Object-Oriented Modeling of Database-Centric Web Applications

Dragan Milicev
University of Belgrade
School of Electrical Engineering, Department of Computer Science and Engineering
E-mail: emiliced@etf.bg.ac.yu

Abstract: *The contemporary three-tier Internet architectures provide a convenient infrastructure for database-centric distributed applications with universal, cross-platform accessibility. However, the software development process for such applications seems to be insufficiently mature and raised on the appropriate level of abstraction to follow efficiently the highly evolved infrastructure and to enable an adequately productive application construction. This paper analyzes the problem in order to discover a common database-centric Web application model that may be formalized. A modeling technique is proposed, which follows well-defined and widely accepted object-oriented modeling principles and the Unified Modeling Language notation. Thus, the approach enables modeling at a high level of abstraction and supports a formalized transition from the object-oriented specification to the running Web application that may be automated. A light prototypal runtime environment and a simple tool that may generate the application from the UML specifications have been implemented. The proposed approach is easy to comprehend, apply, and implement. It is least restrictive, in the sense that it is open for arbitrary extensions that may be implemented using other programming paradigms.*

Keywords: Internet, World Wide Web, Three-Tier Architecture, Object-Oriented Modeling, The Unified Modeling Language (UML), Rapid Application Development, Database, Conceptual Modeling

Introduction

The contemporary three-tier Internet architectures provide a convenient infrastructure for database-oriented distributed applications with universal, cross-platform accessibility [6, 7, 8]. The crisp separation of the user interface, application, and database tiers, which are loosely coupled through standardized interfaces, is one of the most important rationales for their applicability. However, the software development process for such applications seems to be insufficiently mature and raised on the appropriate level of abstraction to follow efficiently the highly evolved infrastructure and to enable an adequately productive application construction. One of the reasons is that the Web implementation model does not relate well to state-of-the-art software development models [4]. Much effort has been made to find a solution to this problem. There are constant attempts to incorporate the object-oriented model into the Web infrastructure generally [6, 4], and with special emphasis on the database-centric applications [2] (see the sidebar). This paper proposes yet another approach, which follows well-defined and widely accepted object-oriented modeling principles and the Unified Modeling Language notation. This is why it does not demand adoption of special-purpose conceptual models and development principles. Besides, it is easy to comprehend, apply, and implement. Finally, it is highly flexible, in the sense that it is open for arbitrary extensions that may be implemented using other programming paradigms, as well as for other conceptual models.

Problem Analysis

The problem is analyzed first, in a way that may discover a common database-oriented Web application model and a pattern that may be formalized. In this case, a modeling technique, a tool for rapid application development, and a runtime environment may support the approach.

Observed from the execution and infra-structural perspectives, a typical database-centric Web application model is shown in Figure 1a. On the client machine, Web pages are presented to the user by a Web browser. The page may be a form with controls (textboxes, listboxes, checkboxes, etc.) that allow retrieval and modification of the data from the database. The user may change the data and activate the "Submit" operation. This action causes the standard *post* method to be invoked, which consists of sending a series of items from the client to the application server. Each item corresponds to one form's control, and carries the name and the value of the control. The application server is responsible for applying business rules, retrieving and storing data to the database, and finding the next page that will be sent to the client as an HTML page. In order to communicate with the database server, the application server performs SQL queries upon the database.

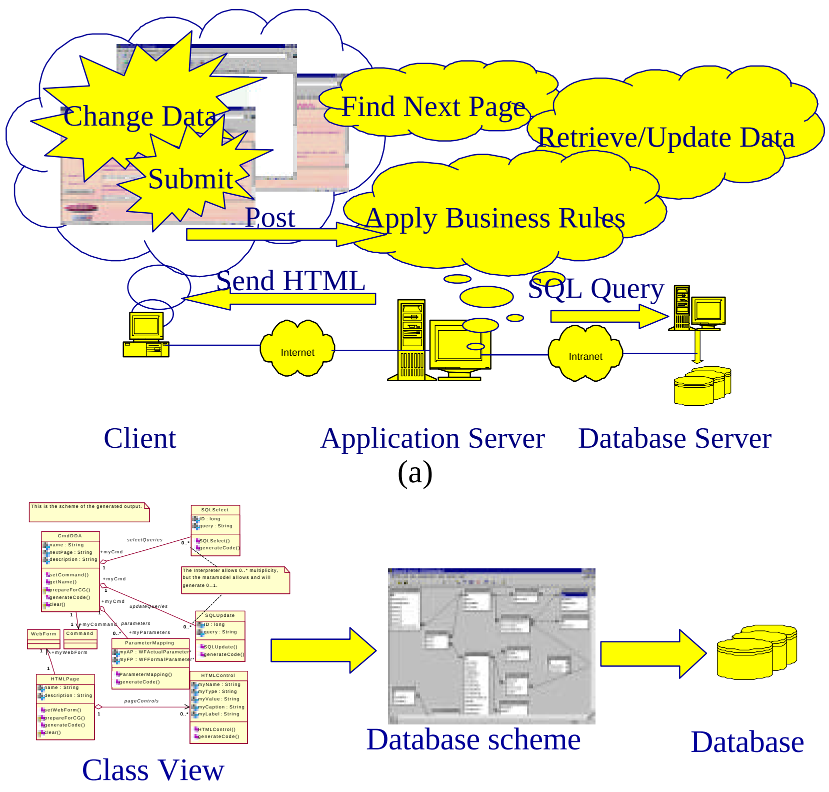


Figure 1: A typical database-centric Web application model. (a) The infra-structural and execution perspective. (b) The conceptual perspective.

Observed from the conceptual perspective (Figure 1b), and following the widely accepted object-oriented principles, a typical database-centric application is an implementation of an object-oriented model [1], where the database scheme is only a persistent implementation of the domain's structural (class) model. Namely, the structural model of the domain for which such application is developed may be specified (in the analysis phase) by the class model with abstractions (classes) from the domain and relationships between them (most notably, associations and generalizations). On the other side, the behavioral aspect of the application is specified by use-cases [1, 5]. However, it may be noticed that a majority of use-cases that have to be implemented in a typical application may be characterized as "structural." These are use-cases that simply create, modify (attribute values of), or delete instances of the domain abstractions and their links (as instances of associations). Often very few use-cases may be qualified as "behavioral," meaning that they navigate through the application structure (instances and links) and provide the functionality that is specific for the application. Besides, structural use-cases may be designed using various patterns that are based on the semantics of the relationships between abstractions. Consequently, these use-cases may be implemented fully or semi-automatically. Consequently, a modeling technique should support modeling at a high level of abstraction, allow fast development of typical structural scenarios, and be open for implementation of non-typical behavioral scenarios.

The Idea of the Proposed Approach

The idea of the proposed solution is based on the observation of a typical man-machine interaction cycle, shown in Figure 2. While executing the application, the user modifies data in a certain starting Web form. This operation does not encompass any interaction with the servers, because it may be completely performed by the Web browser. The cycle starts when the user chooses to submit the form. This operation may be generalized and it may be assumed that the user chooses one of the *commands* that are offered in the form. The data from the form are sent to the server by the post method. The application server should first update the database by the data obtained from the form. This operation will be referred to as performing SQL Update queries, although the queries may be any that change the database (Insert, Delete, or none). Then, the server should find the next Web page for the issued command. Since this page may be another form whose controls should be filled in by the data from the database, the server performs SQL Select queries to retrieve the data. Upon reception of the query results, the server reformats the next HTML page by filling in the concrete values of the controls. Finally, the server sends the ending HTML form to the client, and the cycle is repeated.

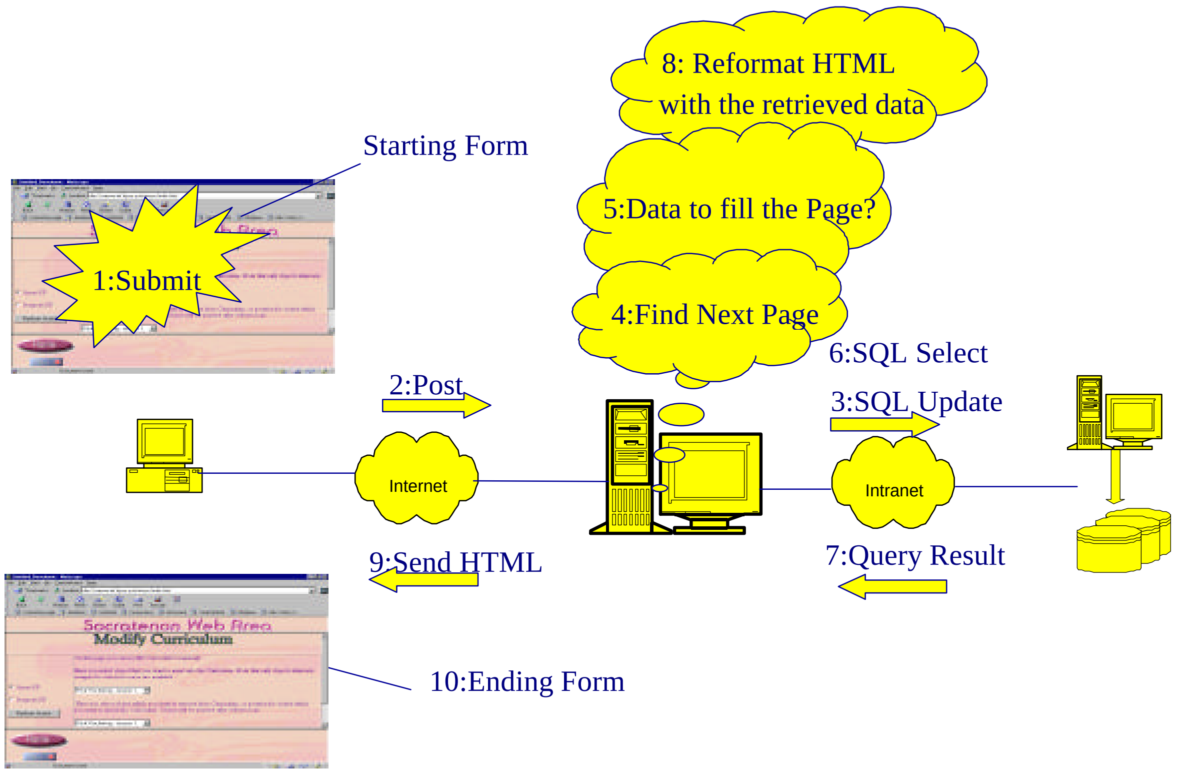
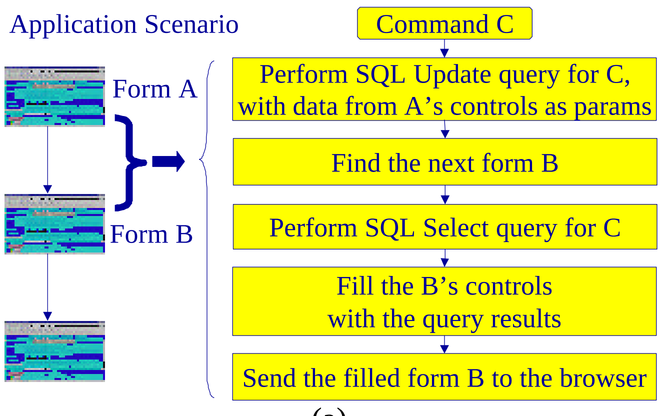
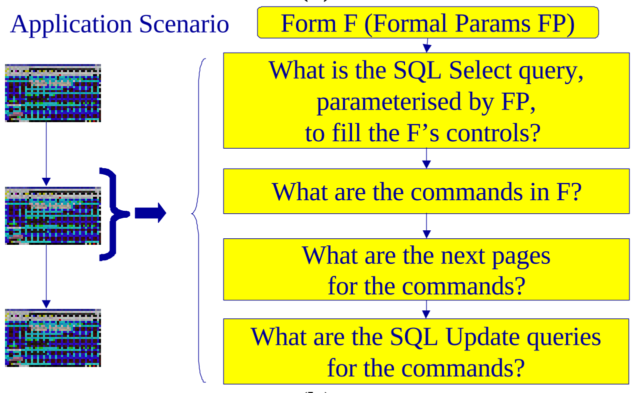


Figure 2: A typical man-machine interaction cycle during execution of a database-centric Web application.


Application Scenario
Command C
Form A
Form B
Perform SQL Update query for C, with data from A's controls as params
Find the next form B
Perform SQL Select query for C
Fill the B's controls with the query results
Send the filled form B to the browser

(a)


Application Scenario
Form F (Formal Params FP)
What is the SQL Select query, parameterised by FP, to fill the F's controls?
What are the commands in F?
What are the next pages for the commands?
What are the SQL Update queries for the commands?

(b)

Figure 3: The conceptual view of the proposed environment. (a) What is important to the computer: the transition cycle. (b) What is important to the user and developer: the application forms, their parameters, queries, and commands.

Consequently, the conceptual model of the presented cycle that is important for the application execution may be defined (Figure 3a). When a command is issued, a set of SQL Update queries must be applied. Each query may be parameterized with the data obtained from the starting form. For example, if the user defines the ID of a database record in the starting form, the query may be parameterized as "UPDATE ... WHERE ID = $ID," where $ID is the reference to the control named ID in the starting form. The server should replace the parameters with the concrete values obtained from the form, prior to applying the query. Then, the server finds the target Web form (HTML page) associated with the issued command. This form may contain controls, with the values also defined as references to the results of the SQL Select queries associated with the command. Such an HTML page is usually called a template page. Thus, the server applies the Select queries (which may also be parameterized with the starting form's controls), and replaces the references (or placeholders) with the concrete results of the queries. Finally, the filled ending page is sent to the client. This way, the application server may execute an interpreter that interprets the application, which is defined in terms of Web forms (as templates), commands, and parameterized queries associated with the commands.

However, the described perspective is execution-oriented and is at a too low level of abstraction. For the user and developer, it is much easier to use more abstract terms, appropriate for the application specification task. For this purpose, another conceptual model is introduced, shown in Figure 3b, which may be transformed uniquely and automatically into the previous one. The application is specified in terms of Web forms, where each form may have *formal* *parameters*. A set of Select queries is assigned to the form, possibly parameterized with the form's formal parameters. Besides, the form may contain controls, the values of which may also refer to the form's formal parameters or Select query results. The formal parameters and the values of the controls represent the set of "local variables" (or local scope) of the form, just as formal parameters and local variables do in procedural programming languages. A set of Update queries is also assigned to the form, parameterized with the form's local variables. Finally, a set of commands is associated with the form. A set of Update queries may also be associated separately with each command, as well as the next Web form. A command passes actual parameters to the next form. The actual parameters may have the values of the invoking form's local variables. Consequently, this model is similar to the commonly accepted and easily understood procedural model, where forms are analogous to procedures.

The Modeling Technique

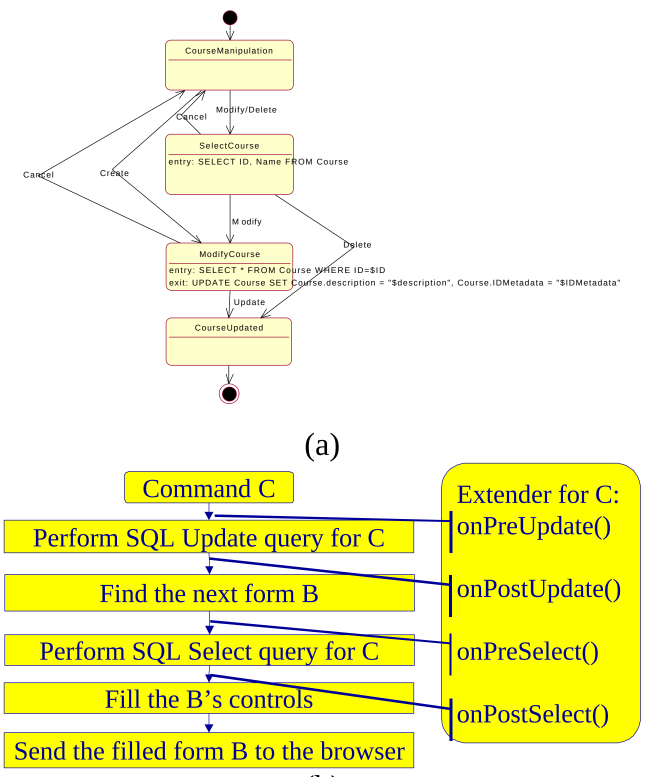

CourseManipulation
Modify/Delete
Cancel
SelectCourse
entry: SELECT ID, Name FROM Course
Cancel
Create
M odify
Delete
ModifyCourse
entry: SELECT * FROM Course WHERE ID=$ID
exit: UPDATE Course SET Course.description = "$description", Course.IDMetadata = "$IDMetadata"
Update
CourseUpdated
(a)
Command C
Perform SQL Update query for C
Find the next form B
Perform SQL Select query for C
Fill the B's controls
Send the filled form B to the browser
Extender for C:
onPreUpdate()
onPostUpdate()
onPreSelect()
onPostSelect()
(b)

Figure 4: Elements of the modeling technique. (a) UML state-transition diagrams for navigational specifications. (b) Extenders for implementation of non-typical behavioral use-cases.

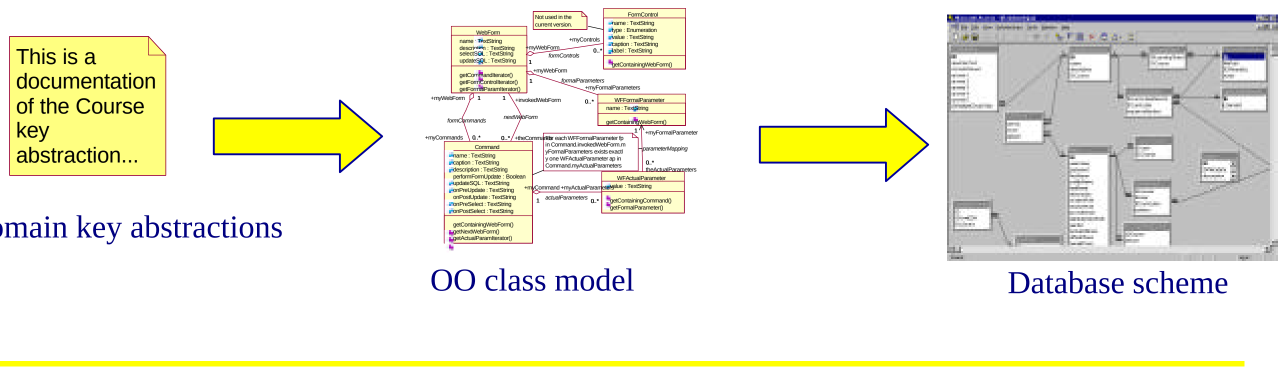
An adequate UML concept for specification of the described elements of the application are UML state-transition diagrams, where states represent Web forms, and transitions represent commands (Figure 4a). Thus, the navigational perspective of the application is easily specified in a standard way. Following the UML style, "structural" use cases that are directly oriented to the database access, are specified with the UML use-case [1] and state-transition diagrams.

In these diagrams, states and transitions are stereotyped [1], denoting that they represent Web forms and commands. They are tagged with the corresponding tagged values [1] as follows. States are tagged with SQL Select and Update queries, and formal parameters. Transitions are tagged with a caption that will be displayed at the starting Web form for that command, optional SQL Update queries, and actual parameters for the target Web form's formal parameters.

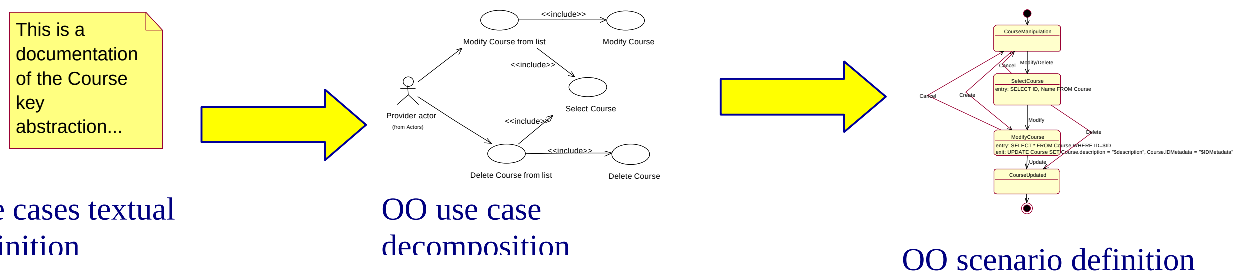
As stated in the introduction, the approach should provide an easy way to implement typical "structural" use-cases, but it should be open enough to support non-typical, "behavioral" use-cases. These use-cases should be implemented in a usual way, using common object-oriented modeling and programming techniques. For the purpose of implementation of such use cases, the so-called *extenders* are provided. An extender is a Java object assigned to a command. This object conforms to a predefined interface, which comprises of a set of operations, each of which is called at one step of the described command interpretation cycle (Figure 4b). This way, the user may attach one Java extender to a command and define its operations arbitrarily. The interpreter will invoke its operations at each step of the interpretation cycle. A built-in extender with all empty methods is attached to a command by default.

Following the UML style, an interaction diagram [1] is used to specify the scenario for a "behavioral" use-case. The use-case is then implemented in Java code, using common forward-engineering techniques [1]. The code is incorporated in a method of an extender, which will be invoked at a certain point of a certain command of the application.

The Process



(a)



(b)

Figure 5: The process. (a) The structural track of the process. (b) The behavioral track of the process.

The proposed process of application development follows the common object-oriented principles. The process has two tracks. The first one deals with the structural, and the second one with the behavioral aspect of the application (Figure 5). The process is started with a textual requirement specification, which, among other technical and non-technical details, contains two major parts.

The first part is the spoken-language yet formal description of the key abstractions of the domain, along with their roles, properties, and relationships. This part defines the vocabulary of the domain. The second one is the spoken-language description of use-cases, expressed in terms of interactions between the system and its actors [1]. Of course, the two tracks are tightly coupled, and they are developed in parallel, iteratively and incrementally.

The first developing track is started from the textual specification of the key abstractions (Figure 5a). From this specification, the object-oriented class model is developed. Then, following the usual rules, the class model is transformed into the relational database scheme. The latter transformation may be applied with or without help of a tool.

The behavioral developing track is started from the functional requirements. First, the use-case model is developed (Figure 5b), where the use-cases are decomposed by "include" and "extend" relationships [1]. Then, the scenarios for the use-cases that encompass interaction with the user are specified using state-transition diagrams, which define the navigation through the application. The typical "structural" use cases are specified in terms of SQL queries that directly access the database. Other non-typical, "behavioral" use cases are specified by object-oriented interaction diagrams [1]. Each interaction is attached to a transition in an interaction scenario defined by a state-transition diagram. This way, the state-transition diagrams represent the navigational skeleton of the application, which is "filled" either with SQL queries for the "structural," or with Java code for the "behavioral" use-cases. State transition diagrams are defined in a formal way and thus may be automatically forward-engineered into the application specification that is interpreted as described. Interaction diagrams may be also forward-engineered into the Java code using common forward-engineering techniques and tools [1]. This way, the application is modeled at a high level of abstraction, using standard UML object-oriented concepts, and is then forward-engineered into a running system.

The Implementation

Following the ideas and concepts described here, a research team from the University of Belgrade has implemented a supporting runtime environment [9]. The environment consists of an application interpreter and a meta database. The interpreter is implemented in Java using servlet technology. It accepts the "post" requests from the client, and performs the issued commands. The meta database stores the application specification that is interpreted by the servlet: the commands and the attached SQL queries. This way, the application skeleton is easily modified by changing the data in the meta database, without need for compilation. This makes the developing and debugging process easier. The application is flexible because it may be changed even dynamically, at runtime. The meta database is implemented using the standard relational database technique, which makes the environment completely portable.

Web pages are implemented as usual HTML pages, and may be developed and viewed by any standard Web page designer. Except for the three dedicated elements, other contents of the page by no means affect the runtime environment. The first dedicated element is the command issued from a page. It is identified by the interpreter as a form control with the name "Command." Thus, the commands offered at a page are implemented as radio buttons in a group named "Command." The value of the chosen radio button specifies the issued command. The second element is the "value" property of an HTML control in the page. If a form control should be filled with the value retrieved from the database by the interpreter, its "value" property must refer to the SQL Select query from which the control will be filled in. This reference is simply defined as a string "$<SQL Select query ID> $<Field name>". The interpreter will replace the "value" property of each such control with the corresponding field of the result of the query, prior to sending the page to the client. The third element is the formal parameters of the page. In order to make them available to the command's SQL Update queries when the page is abandoned, the interpreter stores the values of the page's formal parameters in its hidden controls. Thus, each page must contain hidden controls for its formal parameters. However, all these elements have the standard HTML format, which makes it readable to all standard Web design tools.

Similar holds for the parameterized SQL queries, which are stored in the meta database as strings. Prior to executing an SQL query, the interpreter replaces each reference to a local variable in the string with the value of that variable. The connection with the application and the meta databases the interpreter realizes through the JDBC standard.

A simple prototype forward- and reverse-engineering tool that may transform the higher level state-transition specifications into the data of the meta database, and vice versa, has also been implemented. The tool is able to generate HTML tags for the command controls, along with the SQL queries that are used to fill the meta database. The tool is aimed to be used as an add-in of the Rational Rose development tool [10], allowing a complete integration of Web application design into a usual tool-supported object-oriented development process.

An Example

Among several other minor projects and students' exercises, the proposed technique and process have been successfully used at the University of Belgrade in a large project of an educational Web system called Socratenon [9]. Socratenon is a complex system aimed for education in industry and academy. It is based on a complex conceptual model, which allows customization of curricula to particular students' cognitive states and preferences. Its rich functionality and user interface provide the impression of a virtual classroom.

Although it is still in its experimental phase, Socratenon is a large application. At the moment, it consists of about 130 Web pages and 200 commands (the full version will have over 1000 pages). However, it has been implemented by two students at the cost of about four men-months. Its development has approved the benefits from the proposed approach. First, the method is easy to understand and apply: the students needed only a couple of days of practice to become fully familiar with the method. Second, the application is easy to modify and debug. Finally, the application is extensible and fully portable.

However, as the whole process was not automated when the project started, the development suffered from the difficulty of organization of the complex model. When the modeling tool that will support the whole described approach and application generation becomes fully operational, this problem is expected to vanish. Standard UML techniques for hierarchical model organization (packages and diagrams) are expected to be an efficient cure for the problem of model complexity.

Comparison with the Related Work

The proposed approach will be briefly compared with the existing solutions from two perspectives: infrastructure (implementation) and modeling. From the perspective of infrastructure, the approach may be compared with other HTML-DBPL integration technologies (see the sidebar). The proposed infrastructure (the interpreter, commands, and database access) can be compared with other similar technologies that enable HTML extensions for database access, such as Microsoft Active Server Pages (ASP) or Java Server Pages (JSP). In contrast to other approaches, our approach uses a simple concept of referring to the SQL queries from the "value" properties of page controls, so the HTML pages may be created and viewed by any standard Web page designer. Besides, the SQL queries are stored in the meta database on the server, rather than in the page itself, which makes the application more compact, and the pages independent of the technology. The whole infrastructure is very simple, flexible, and portable, because it is based on all standard technologies: HTML, Java servlet, relational database, and JDBC. The technique, however, does not prevent usage of other standard technologies at the client side, such as Applets or JavaScript. Furthermore, the application is easily and arbitrarily extensible at the server side by any Java code in the extenders. Finally, the technology is open and not vendor-specific.

From the modeling perspective, since the whole modeling technique and process for developing Web-centric database applications have been proposed, the approach may be categorized in the model-driven category (see the sidebar). As opposed to other available approaches of that category reviewed in [2], the proposed approach follows the usual object-oriented habits of developing applications. It also uses only the standard UML concepts and extensibility mechanisms, rather than vendor-specific and dedicated modeling concepts. The approach is compatible with all other UML modeling concepts available to the developer, such as the class view, use-case view, interactions, etc. [1]. This enables the developer to design object-oriented applications with Web interface and with arbitrary complex functionality using usual object-oriented and UML methods.

As for the three perspectives proposed in [2] (structure, navigation, and presentation; see the sidebar), our approach supports the structural perspective in a usual object-oriented way. Namely, classes and their relationships, which are transformed into the database scheme afterwards, model the structural aspect of the application. The only navigational concept currently supported by our approach is the state-transition paradigm, i.e., the application is represented as a graph with explicit transitions from page to page. As suggested by other researchers, other navigational and presentational concepts are also needed for Web applications, such as indexed search [3] and composite/abstract contents definition [4]. This is not directly supported by our approach, but the definition of complex navigation and presentation is completely left to the developer to incorporate it in the pages. These are weaknesses of our approach, because the developer may not specify a unique presentational style for conceptually close application pages. On the other side, this may be an advantage, because a specialized page designer tool may be more suitable for customizing page outlook. However, our approach is fully open or orthogonal to the navigational and presentational conceptual models from other approaches. In other words, our conceptual model may be easily enhanced by the abstractions supporting navigation and presentation styles. Our current research investigates such possible conceptual models that may be automatically generated from the application's structural definition, and then transformed into the described executable model. The conceptual models and their automatic mapping may be easily specified using our originally developed metamodeling technique and tool, as it will be reported elsewhere.

Conclusions

The Web has evolved into a global and mature infrastructure for deploying large-scale enterprise applications, which assume universal and cross-platform database access. However, the development of these applications is too often at-hoc and not supported by rigorous, systematic approach, as in other application domains. The approach proposed in this paper is expected to contribute to the solution. It adapts the well-defined and broadly accepted object-oriented development principles for the Web environment. The proposed modeling technique and the process are fully compliant with the UML philosophy. The runtime environment is lightweight, efficient, portable, and extensible. The first experiences show that the approach is very easy to learn and apply. Finally, it is also flexible and open to further improvements.

Acknowledgements

The described method has been developed for the needs of the Socratenon project, which was conducted in collaboration with the University of Salerno, Italy, and under the leadership of Prof. Dr. Veljko Milutinovic from the University of Belgrade, to whom the author is grateful for numerous useful comments. The author is grateful to Dejan Marjanovic and Nenad Nikolic who have implemented the runtime environment. Milan Milicevic and Milan Trajkovic have implemented the Socratenon system. The author highly appreciates the fruitful collaboration with Prof. Dr. M. De Santo, Prof. Dr. S. Salerno, Mr. M. Marsella, and the development team from the University of Salerno, Italy.

References

[1] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1999

[2] Fraternali, P., “Web Application Development: Tools and Approaches,” *Proc. 7th World Wide Web Conf.*, April 1998, ftp://ftp.elet.polimi.it/pub/Piero.Fraternali/www_docs/www7/webtools.html

[3] Fraternali, P., Paolini, P., “A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications,” Tech. Rep. 1997-X, Politecnico di Milano, Dipartimento di Elettronica e Informazione, August 1997, http://www.ing.unico.it/autoweb

[4] Gellersen H.-W., Gaedke, M., "Object-Oriented Web Application Development," *IEEE Internet Computing*, Vol. 3, No. 1, January/February 1999, pp. 60-68

[5] Jacobson, I. et al., *Object-Oriented Software Engineering — A Use Case Driven Approach*, Addison-Wesley, 1992

[6] Manola, F., "Technologies for a Web Object Model," *IEEE Internet Computing*, Vol. 3, No. 1, January/February 1999, pp. 38-47

[7] Milutinovic, V., *Infrastructure for E-Business on the Internet*, to appear, CRCPress, New York, New York, USA, 2000, available at http://galeb.etf.bg.ac.yu/~vm

[8] Milutinovic, V., "Mini-Track on System Support for E-Business on the Internet," *Proc. HICSS-33*, Maui, Hawaii, USA, January 2000, pp. 159.1-159.3

[9] Nikolic, N., Trajkovic, M., Milicevic, M., Milicev, D., Marjanovic, D., Sokic, I., Milutinovic, V., De Santo, M., Salerno, S., Ritrovato, P., Marsella, M., "Socratenon — A Web-Based Training System with an Intellect," *Proc. HICSS-33*, Maui, Hawaii, USA, January 2000, pp. 160.1-160.10

[10] Rational Software Corporation, *Rational Rose*, http://www.rational.com

URLs of Selected Products

1. *Access*, Microsoft Corporation, http://www.microsoft.com/access/
2. *Active Server Pages*, Microsoft Corporation, http://msdn.microsoft.com/workshop/server/asp/ASPover.asp
3. *Cold Fusion Web Construction Kit*, Allaire Inc., http://www.allaire.com/products/ColdFusion/
4. *Crystal Report Print Engine*, Seagate, http://www.seagatesoftware.com/products/CrystalReports/
5. *Delphi Client/Server Suite*, Borland, http://www.borland.com/delphi/
6. *Designer*, Oracle, http://www.oracle.com/tools/designer/index.html
7. *Developer*, Oracle, http://www.oracle.com/tools/developer/index.html
8. *HahtSite*, HAHT Software, http://www.haht.com/
9. *Java Server Pages*, Sun Microsystems, http://www.java.sun.com
10. *NetDynamics*, NetDynamics, http://www.netdynamics.com/product/overview/
11. *PowerBuilder*, Sybase, http://www.powersoft.com/products/powerbuilder/
12. *StoryServer*, Vignette, http://www.vignette.com/
13. *Visual InterDev*, Microsoft Corporation, http://msdn.microsoft.com/vinterdev/

Sidebar: Database-Centric Web Application Development: An Overview

The work of Fraternali [2] provides an excellent classification and evaluation of tools and approaches for development of data-intensive web applications. The evaluation is driven by the following concerns:

- Software engineering: what support does a solution provide to the development of an application according to software engineering principles?
- Architecture: what support does a solution provide to application scale-up and reconfiguration?
- User-perceived quality: what kind of support does a solution ensures to make an application more adherent to the end-user's requirements in terms of structure, presentation, and navigation.

The survey emphasizes the following three design perspectives as relevant:

- Structure: what are the objects constituting an application and which semantic relationships connect them? In other words, what is the metamodel of the modeling approach?
- Navigation: How are the objects reached, i.e., what are the available paradigms for specifying the navigation through the application structure?
- Presentation: How is the information organized and presented on the screen?

According to the level of support the tools offer to the structured development of Web applications, they are classified into five categories. The first category contains visual HTML editors and site managers. The second one contains Web-enabled hypermedia tools. They both do not really support the development of large-scale Web-database applications, but are interesting because they pioneered many concepts (such as presentation styles and top-down site design) later integrated into more complex environments.

The third category is the first that explicitly addresses the integration of Web and databases, and includes HTML–database programming language (DBPL) integrators. These are proprietary intermediate programming languages and infrastructures, enabling either HTML extensions (in the form of page templates), or DBPL extensions that provide database access through the Web. However, these approaches address the integration at the programming language level, without support for higher-level software modeling. That is why they are used typically at the implementation level of the next category tools.

The fourth category takes quite a database-centric approach to Web-database integration, by addressing the migration of client/server, form-based applications to the Web. It includes Web form editors, report writers, and database publishing wizards. These tools offer a higher level of support than those in the previous category, but still concentrate on the implementation phase.

Finally, the last category is model-driven application generators [3, 11, 12, 13, 14]. They cover the whole developing process, from analysis to implementation. They use conceptual modeling of structure and behavior of the system, and code generation techniques to produce the implementation in the underlying infrastructure. They apply the fundamental principles of software engineering: the application is first modeled conceptually at a higher-level of abstraction, and then implemented through code generation. However, they differ in their approach to the definition of the conceptual abstractions (the metamodel). In some cases, they are drawn from the database world (i.e., entity-relationship), enhanced with the concepts to support the presentation and navigation requirements [14]. This might be a considerable limitation in the quality of the resulting application. On the other side, there are approaches very ambitious in defining the structural, navigational, and presentational metamodels as orthogonal perspectives to the application [3].

However, the available techniques of the last category have some drawbacks. One is the lack of an explicit compliance to the contemporary object-oriented modeling techniques, notations (such as UML), and tools. Besides, they are always either proprietary products or very complex, prototypic research efforts. This is sometimes accompanied with their limited extensibility and openness to other standard and widespread software engineering and programming techniques.

References

[11] Diaz, A., Isakowitz, T., Maiorana, V., Gilabert, G., "RMC: A Tool to Design WWW Applications," *Proc. 4th World Wide Web Conf.*, December 1995

[12] Kesseler, M., "A Schema-Based Approach to HTML Authoring," *Proc. 4th World Wide Web Conf.*, December 1995

[13] Schwabe, D., Rossi, G., "The Object-Oriented Hypermedia Design Model," *Communications of the ACM*, Vol. 38, No. 8, August 1995, pp. 45-46

[14] *Designer*, Oracle, http://www.oracle.com/tools/designer/index.html