Specification of web applications design in CASE using UML and its mapping to an implementation environment

Peter Blšták*
peter.blstak@softec.sk

Mária Bieliková**
bielik@fiit.stuba.sk

Abstract: Software systems design requires constantly prompter and more systematic approach. Therefore, new ways of software systems design specification are being searched for and support tools are being developed. In this paper we discuss an issue of effective web applications design using UML. Further, we elaborate an issue of transition from the applications design to its implementation via code generation. We describe the design specification, which is based on domain object-orientated design realized using class diagrams by which the application and database tier are modeled. A part of design specification is a dynamic model of the web application presentation layer, which uses state diagrams. For simplifying the application code generation and for increasing the ratio of generated code we have developed target framework called TF framework. Based on the presented design specification, we have designed, implemented and evaluated the application code generator for the TF framework using the CASE tool Poseidon for UML.

Key Words: web application, design specification, modeling, UML, code generation.

1 Introduction

Faster communication links, grooving Internet and intranet cause an increasing client server applications use. These applications provide a rich functionality to a user through networks. Nowadays these applications are often web-based, i.e. they communicate with users by the web interface using web browsers (thin clients). In the context of the web applications increasing popularity and their use together with their grooving complexity, there has been developed a lot of technologies to support faster and simpler development [1, 5, 7, 8]. At present the market requires more systematic and faster web applications development, putting accent on precise technical application documentation, easy application extensibility and adaptation to new application requirements. The answer to requirements of more systematic and faster development can be an improvement of specification of the web application design, implementation or the usage of a framework for building applications and implementation of support tools automating transition from application design to its implementation.

As the main issue here appears the specification of a web application design. That is which application key parts should be expressed by the design and how they should be represented by the design model. Resolving this crucial aspect of application development is a good groundwork for building application framework and automation development.

* Softec ltd., Kutuzovova 23, 831 03 Bratislava, Slovakia

** Institute of Informatics and Software engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovicova 3, 812 16 Bratislava, Slovakia

The developing process of web applications differs from the developing process of traditional software. Therefore, it is necessary to adapt existing methods and techniques for requirements of web applications development. In past five-eight years a few methodologies supporting web applications design have sprung up. They have been mostly based on proprietary methods and techniques, which were not supported by widely accepted CASE tools and prominent companies in this area. This has caused no expand and practically no use of systematic approach to design of web applications. Nowadays it is evident that for realizing analysis and design of software systems it is right and appropriate to use the Unified Modeling Language (UML) [3] with appropriate extensions [4, 5, 7].

In this work we concentrate on specification of web application physical (detailed) design. We have proposed a design approach, at which we put emphasis on the ability of increasing development process efficiency by automation of a transition from the application design to its implementation using the application code generation. The design is based on adapted UML class and state diagrams, which enable us to model almost all key aspects of web applications. We cover the presentation, application and database layers. As mentioned above, we concern ourselves with physical design. So we assume that requirements analysis and conceptual application design has been realized by a standardized way, for example by using the UML techniques like use cases and class diagrams.

To succeed in the application code generation, it is necessary to restrict a design specification, generator and possibly target implementation environment to support development process of certain type of applications. In this work we have decided to support development of specialized type of applications – small web-based information systems. This type of applications has usually simple functionality and is focused on store and manipulation a big amount of data with relatively static structure. At the same time it has widespread sphere of use. Therefore it represents suitable type of applications for modeling a code generation.

This paper is structured as follows: Section 2 describes our proposal for web applications design specification. In Section 2.1 we concentrate on the application and database tiers modeling by use of UML class diagrams. Section 2.2 is devoted to description of the presentation tier modeling using UML state diagrams, which represent navigational aspects of application. In Section 3 we evaluate proposed web application design specification by prototyping code generator and realization of small library system. Finally, Section 4 presents conclusions in the context of related works and an overview of our future research.

2 Specification of Web Application Design

Our proposal for web application design specification is based on capabilities of UML considering requirements of a web application design. Our work follows two main objectives. On the one hand, we intend to accomplish simplicity and good arrangement of web applications design and on the other hand, we want to be able to generate as much as possible of the web application code from its specified design. Since these two objectives are in antinomy, we have had to arrive at compromise between amount of information in the web application design and amount of application code, which we are able to generate from the specified design. Thus, it was about finding a state in which the addition of web application features to its design would cause no more simplification of its development (using the code generator).

The physical web application design follows domain object-oriented design of the application tier. We have also decided to provide a modeling of the web application presentation tier (especially navigation aspects).

Design of the web application in our proposal consists of two parts:

- design of the application and database tiers expressed by UML class diagram and
- design of presentation tier expressed by adapted UML state diagram.

2.1 Application and Database Tier Design

As mentioned above, the application tier is modeled by the UML class diagram, which represents application domain and follows the conceptual application domain model [7, 8] (likely expressed by a class diagram too). Each class represents an application domain entity maintained by the application. An example of such entities can be a book, a borrowing and a reader of a library system. These entities are represented in the model by classes `Book`, `Borrowing` and `Reader` with their attributes, their methods and associations between them (shown in *Fig. 1*).

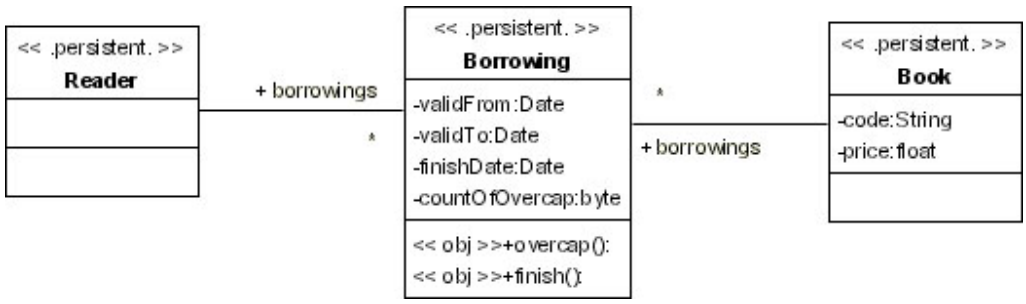

Fig. 1 – Book borrowing model

We have introduced two types of application classes represented by a stereotype. The first type of class is the application persistent domain class (stereotype `persistent`). This type of class represents classical domain entities. Each instance of this type of class has automatically ability to store itself in a database store (RDBMS). Persistence is applied to all class attributes and its associations, which are not marked as derived. The persistence can be realized by hand or by object relational (OR) mapping using some OR mapping tool or library. The second kind of persistence realization is recommended. Attributes and associations can be either derived or persistent. Derived attributes and associations are necessary to be calculated by fly using actual object state and it is not possible to change directly their values.

Associations represent relations between domain entities. To enable the manipulation of associated objects (e.g., get, set, add, remove) it is necessary to mark appropriate association end in the class diagram as navigable. The association manipulating methods may use the name specified by the role name of appropriate association end or may be gained from the name of associated class and association cardinality. For navigable ends of associations with cardinality of one it is possible to access its values. Similarly, for navigable ends of associations with cardinality of many it is possible to automatically realize accessing values, adding new objects and removing objects of association. We suggest not to use associations with cardinalities 0..1 : 0..1 and 1 : 1. In those cases there arise the problems with foreign key position determination in RDBMS or with the consistence keeping, if there are used foreign keys for both entities.

Methods of class represent functionality of their instances or the functionality of the class itself. In addition to standard methods we establish in our design special kind of methods – called application methods. The application methods are designed to be accessible from the presentation tier of designed application. They represent essential part of the application

functionality. Application methods are of three types, which are in the class diagram represented by their stereotypes (`obj, glob, coll`):

- *object application method* – executes functionality on the actual instance of the specified class (e.g., the title reservation in library system, which is realized by invoking a method on the actual title object),
- *global application method* – executes functionality independent of actual concrete instance of a class and is only related to the type of the instances – class (e.g., batch processing of some domain entity),
- *collection application method* – executes functionality over the selected collection of objects of given class (e.g., execution of mass printing report).

To support presentation of domain entities in the presentation tier of web application we extend standard properties of classes, its attributes, associations and methods in the following way. For every attribute, association and application method is able to specify its *caption* and *description*. Caption provides us with a short presentable name of the attribute, association or application method. Description is used to provide a presentable explanation text of the attribute, association or application method. Both properties are intended for communication with a user of the application. The properties may be used in the implementation of the application as labels and tooltips of text fields, tables or buttons and hyperlinks.

In addition, for every application method we can specify *confirmation* and a collection of *application roles*. The confirmation is a presentable text, which is used to ask the user, if he really wants to execute chosen application method after his corresponding action in the presentation tier of application. If there is no confirmation text specified, method is executed with no prior question. Application roles specify which groups of users can execute appropriate application method. This may be used in implementation by disabling or hiding buttons and hyperlinks, which corresponds to specified application method. All of these properties extensions are provided using UML tagged values.

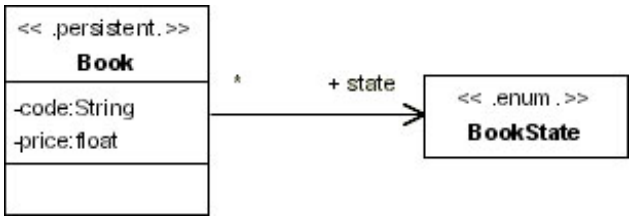

Fig. 2 – Enumerated values of book state

The second type of the class in our design specification is a specialized entity, which allows us to describe enumerations. *Enumeration* is an extensible collection of enumerated values, which is related to some domain entity as its attribute. Enumerations are represented in the model as associations with cardinality one to the class with stereotype `enum`. An example of enumeration is shown in *Fig. 2*. The name of the enumeration class represents name of the particular enumeration. For enumeration it is possible to specify its values by entering a collection of ordered pairs (`code`, `value`), which are realized using UML tagged values.

Part of the design specification presented in this section provides the way of modelling key aspects of application and database tier of web application. This includes domain classes, definition of RDBMS application schema structure and also the specification for OR mapping. Besides, it provides the foundation for building the presentation tier of the web application (essential structure and texts of forms accessing domain objects and a part of presentation tier navigation, which uses domain object associations).

2.2 Presentation Tier Design

Basic graphical and navigational parts of the presentation tier of web applications are forms or pages. Hence, we decided to support design of the presentation tier at this level of abstraction.

Forms in web-based information systems are mostly directed towards providing a view of some application domain entities. In the most cases it is the detailed view on one domain entity (e.g., detail information about one book title) or the view on a collection of domain objects of given type with an ability of sorting and filtering (e.g., collection of book titles, which match entered search criteria). In addition to presentation of domain entities values, forms give us a way to start the execution of some processes on presented domain entities. This is done by using hyperlinks and form buttons.

To express all of these features in our design specification of the presentation tier, we have proposed to use UML state diagram [2]. The state diagram provides a way of specifying individual pages of the presentation tier, the navigation trough these pages, specification of the navigational menu, and the binding of pages to the application middle tier. Forms are represented in the model by states and transitions between forms, which are bound to some kind of execution, are represented by state transitions (shown in *Fig. 3*). Actual state (form) may be changed by firing an action or execution process by the user, which may cause the transition to another state of the presentation tier, i.e. the other form is displayed to the user.

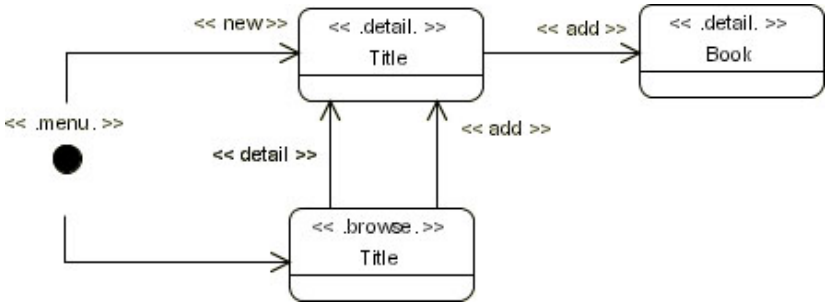

Fig. 3 – Forms and navigation between them

In our design, forms may be one of three types – detail form, browse form or simple form. The form type is specified by the stereotype of appropriate state in the model (`detail`, `browse` or `simple`). For all form types we can specify a form title and collections of read-only and read-write application roles by entering corresponding tagged values.

The `detail` and `browse` forms are specialized to provide a view on domain entities of certain chosen type, i.e. one of the classes specified in the design of the application middle tier (described in section *2.1*). Association of a domain class to the `detail` or `browse` form is specified by the state name (which has to match the domain class name) or by a special property of the state, which specifies the class name (expressed by tagged value `domainClass`). The `detail` form is a view on one instance of domain entity and its possible associated objects. It allows making changes to the domain object, storing it to a persistent store, sorting of associated objects automatically. It also allows executing object and global application methods on actual object, and the execution of object and collection application methods on its associated objects, which includes addition and removal of associated objects. The `browse` form is a view on collection of instances of a domain entity. It is automatically possible to filter and sort the object collection, and to execute all object and collection application methods on objects of that collection. Third type of forms does not provide a view on the domain entity and the content of such forms is not specified in the UML model. For these forms it is possible to specify only a form transition, which

corresponds to the form methods (it is not possible to execute application methods – no domain instance is available).

Specialized state in the model is the starting state with stereotype `menu`. This state represents the application navigation menu and transitions from this state represent items of this menu.

As mentioned above, state transitions correspond to methods, which can be fired from the presentation tier by user. In the HTML presentation they are represented by hyperlinks and form buttons. Similarly as for application methods, for all transitions can be specified caption and description texts and application roles, which can execute processing bound to appropriate transition. Representation of these properties is realized by tagged values.

In addition, we can also specify for transitions the form open type, which is related to opening form (state – form at the end of transition). The open type has relation to stack of opened forms, which allows us to manage and organize opened forms. Form stack allows returning to previous opened forms (realization of back transition to not predefined unknown form) and may provide the user with context navigation functionality, which may reduce the risk of "lost in hyperspace". Open type can be one of these options:

- *add* – addition of form at the end of form stack (standard form open)
- *replace* – addition of form at the end of form stack with previous removing of last form in it (replacing of last form in stack form)
- *clear* - addition of form to the form stack, which removes all previously opened forms from it (typically used by transitions from navigation menu)

Transitions may be of standard or special type. Standard transitions correspond to application specific domain logic. Specialized transitions correspond to common often used application functionality, which is not necessary to specify in the application code and it is possible to fully use them across multiple applications and applications' designs.

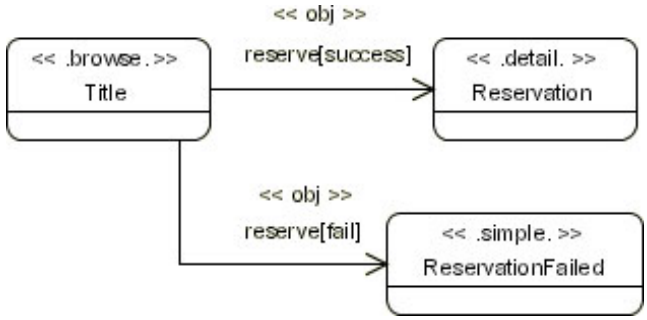

Fig. 4 – Transitions related to application method

In case of standard transitions the transition is bound to an application method of the domain object or method of the source form. Stereotype of the transition defines the type of method (`obj`, `coll`, `glob` and `form`), transition event specifies name of method and transition guard represents logical result, which condition the transition to target form. By specifying more than one transition for the same method and the same form, we may achieve conditional flow of forms dependent on result of executing method (shown in *Fig. 4*).

To specialized type of transitions belong `new`, `detail`, `add` and `select` transitions, which are in model marked by corresponding stereotype. Transition `new` represents transition to the detail form, which presents a new object with initialized values. It is possible to modify these values and save new object in the persistent store. The `detail` transition represents a transition to the detail view of the object, which is associated by currently viewed object. It is mostly used to show details of the object, which is actually shown in the browse view. The

`add` transition represents opening of the detail form for entering new object values. After submission of this form the corresponding new object is added to specified association of the object shown in source form. The `select` transition allows specifying zoom functionality, i.e. opening of the browse form in selection mode and after submission setting selected objects to specified association of the object shown in source form. In the case of transitions of type `detail`, `add` and `select`, it is possible to specify name of association, to which the corresponding action is related to. Name of association is specified by the name of transition. If the name of the transition is not provided, association to which is the action related to is determined by type of domain entity shown in target form.

For application methods, which do not cause transition to other form it is not necessary to specify the state transition in model. All such application methods are automatically accessible from the forms, which define a view on corresponding domain object.

3 Evaluation

For the need of evaluation of our approach to design specification, we have created prototype framework TF (Tine Framework) and prototype of the application code generator. TF is build on J2EE standards and is realized by integration and extensions of Struts opensource presentation framework and opensource OR mapping product OJB (ObJect Relational Bridge). The TF code generator is realized for CASE tools supporting XMI standard (in our case Poseidon for UML) and is based on opensource generator engine AndroMDA.

We have evaluated proposed approach by designing the library system and using created code generator for generating the core of fully functional library application. Generated library system contains six persistent domain entities, two application specific enumerations and nine partly reusable forms. The structure and almost all of the key parts of the application was generated from its design model. To finalize the library system it was necessary to implement (five) application methods and slightly modify generated forms to adapt our presentation needs. In other words it was generated more than 95 % (lines) of application source code.

The proportion of generated code depends on the application specific functionality (other than browse, filter, insert, update and delete), which has to be implemented manually, and on the complexity of graphical design of created application. In our evaluation application we have reached excellent proportion of generated code mainly because of the simplicity of the library system and no special graphical design requirements. In real applications we expect to achieve about 60 – 80 % automation.

The evaluation proves that our approach is usable and provides good support of the systematic way for rapid application development. To reuse this approach and generator in real project it would be necessary to adapt some of the generator templates and common files (especially the templates for JSP generation and CSS styles, which affects application graphical design).

4 Related Works and Conclusions

Since the middle of nineties there have been proposed a lot of methods and techniques, which support the web application design and development process. An overview is presented in [6] where the most relevant methods, such as OOHDM [8], WSDM [9], and corresponding techniques are described. In the most cases these methods use proprietary or own techniques [1, 9] or have no support for automation. So they are not appropriate for common use. Some of design methods and methodologies are based on UML “standard” and also provide automation support, but they are often too complex [1, 5].

In this paper we have presented an approach focused on the physical design of web-based data intensive applications. It concentrates only on particular kind of web-based applications (in contrast of [5, 8, 9]). Our approach is based on UML class and state diagrams, which are extended using standard UML mechanisms to fulfill requirements of web application design. Class diagrams are used to model the application and database tiers of the web application and state diagrams are used to model the presentation tier of the web application with accent on the navigational aspect of presentation.

The strength of our approach is given by the fact that it covers all the key features of chosen kind of web applications, keeps easy, produces minimal overhead to development process and allows to fully automating transition from application design to its implementation by application code generation. All of these properties of our approach were attested by prototype code generator, which is independent of CASE tool, prototype framework and realization of the library system.

The objectives of our future work are to extend modeling of transient and session objects of application tier, which will have impact also on presentation tier design and to redesign JSP code generation to two steps generation using XSLT in step two allowing independence of graphical design from main generation.

This work has been partially supported by the Grant Agency of SR grant No. VG1/ 0162/03, by Science and Technology Assistance Agency under the contract No. APVT-20-007104 and by Softec ltd.

References

1. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M.,Comai, S., Matera, M.: *Design Data-Intensive Web Applications*, Morgan Kaufmnn Publishers, ISBN: 1-55860-843-5, 2003.
2. Dolog, P., Bieliková, M.: *Transforming State Diagrams into Navigation Objects in Hypermedia Applications*, KIVT FEI Slovak University of Technology, 2002.
3. Fowler, M., Scott, K.: *UML distiled – Applying the Standard Object Modeling Language*, Addison Wesley, 5th Printing, Dec. 1997, ISBN 0-201-32563-2.
4. Hennicker, R., Koch, N.: *A UML-based methodology for hypermedia design*, Proc. of UML 2000 Conference, York, England, Springer LNCS 1939, Oct. 2000. http://www.pst.informatik.uni-muenchen.de/personen/kochn/Uml2000.pdf
5. Knapp, A., Koch, N., Moser, F., Zhang, G.: *ArgoUWE: A CASE Tool for Web Applications*, In First International Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE03) held in conjunction with OOIS03, Sep. 2003. http://www.pst.informatik.uni-muenchen.de/personen/kochn/emsise-argouwe-knapp-others.pdf
6. Koch, N.: *A Comparative Study of Methods for Hypermedia Development*, Technical Report 9905, Ludwig-Maximilians-Universität München, Nov. 1999. http://www.pst.informatik.uni-muenchen.de/personen/kochn/techrep/hypdev.pdf
7. Pribula, E., Slyško P., Šolc, R.: *Design of framework JavaTEC – 1.phase*, Softec s.r.o., Sep. 2001.
8. Schwabe, D., Rossi, G., Barbosa, S. D. J.: *Systematic hypermedia application design OOHDM*, In Proc. of the ACM International Conference on Hypertext, Washington, USA, ACM Press, Mar. 1996.
9. Troyer, O. D., Leune, C.: *WSDM : A User-Centered Design Method for Web Sites*, Computer Networks and ISDN systems, Proceedings of the 7th International World Wide Web Conference, Elsevier, 85-94, 1998.