphical manner providing an enhanced overview of the application logic indicating which user interface components

access which operations and which operations access which content data.

- All models provide fine granular modeling elements and extension facilities that support the development of web applications with arbitrary functionality.

The rest of the paper is organized as follows: In Section 3 we describe the model-driven web application development approach that is supported by the flashWeb CAWE tool. We briefly introduce the flashWeb models and explain how they are used in conjunction. We also present the idea of graphical connections between models. In Section 4 we describe the flashWeb CAWE tool that supports the graphical specification of web applications based on the introduced models. We also briefly describe a code generation strategy producing ready-to-run web applications. Finally, we conclude the paper and discuss open research issues in Section 5.

2 Related Work

Over the last decade, numerous research approaches have been proposed in the field of model-driven web engineering. Some examples are Hera (Houben, G. J. et al. 2003), OO-H (Gomez, J. et al. 2000), OOHDM (Schwabe, D. et al. 1996), UWE (Koch, N. and Kraus, A. 2002), W2000 (Baresi, L. et al. 2001) and WebML (Ceri, S. et al. 2000, 2002). Most of these approaches were designed to support the development of online information systems that present large amounts of data in a hyperlinked manner. Some of these approaches are also supported by CAWE tools that help to create graphical models and in some cases generate a web application partially. VisualWADE (Gomez, J. 2004) supports OO-H, ArgoUWE (Knapp, A. et al. 2003) builds UML models for UWE and WebRatio (Acerbis, R. et al. 2004) supports the development of WebML web applications.

However, over the last few years web applications have evolved into more complex systems that also provide data management functionality and support business processes. Hence, we need web engineering solutions with enhanced data management functionality. This is only partially supported by existing approaches. Moreover, some approaches jump right to support business processes without offering a solid basis for data management. We compare the flashWeb development process to UWE, to WebML and to OO-H because these approaches have the most in common with our solution.

The UWE approach uses UML throughout the whole development process. A Conceptual Model captures real world entities and their relationships. Navigation nodes of the Navigation Structure Model are derived from classes of the Conceptual Model. Navigation patterns are derived from associations between Conceptual Model classes and extended with indexes. Thus, navigation nodes (web pages) correspond to entities of the Conceptual Model resulting in a coarse-grained content presentation. In contrast to that, the Composition/Navigation Model of flashWeb is independent from the Content Model. Thus, in our case a user interface page can represent several entities of the Content Model by accessing arbitrary operations of the Operation Model.

As a response to arising data management requirements, WebML (Bongio, A. et al. 2000) provides model elements that support simple data manipulation. Similar to WebML, we support the idea of modeling operations with appropriate modeling primitives. However, we pursue a different approach regarding where operations are defined and how they are connected to user interface components. WebML allows the developer to use basic data management operations, like to add, to modify or to delete content objects. It also provides model elements to create or to delete associations between content objects. However, operations are directly defined in the Hypertext Model of WebML, which makes this model complicated. To solve this problem, we strictly separate modeling elements that concern page composition and data access into different models. Our solution also provides composite operations that are defined only once and can be easily *called* by elements of the Composition/Navigation Model.

The OO-H (Gomez, J. et al. 2000) approach is based on OO-Method (Pastor, O. et al. 1997) and uses an Object Model that is similar to our Content Model. It allows the definition of objects through classes that hold attributes and services. The Dynamic Model is additionally used to define object states and object interactions. The Navigation Access Diagram defines the user interface of the web application using Navigation Links that extend classes of the Object Model. In contrast to that our Composition/Navigation Model does not include classes of the Content Model or the Operation Model. It associates user interface elements to operations of the separate Operation Model utilizing graphical connections. Thus, in our case a better separation of concerns is achieved.

3 The flashWeb Modeling Approach

The flashWeb CAWE tool supports the model-driven development of web applications that provide advanced data management functionality. It utilizes graphical models throughout the entire development process. Different aspects of the web application are captured with different models assuring a clear separation of concerns. In a previous publication (Jakob, M. et al. 2006) we introduced the initial idea of an Operation Model that accomplishes the separation of data management functionality from other modeling concerns. Additionally, we showed (Jakob, M. et al. 2006) that the generation of a large web application with a comprehensive Operation Layer for data access is possible. However, that first approach utilizes a preliminary prototype version of our models that are encoded in XML. The graphical models presented in this paper are not just a different representation of those text-based models, but provide besides the obvious advantages many advanced concepts.

The flashWeb development approach is based on three graphical models, the Content Model, the Operation Model and the Composition/Navigation Model. The Presentation Model is a textual model that defines position and visual appearance of user interface components. The following sections provide a brief description of each model and of model interactions.

3.1 Content Model

The Content Model of flashWeb defines the content structure of the web application using a UML compliant notation. Basically, it captures real world entities and their relationships with a UML class diagram. The Content Model supports classes and associations, aggregation and composition as well as specialization/generalization. Class attributes are described by their names, data type and their multiplicity. Associations are specified by their name, role names and multiplicities for the participating classes and optionally by an association class. The bottom half of Figure 3 depicts a very simplistic Content Model example which is an excerpt from the specification of an online auction system.

3.2 Operation Model

The Operation Model defines data management operations that provide full read and write access to the web applications content storage. This model serves as an intermediary between the Composition/Navigation Model and the Content Model. Elements of the Composition/Navigation Model may be associated with arbitrary operations of the Operation Model that retrieve, modify or delete web application content. An important feature of the Operation Model is the reuse of operational constructs. Basic and composite operations are defined only once and can be used by an arbitrary number of Composition/Navigation Model elements.

The syntax of the Operation Model is simple. Each Content Model class has its counterpart in the Operation Model. However, the definition of a separate Operation Model is still advisable for the following reasons. First, the developer may define further Operation Model classes that can be used as containers for composite and custom operations. These operations may affect several Content Model objects so putting them into a single Content Model class is not justifiable. Note that depending on the customization level a web application may have a fair amount of custom and composite operations. Secondly, using different models for content and operations provides a better separation of concerns. An example class that defines operations for a content object is depicted on the left hand side of Figure 2.

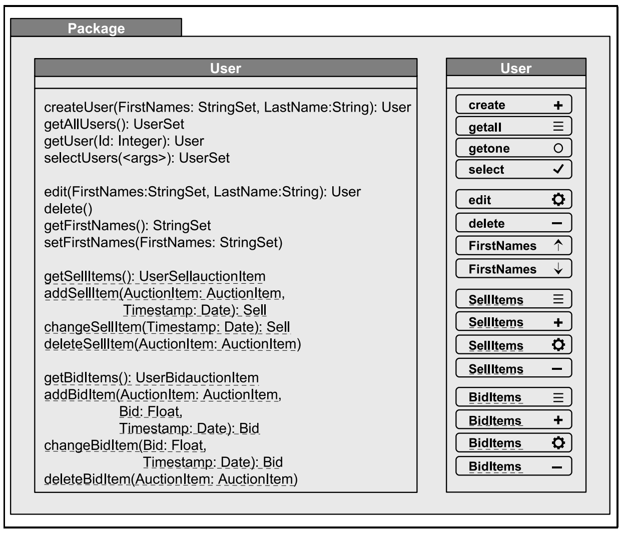
Package
User
createUser(FirstNames: StringSet, LastName:String): User
getAllUsers(): UserSet
getUser(Id: Integer): User
selectUsers(<args>): UserSet
edit(FirstNames:StringSet, LastName:String): User
delete()
getFirstNames(): StringSet
setFirstNames(FirstNames: StringSet)
getSellItems(): UserSellauctionItem
addSellItem(AuctionItem: AuctionItem, Timestamp: Date): Sell
changeSellItem(Timestamp: Date): Sell
deleteSellItem(AuctionItem: AuctionItem)
getBidItems(): UserBidauctionItem
addBidItem(AuctionItem: AuctionItem, Bid: Float, Timestamp: Date): Bid
changeBidItem(Bid: Float, Timestamp: Date): Bid
deleteBidItem(AuctionItem: AuctionItem)
User
create
getall
getone
select
edit
delete
FirstNames
FirstNames
SellItems
SellItems
SellItems
SellItems
BidItems
BidItems
BidItems
BidItems

Figure 2: Basic Operation Examples

This is a UML-near notation and defines class-level operations (underlined), instance-level operations that access one object of the class (not underlined) and instance-level operations that access related objects through associations (dashed underlined). The operation signature is composed of the operation name, the parameter list and the data type of the operations return value. Associations between classes are visualized in the Content Model and are not repeated in the Operation Model. In most cases, a flashWeb developer is not interested in the exact operation signatures. Therefore, the Operation Model also employs a simplified graphical notation that abstracts from details. The right hand side of Figure 2 shows an example of this notation. Each operation is represented with an alias and an icon.

3.3 Operation Types

The Operation Model supports three types of operations: basic operations, composite operations and custom operations.

Basic operations provide simple access to web application content. They allow to retrieve, to create, to modify or to delete content objects. Additionally, they also provide similar access to associations. Figure 2 lists the most important basic operations for the *User* class displaying both the UML-near and the simplified graphical notation.

Composite operations combine two or more operations as one transaction. These operations play an important role in the Operation Model because the application logic of a web application often requires operations to be executed as one transaction. A simple example is the creation of an auction item and the subsequent association of this item with a user in one transaction. Such a composite operation combines two basic operations, one that creates an auction item and another that associates this item to a user. An arbitrary number of basic operations can be flexibly composed into composite operations by so called operation connectors. These connectors define all necessary mappings between results and parameters of participating operations.

Finally, the custom operation type is an extension facility of the Operation Model for non-standard application code. It allows the definition of custom application code in the programming language that implements the Operation Layer of the web application, e. g. Python, Java, etc. This extension mechanism is necessary for web applications that require more than standard data management functionality.

3.4 Connecting Operations to Content

Graphical connections between models are a key characteristic of the flashWeb development process. They express an important part of the web applications application logic and provide enhanced overview thereof. Whereas we use a UML-near notation for the Content Model and the Operation Model, connections between models clearly exceed the possibilities that are provided in UML. Capturing the same semantics, e. g., by using UML stereotypes, would result in a much more complicated graphical representation. Thus, we advocate using a domain specific notation to achieve a simple yet powerful way to develop web applications. Figure 3 shows an example of basic operations and their data access for the *User* and *AuctionItem* classes.

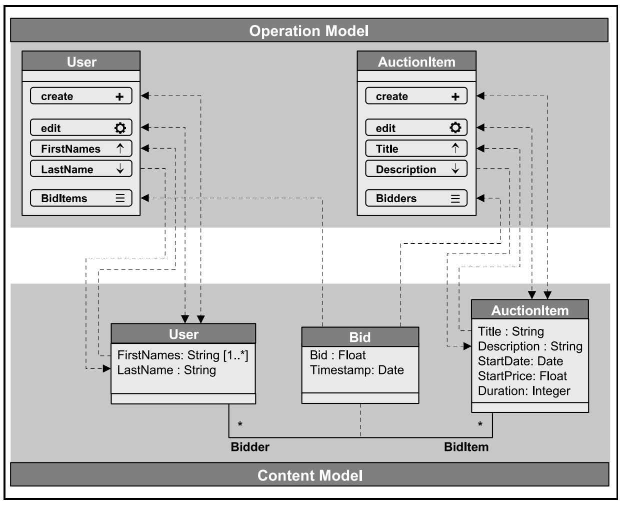

Figure 3: Operations accessing Content

Connections between the Content Model and the Operation Model (dashed arrows) directly visualize for each operation the manner of data access. Data access arrows can be unidirectional or bidirectional indicating whether an operation receives data from the content storage, writes data to the content storage or both. For example the class-level *create+* operation creates a new object and is connected to the corresponding Content Model class. The bidirectional arrow between this operation and the Content Model class indicates that the operation modifies the web applications content and receives data from the content storage, i.e., the newly created class object.

3.5 Composition/Navigation Model

The Composition/Navigation Model of flashWeb defines the web applications user interface. Its modeling elements allow piece-by-piece web page construction and the connection of different elements by navigation edges. Basically, there are four types of model elements. First, structure elements allow to create pages and to divide pages into different areas. Secondly, content elements simply present the web applications content in different types of object views and object listings. Thirdly, data-entry elements allow the definition of user interface components that require the user to provide information that is stored or processed by the web application. Finally, navigation elements connect different elements of the user interface with navigation edges thereby creating the navigation structure of the web application. Note that these categories only define the main purpose of modeling elements. Many elements combine one or more of these aspects in some way. The top half of Figure 4 depicts a very simplistic Composition/Navigation Model example.

3.6 Connecting the User Interface to Operations

Graphical connections between the Composition/Navigation Model and the Operation Model play a central role in the flashWeb development process. All connections of this type are explicitly defined by the web application developer. Connections between the two models (dashed arrows in Figure 4) indicate which user interface components access which operations. Connections may be defined using unidirectional or bidirectional operation access arrows. They indicate whether an operation only receives parameters, only returns a value or both. Connections between the models are labeled in both directions. Labels show which parameters are passed on to an operation and which variables are entered into the namespace of a Composition/Navigation Model element. A special name *result* identifies the result value of an operation. Figure 4 depicts an example showing a simple Composition/Navigation Model and an Operation Model.

The Composition/Navigation Model displays two web pages. The *User Page* includes an Object View element that displays a *User* object and an Entry Form element that defines a search form. The Object View element receives a *User* object from the operation *get* of the *User* class. The *UserId* variable from the namespace of the *User Page* is renamed to *Id* and passed on to this operation. The operation returns a *User* object in the *User* variable. The Object View element uses the value of this variable and displays the *User* object. The *User Page* also provides a search form with two fields. The Name field has the type Input and requires the user to provide some text. The *Category* field of the type Selection receives the list of categories from the *getall* operation

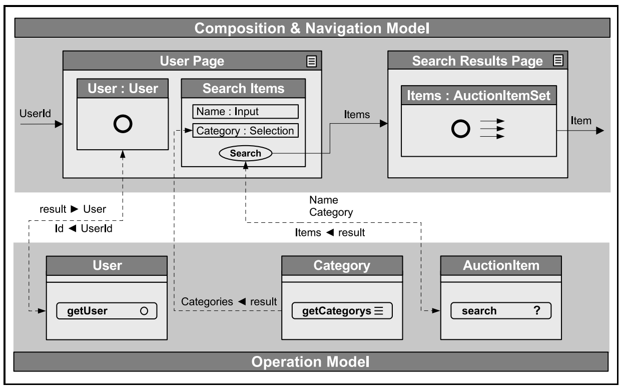

Figure 4: User Interface accessing Operations

of the *Category* class. The search form also provides a search button that calls the search operation in the *AuctionItem* class. Note that the search operation in this example is a custom operation. The operation call (if the user activates the search button) sends the parameters *Name* and *Category* to the search operation and receives a list of found *AuctionItem* objects. Finally, search results are passed on to the *Search Result Page* in the *Items* variable. This page includes an Object Index showing all results.

3.7 Presentation Model

The Presentation Model of flashWeb is not a graphical model. It consists of a set of style definitions that can be assigned to elements of the Composition/Navigation Model in a flexible way. Style definitions are structured into categories like size, color, position, etc. Each element can be assigned an arbitrary number of style definitions resulting in the final rendering of the element.

4 The flashWeb CAWE Tool

The introduced flashWeb modeling approach pursues the rapid development of web applications. To achieve that, a CAWE tool is required that facilitates the fast creation of flashWeb models, the generation of application code and the deployment of code into the desired target platform. In the following sections we present the flashWeb CAWE tool that fulfills these requirements. The demonstration will include the description of the user interface, a step-by-step explanation of the model creation process as well as the description of the code generator and the generated web application.

4.1 User Interface

The user interface of the flashWeb CAWE tool consists of a menu, a tool bar, a fly-out palette, a status bar and a canvas. Figure 5 depicts the mentioned components.

The menu provides common actions like create, open or save a project. Frequently used actions, e. g. redo and undo, are additionally available in the tool bar. In contrast to the menu and the tool bar, actions provided by entries of the fly-out palette are specific to model creation. The entries are categorized corresponding to the three graphical models: the Content Model, the Operation Model and the Composition/Navigation Model. Actions that concern creating elements of a certain model reside in the corresponding category. The last category concerns the creation of connections between models.

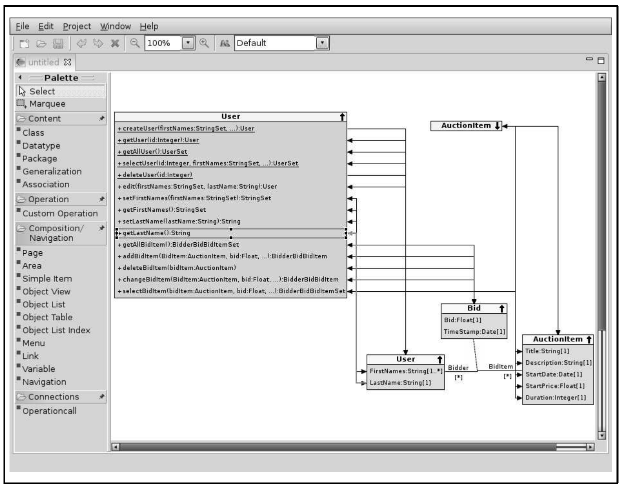
File Edit Project Window Help
100%
Default
untitled
Palette
Select
Marquee
Content
Class
Datatype
Package
Generalization
Association
Operation
Custom Operation
Composition/ Navigation
Page
Area
Simple Item
Object View
Object List
Object Table
Object List Index
Menu
Link
Variable
Navigation
Connections
Operationcall
User
+createUser(firstNames:StringSet, ...):User
+getUser(id:Integer):User
+getAllUser():UserSet
+selectUser(id:Integer, firstNames:StringSet, ...):UserSet
+deleteUser(id:Integer)
+edit(firstNames:StringSet, lastName:String):User
+setFirstNames(firstNames:StringSet):StringSet
+getFirstNames():StringSet
+setLastName(lastName:String):String
+getLastName():String
+getAllBidItem():BidderBidBidItemSet
+addBidItem(BidItem:AuctionItem, bid:Float, ...):BidderBidBidItem
+deleteBidItem(bidItem:AuctionItem)
+changeBidItem(BidItem:AuctionItem, bid:Float, ...):BidderBidBidItem
+selectBidItem(bidItem:AuctionItem, bid:Float, ...):BidderBidBidItemSet
AuctionItem
Bid
Bid:Float[1]
TimeStamp:Date[1]
User
FirstNames:String[1..*]
LastName:String[1]
Bidder
[*]
BidItem
[*]
AuctionItem
Title:String[1]
Description:String[1]
StartDate:Date[1]
StartPrice:Float[1]
Duration:Integer[1]

Figure 5: Content and Operation Modeling

The status bar displays various messages that inform the web application developer about the state of the models, e. g., error messages. An example is a message indicating that the type referenced by a model element cannot be resolved.

Last but not least, the canvas holds the graphical models. It is unlimited in space, but only a certain part is visible. The visible part can be zoomed and moved in all directions. This functionality is essential for large models, so the web application developer is able to obtain a better overview. Note that there exists only one canvas for all models and it is not separated into parts respective to the models. Hence, the web application developer is free to place model elements arbitrarily.

The number of connections beetween models increases quickly. The flashWeb CAWE tool provides several features to handle this circumstance. First, most model elements can be collapsed, so that their subelements and most of their connections become hidden. Figure 5 shows the collapsed *AuctionItem* operation class and other classes that are decollapsed. Secondly, all incoming and outgoing connections of a selected model element are highlighted. Observe the highlighted connection between the *getLastName* operation and the *LastName* attribute in Figure 5. Finally, there exists the possibility to hide all connections of a certain type temporarily, e. g., to hide all connections between the Content Model and the Operation Model.

To enhance the comfort of model editing, the flashWeb CAWE tool provides support for undoing changes. This feature is combined with the possibility to redo undone changes.

4.2 Modeling

In this section we demonstrate by example the functionality of the flashWeb CAWE tool. Figure 5 depicts the small Content Model and Operation Model example from Section 3.4, where the semantics of this example is explained in more detail.

The creation of a model element is a simple task. The following steps describe how the *User* content class is created with the flashWeb CAWE tool. First, the web application developer selects the *Class* element of the *Content* category from the palette. Secondly, he clicks on the canvas at a desireable location. Finally, he fills in the name of the class, e. g., User in the dialog that is presented to him. After that, the class is created. Note that the counterpart in the Operation Model is automatically generated. Moreover, the flashWeb CAWE tool automatically creates basic operations, like *createUser*, and connects them to the Content Model.

Attributes for a content class can be added using the context menu of the corresponding model element. The required information, the name, the type and the mulitplicity of the attribute is queried by a dialog. The creation of a content class attribute results in the generation of according access operations and the adaptation of operations that depend on class attributes. Thus, due to the creation of the attribute *FirstNames* the operations *setFirstNames* and *getFirstNames* are generated and the basic operations *createUser*, *selectUser*, *deleteUser* and *edit* are expanded by the parameter *firstNames.* If the length of the parameter list exceeds a defined threshold, only an abbreviated form is displayed. The full operation signature is obvious and is shown in a tooltip that pops up if moving the mouse pointer over the operation. As described in Section 3.2 there exists a simplified graphical notation for operation signatures that can be used as an alternative.

Creating connections between model elements is also a simple task. Subsequently, we describe how the *Bid* association between the *User* and *AuctionItem* Content Model classes is created. First, the developer selects the *Association* palette entry. Secondly, he selects the *User* class as source and then the *AuctionItem* class as target of the association. Finally, he provides the name, the role names and the multiplicities for the dialog that is opened. Having entered the necessary information, the association is created. Basic access operations in the corresponding Operation Model classes and appropriate connections to the Content Model are generated automatically.

Figure 6 depicts a small Operation Model and Composition/Navigation Model example from Section 3.6. It shows the two pages *User Page* and *Search Results Page* that model pages of the user interface.

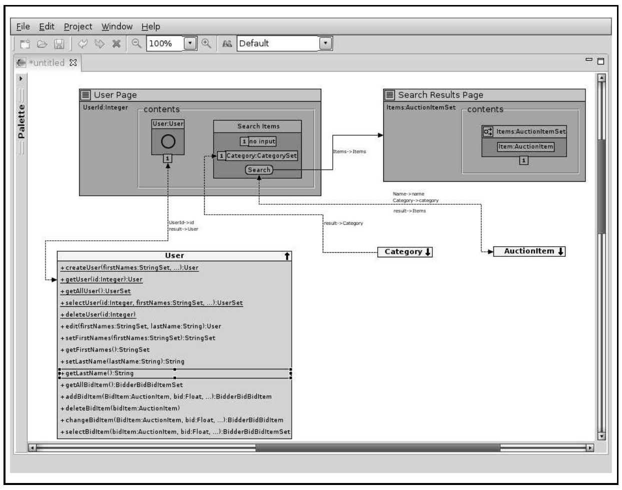
File Edit Project Window Help
100%
Default
*untitled
Palette
User Page
UserId:Integer
contents
User:User
1
Search Items
1 no input
1 Category:CategorySet
Search
Search Results Page
Items:AuctionItemSet
contents
Items:AuctionItemSet
Item:AuctionItem
1
Items->Items
Name->name
Category->category
result->Items
UserId->id
result->User
result->Category
Category
AuctionItem
User
+createUser(firstNames:StringSet, ...):User
+getUser(id:Integer):User
+getAllUser():UserSet
+selectUser(id:Integer, firstNames:StringSet, ...):UserSet
+deleteUser(id:Integer)
+edit(firstNames:StringSet, lastName:String):User
+setFirstNames(firstNames:StringSet):StringSet
+getFirstNames():StringSet
+setLastName(lastName:String):String
+getLastName():String
+getAllBidItem():BidderBidBidItemSet
+addBidItem(BidItem:AuctionItem, bid:Float, ...):BidderBidBidItem
+deleteBidItem(bidItem:AuctionItem)
+changeBidItem(BidItem:AuctionItem, bid:Float, ...):BidderBidBidItem
+selectBidItem(bidItem:AuctionItem, bid:Float, ...):BidderBidBidItemSet

Figure 6: Composition and Navigation Modeling

The body of a page element consists of two parts, one that defines variables and another that contains subelements. The section containing subelements is surrounded by a frame and bears the *contents* label. For example, the *User Page* page in Figure 6 defines the variable *UserId* and contains two subelements. An Object View element that displays a *User* object and Custom Form named *Search Items.*

Model elements of the Composition/Navigation Model that require an input from a variable or from an operation call, possess an input slot. Input slots are visualized as small rectangles attached to the corresponding element, e. g., the Object View element on the *User Page* page. The slots of an element are numbered consecutively to identify input parameters uniquely. The web application developer may connect an input slot to a variable of a surrounding Composition/Navigation Model element or to an operation of the Operation Model. This can be achieved either by using the context menu of the input slot or by selecting the *Operation Call* element from the palette. Note that the input slot of the Object View element displaying a *User* object is connected to the *getUser* operation, which receives its input, i.e., the user id from the *UserId* variable of the *User Page*. Mappings between namespaces are displayed as labels on the corresponding edge. The label UserId→id denotes that the *UserId* variable from the namespace of the *User Page* page is mapped to the *id* parameter of the *getUser* operation. Additionally, the result→User label indicates that the result of the corresponding operation is mapped to the *User* variable in the namespace of the connected Object View element.

A successful generation of the web application demands holding all models in a consistent state. To support this task, the flashWeb CAWE tool provides three useful features. First, all dialogs immediately validate entered data, e. g., the dialog querying the name of a class validates that the entered name is unique. In case of a violation the dialog shows an appropiate error message and prevents the web application developer from proceeding. Secondly, the CAWE tool provides information through pop-ups if the developer tries to select a model element as part of an action, indicating whether the selection is appropriate. Finally, an element having an inconsistent state is accordingly annotated and the status bar shows an appropriate error message.

4.3 Code Generation and Deployment

The flashWeb CAWE tool provides a flexible strategy for integrating code generator components that support different target platforms, e. g., JavaEE, Zope or Ruby on Rails. It bases upon the Rich Client Platform (RCP), which constitutes a small subset of the Eclipse Platform. The RCP focuses on the development of applications with the utilization of the plugin-based architecture of Eclipse, but is independent from the Eclipse IDE. A key benefit of this architecture is the ease of extensibility, which avails building the code generation and deployment part of the tool. Code generator components are Eclipse plugins that register themselves to the flashWeb CAWE tool. Generator plugins provide two modes of operation: on-demand and on-the-fly. The on-demand mode requires the web application developer to trigger the code generation process explicitly. The on-the-fly mode adjusts the generated code each time the web application developer changes the models. Note that generator plugins can be configured to deploy generated code to the designated target platform automatically. Using the on-the-fly mode and the deployment feature of the flashWeb CAWE tool enables the developer to propagate model changes instantly to the implementation of the web application thereby minimizing the edit-generate-test cycle. Currently, the flashWeb CAWE tool provides one fully functional code generator plugin for the Zope 3.3 application server. The demonstration includes the description of code generation features and the presentation of the generated web application.

5 Conclusions and Future Work

This paper introduces flashWeb, a CAWE tool for the model-driven development of web applications with focus on data management functionality. The overall strategy of the CAWE tool is to provide a wide range of standard modeling elements that support most of the web application developers needs. Additional functionality can be inserted into the modeling process using custom elements, e. g., custom operations. The presented CAWE tool has been used to create several small applications that serve as proof of concept for the suitability of our approach. However, we plan to thoroughly evaluate flashWeb in different application areas creating medium and large projects.

References

Acerbis, R. et.al. (2004), WebRatio: an Innovative Technology for Web Application Development, *in* Proc. of ICWE2004, Munich, Germany.

Baresi, L., Garzotto, F. & Paolini, P. (2001), Extending UML for Modeling Web Applications, *in* Proc. of HICCS34, Maui, Hawaii, USA.

Bongio, A., Ceri, S., Fraternali, P. & Maurino, A. (2000), Modeling Data Entry and Operations in WebML, *in* Proc. of WebDB2000, Dallas, USA.

Ceri, S., Fraternali, P., & Bongio, A. (2000), Web Modeling Language (WebML): a Modeling Language for Designing Web Sites, *in* Computer Networks Vol. 33, p. 137–157.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S. & Matera, M. (2002), Designing Data-intensive Web Applications, Morgan-Kaufmann.

Conallen, J. (2003), Building Web Applications with UML, Addison-Wesley.

Gomez, J. (2004), Model-Driven Web Development with VisualWADE, *in* Proc. of Source.

Gomez, J., Cachero, C. & Pastor, O. (2000), Extending a Conceptual Modelling Approach to Web Application Design, *in* Proc. of CaiSE2000, Stockholm, Sweden.

Houben, G. J., Barna, P., Frasincar, F., & Vdovjak, R. (2003), Hera: Development of Semantic Web Information Systems, *in* Proc. of ICWE2003, Oviedo, Spain.

Jakob, M., Schwarz, H., Kaiser, F. & Mitschang, B. (2006), Modeling and generating application logic for data-intensive web applications, *in* Proc. of ICWE2006, Palo Alto, USA.

Jakob, M., Schwarz, H., Kaiser, F. & Mitschang, B. (2006), Towards an Operation Model for Generated Web Applications., *in* Proc. of MDWE2006, Palo Alto, USA.

Knapp, A., Koch, N., Moser, F. & Zhang, G. (2003), ArgoUWE: A Case Tool for Web Applications, *in* Proc. of EMSISE2003, Geneva, Switzerland.

Koch, N. & Kraus, A. (2002), The Expressive Power of UML-based Web Engineering, *in* Proc. of IWOOST02, Cyted.

Lima, F. & Schwabe, D. (2003), Modeling Applications for the Semantic Web, *in* Proc. of ICWE2003, Oviedo, Spain.

Pastor, O. et al. (1997), OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods, *in* Proc. of CaiSE1997, Barcelona, Spain.

Schwabe, D., Rossi, G. & Barbosa, S. (1996), Systematic Hypermedia Application Design with OOHDM, *in* Proc. of Hypertext96, Washington, USA.