WebODRA - A Web Framework for the Object-Oriented DBMS ODRA

Mariusz Trzaska
Chair of Software Engineering
Polish-Japanese Institute of Information Technology
Warsaw, Poland
mtrzaska@pjwstk.edu.pl

Abstract—**The modern Web requires new ways for creating applications. We present our approach combining a web framework with a modern object-oriented database. It makes it easier to develop web applications by rising the level of abstraction. In contrast to many existing solutions, where the business logic is developed in an object-oriented programming language and data is stored and processed in a relational system, our proposal employs a single programming and query language. Such a solution, together with flexible routing rules, creates a coherent ecosystem and, as an additional benefit, reduces the impedance mismatch. Our research is supported by a working prototype of the web framework for ODRA, a powerful object-oriented database management system. Furthermore, a simple web application (a forum) has been created to prove usefulness of the approach and the framework.**

Keywords-Web frameworks; Web tools; Web applications; Object-Oriented Databases.

I. INTRODUCTION

Modern web applications are usually developed using the three-tier architecture: a presentation layer, business logic (a middle tier) and a data tier. Each of them can be developed through a different technology and can utilize incompatible data models.

Typically, the middle tier is developed using an object-oriented programming language like Java, MS C#, Ruby, etc. However, the object-orientedness is a bit blurry. There is no single, well-accepted, specific definition or set of properties which determines features of an object-oriented programming language. Java and C# are pretty close to each other in that area, but for instance Ruby is based on different concepts, in particular, duck typing [1].

Contrary to implementation of the business logic, the data is usually stored using a relational database system. This causes a negative phenomenon known as impedance mismatch. During the years, numerous approaches have been formulated to solve or minimize the problem. Following Trzaska [2], the solution could use a single model both for the business logic and for the data. In this paper, we would like to employ the idea for a tool aiming at creating web applications. We propose a paradigm which uses the same high level language for working with data and implementing a business logic. In fact, those two utilizations are indistinguishable.

On the software level, our tool is implemented as a prototype system, called WebODRA, which integrates two independent components:

- The object-oriented DBMS ODRA with SBQL, a powerful programming and query language,
- A web server.

This approach increases significantly the level of abstraction, which reduces implementation time, decreases the number of errors and of course, completely eliminates the impedance mismatch. The programmers are able to focus on website's creation using a single, coherent technology.

The main contribution of the paper are the following:

- A new coherent paradigm of creating web application using the same high level programming and query language;
- A working prototype implementation of the approach containing object-oriented database, web server and all the necessary components.

The rest of the paper is organized as follows. To fully understand our motivation and approach, some related solutions are presented in Section 2. Section 3 briefly discusses key concepts of the utilized database and programming/query language. Section 4 presents the prototype implementation of the proposed web framework. Section 5 is devoted to a sample utilization of the prototype. Section 6 concludes.

II. RELATED SOLUTIONS

There are a lot of different web frameworks using many approaches. Just to name the most popular ones (by platform):

- Java: Apache Struts, Java Server Faces, JBoss Seam, Spring, Grails (Groovy);
- MS C#: ASP.Net, ASP.NET MVC, Kentico;
- PHP: CakePHP, Symfony, Zend;
- Smalltalk: Seaside [3];
- Ruby: Ruby on Rails, Sinatra.

They differ in some details but unfortunately share the same problems related to inconsistent models for programming languages and data. Even when an object-relational mapper is utilized the problems decrease not vanish. For instance, the Ruby's Active Record requires some additional information from a programmer to specify some non-mappable objects like arrays [4].

However, it is also possible to find solutions, where a website is developed using a single model. The next paragraphs contain description of such frameworks.

CouchApp [5] is a technology which allows to create applications delivered to the browser from CouchDB [6]. The applications are implemented using JavaScript and HTML5. The general idea is quite similar to our approach because CouchDB is a database management system. However, on contrary to our framework, the DBMS follows the NoSQL philosophy and allows to store documents in the JSON [7] format. There is also no query language similar to SQL or our SBQL (see section 3). All database queries are performed using dedicated API and JavaScript. The result is also returned as a JSON data.

Of course, every web application, needs a GUI. In case of CouchApp a GUI is created as a transformation of returned JSON data into some other format. For instance there are functions, which together with dedicated views, are able to convert the data into HTML, XML, CVS, etc.

Another approach to create a web application might employ the Model Driven Architecture (MDA) paradigm. The idea is to define a model (or models) and, through some transformations, receive a working application. There is a lot of such systems [8, 9, 10]. However, they are not widely utilized. One of the reason could be the amount and type of work which has to be done to get a working website. For instance [10], which is quite common for all MDA solutions, needs the following models and information to be precisely defined:

- UWA requirements,
- Information model,
- Navigation model,
- Transaction & operation model,
- Publishing model,
- Customization model,
- Logical models (UML diagrams): class, sequence.

Of course, the above information is not only required by MDA tools. Furthermore, they have to be provided by all websites' developers. However, it seems that the way of defining them, makes the difference in popularity.

The last described solution is not exactly a framework for programmers. Oracle Application Express [11] is more like a tool for a rapid web application development for the Oracle database. It is available, under different names, since 2000. The application requires a dedicated server and provides easy-to-use programming environment accessible via a web browser.

Most of its functionalities is available via dedicated graphical user interfaces, various wizards and helpers. But, still there are possibilities for using a programming language, namely PL/SQL. SQL, despite of thirty-years existence, and big popularity is the subject of heavy criticism. The SQL's flaws like: inconsistencies, incompatibilities between vendors and shortcomings of the relational model, decrease a value of the solution. Furthermore, application generators have some inherent shortcomings which make them less flexible (in terms of usability, functionality, GUI) than application developed by programmers. We believe that using a more powerful programming and query language together with an object-oriented model can formulate a much better approach.

III. THE ODRA DATABASE

As mentioned previously, our proposal for creating websites is based on utilization an object-oriented database together with a powerful query and programming language. DBMS could be used as a source for data and could be utilized to implement a business logic. For the purpose of the first requirement we need a database query language. However, because of the second necessity, we might need something more flexible and powerful: a fully-fledged programming language with imperative constructs. Both criteria are met by our prototype DBMS called ODRA.

ODRA (Object Database for Rapid Application development) is a prototype object-oriented database management system [12, 13, 14, 15] based on SBA (Stack-Based Architecture) [16]. The ODRA project started to develop new paradigms of database application development. This goal is going to be reached mainly by increasing the level of abstraction at which the programmer works. ODRA introduces a new universal declarative query and programming language SBQL (Stack-Based Query Language) [12], together with distributed, database-oriented and object-oriented execution environment. Such an approach provides functionality common to the variety of popular technologies (such as relational/object databases, several types of middleware, general purpose programming languages and their execution environments) in a single universal, easy to learn, interoperable and effective to use application programming environment.

ODRA consists of three closely integrated components:

- Object Database Management System (ODMS),
- Compiler and interpreter for object-oriented query programming language SBQL,
- Middleware with distributed communication facilities based on the distributed databases technologies.

The system is additionally equipped with a set of tools for integrating heterogeneous legacy data sources. The continuously extended toolset includes importers (filters) and/or wrappers to XML, RDF, relational data, web services, etc.

ODRA has all chances to achieve high availability and high scalability because it is a main memory database system with memory mapping files and makes no limitations concerning the number of servers working in parallel. In ODRA many advanced optimization methods that improve the overall performance without compromising universality and genericity of programming interfaces have been implemented.

The next subsections contain a short discussion of the ODRA main features including its query and programming language SBQL.

A. *ODRA Object-Oriented Data Model*

The ODRA data model is similar to the UML object model. Because in general UML is designed for modeling rather than for programming several changes have been made to the UML object model that do not undermine seamless transition from a UML class diagram to an ODRA database schema. The ODRA object model covers also the relational model as a particular case; this feature is essential for making wrappers to external sources stored in relational databases. Below, we present a short description of the main data model elements:

- Objects. The basic concept of the ODRA database model is object. It is an encapsulated data structure storing some consistent bulk of information that can be manipulated as a whole. A database designer and programmers can create database and programming objects according to their own needs and concepts. Objects can be organized as hierarchical data structures, with attributes, sub-attributes, etc.; the number of object hierarchy levels is unlimited. Any component of an object is considered an object too.
- Collections. Objects within a collection have the same name; the name is the only indicator that they belong to the same collection. Usually objects from a collection have the same type, but this requirement is relaxed for some kinds of heterogeneous collections. Collections can be nested within objects with no limits (e.g., in this way it is possible to represent repeating attributes).
- Links. Objects can be connected by pointer links. Pointer links represent the notion that is known from UML as association. Pointer links support only binary associations; associations with higher arity and/or with association classes are to be represented as objects and some set of binary associations. This is a minor limitation in comparison to UML class diagrams, introduced to simplify the programming interface. Pointer links can be organized into bidirectional pointers enabling navigation in both directions.
- Modules. In ODRA the basic unit of database organization is a module. As in popular object-oriented languages, a module is a separate system component. An ODRA module groups a set of database objects and compiled programs and can be a base for reuse and separation of programmers' workspaces. From the technical point of view and of the assumed object relativism principle, modules can be perceived as special purpose complex objects that store data and metadata.
- Types, classes and schemata. A class is a programming abstraction that stores invariant properties of objects, in particular, its type, some behavior (methods, operations) and (optionally) an object name. A class has some number of member objects. During processing of a member object the programmer can use all properties stored within its class. The model introduces atomic types (integer, real, string, date, boolean) that are known from other programming languages. Further atomic types are considered. The programmer can also define his/her own complex types. Collection types are specified by cardinality numbers, for instance, [0..*], [1..*], [0..1], etc.
- Inheritance and polymorphism. As in the UML object model, classes inherit properties of their superclasses. Multiple inheritance is allowed, but name conflicts are not automatically resolved. The methods from a class hierarchy can be overridden. An abstract method can be instantiated differently in different specialized classes (due to late binding); this feature is known as polymorphism.
- Persistence and object-oriented principles. The model follows the orthogonal persistence principle, i.e. a member of any class can be persistent or volatile. Shared server objects are considered persistent, however, non-shared objects of a particular applications can be persistent too. The model follows the classical compositionality, substitutability and open-close principles assumed by majority of object-oriented programming languages.

Distinction between proper data and metadata (ontology) is not the property of the ODRA database model. The distinction can be important on the business model level, but from the point of view of ODRA both kinds of resources are treated uniformly.

B. *Query and Programming Language SBQL*

SBQL (Stack-Based Query Language) is a powerful query and programming language addressing the object model described above. SBQL is precise with respect to the specification of semantics. SBQL has also been carefully designed from the pragmatic (practical) point of view. The pragmatic quality of SBQL is achieved by orthogonality of introduced data/object constructors, orthogonality of all the language constructs, object relativism, orthogonal persistence, typing safety, introducing all the classical and some new programming abstractions (procedures, functions, modules, types, classes, methods, views, etc.) and following commonly accepted programming languages' and software engineering principles.

SBQL queries can be embedded within statements that can change the database or program state. We follow the state-of-the-art known from majority of programming languages. Typical imperative constructs are creating a new object, deleting an object, assigning new value to an object (updating) and inserting an object into another object. We also introduce typical control and loop statements such as if…then…else…, while loops, for and for each iterators, and others. Some peculiarities are implied by queries that may return collections; thus there are possibilities to generalize imperative constructs according to this new feature.

SBQL in ODRA project introduces also procedures, functions and methods. All procedural abstractions of SBQL can be invoked from any procedural abstractions with no limitations and can be recursive. SBQL programming abstractions deal with parameters being any queries; thus, corresponding parameter passing methods are generalized to take collections into account.

SBQL is a strongly typed language. Each database and program entity has to be associated with a type. However, types do not constraint semi-structured nature of the data. In particular, types allow for optional elements (similar to null values known from relational systems, but with different semantics) and collections with arbitrary cardinality constraints. Strong typing of SBQL is a prerequisite for developing powerful query optimization methods based on query rewriting and on indices.

C. Virtual Updatable Views

Another interesting and quite unique ODRA property are updatable views. Classical SQL views do the mapping from stored data into virtual data. However, some applications may require updating of virtual data; hence there is a need for a reverse mapping: updates of virtual data are to be mapped into updates of stored data. This leads to the well-known view updating problem: updates of virtual data can be accomplished by updating of stored data on many ways, but the system cannot decide which of them is to be chosen. In typical solutions these updates are made by side effects of view invocations. Due to the view updating problem, many kinds of view updates are limited or forbidden.

In the ODRA project (basing on previous research) another point of view has been introduced. In general, the method is based on overloading generic updating operations (create, delete, update, insert, etc.) acting on virtual objects by invocation of procedures that are written by the view definer. The procedures are an inherent part of the view definition. The procedures have full algorithmic power, thus there are no limitations concerning the mapping of view updates into updates of stored data. SBQL updatable views allow one to achieve full transparency of virtual objects: they cannot be distinguished from stored objects by any programming option. This feature is very important for distributed and heterogeneous databases.

IV. Our Proposal

Basically, every web application, no matter how it is developed, requires the following set of logical components:

- A graphical user interface,
- A routing system,
- A business logic,
- Data to work with.

The above components could be implemented using various approaches. In some cases a programmer has to manually define them whereas other solutions use generators to create some of them automatically. Additionally, real world websites also require some static files: html templates, css, jpeg, etc.

We have decided to use pure programmatic approach which means that all necessary definitions are provided by a programmer. It may look like a lot of work, but thanks to the high level of abstraction, the amount of information is significantly reduced.

Another feature which simplifies development is MVC (Model – View - Controller) architecture which has been also utilized in many previously mentioned frameworks. Comparing to the other frameworks, our approach uses the same object-oriented model both for a business logic (Controller) and data (Model). This method not only removes the impedance mismatch but also allows using a powerful query and programming language for developing a business logic (behavior of the application). Furthermore, it is known that query languages operate on higher level of abstraction, effectively reducing the amount of code which needs to be written to achieve the same goals. For instance, a few tenths lines of Java code could be equivalent to a literally few lines of SBQL (or SQL). Not to mention performance and various optimizations, which are much more advanced in query languages.

Another very important area of a web framework is a graphical user interface. There are different methods to deal with the topic, some of them follows the MVC pattern. One of the most popular is using a server-side templating engine. A template contains an HTML code mixed with special tags, usually provided by the framework. In most cases, the tags allow to embed parts of a programming language (e.g., Java), mainly to insert some data (e.g., a list of products or customers). However, some programmers use them to implement additional functionality which duplicates the controller’s responsibility. Of course it is an incorrect application of the tags affecting maintainability of the code. At the end, tags are processed by an engine, a final HTML page is generated and sent to a web browser.

Figure 1 contains a simplified logical architecture of our prototype framework for developing web application called WebODRA. The framework consist of two principal parts:

- A web server. It is responsible for responding to incoming requests from a web browser. The implementation of the server is based on open source tool called Jetty [17];
- ODRA Database Management System. This is a standard instance of the ODRA server introduced in Section 3.

The following subsections describe each of the components (from Figure 1) in details.

A. Routing Module

In the center of WebODRA is a routing module which is responsible for a correct processing of incoming web requests. The module is driven by rules defined by a programmer. Each definition, written in SBQL (as an object with specific properties), contains the following information:

- Url. A regular expression which will be applied to the incoming request’s url. If there is a match, then the rule will be executed;
- Weight. It affects an order of the processing;
- Name. Human-readable name of the rule. It is especially useful during logging;
- Additional Data. The utilization of the additional data depends on rule’s kind;
- Rule’s Kind. The kind of the rule which affects processing:

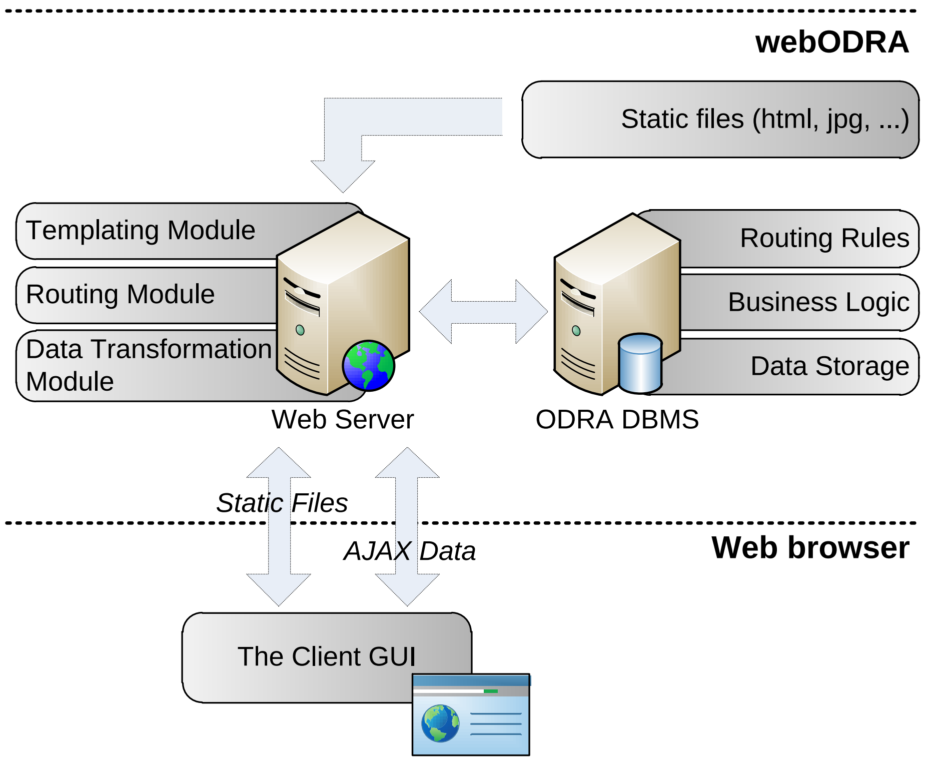


Figure 1. Logical architecture of WebODRA

- Passthrough. The web framework ignores those rules and they are processed by the Jetty server. They serve static files like: pictures, css, Java script, etc.;
- Data route. They contain a SBQL method ' s name to execute. The method will get all HTML form parameters entered by a user which makes possible processing them by a SBQL code. The result of the method is transformed (see further) and returned to the browser;
- Page route. An HTML page which is post-processed by our simple templating engine (see further).

B. *The Client GUI*

As mentioned previously, typical server-side web templating engines, may lead to overuse tags by implementing some business functionality. To prevent this we have decided to use a client-side GUI framework. The idea is based on embedding in a web page some (meta) information which will be used to present business data. We have chosen a framework called Knockout [18] which utilizes new HTML5 *data-* attributes. They allow to create custom attributes and store any information. The process of showing a web page contains two steps. First, a HTML page is downloaded from a server, containing the markers. Then the library sends an AJAX request to asynchronously retrieve necessary data which are then "injected" into the page.

The user data submission is performed on a similar rules. An asynchronous request is send to the server, triggering a Data Rule which process the provided data.

Standard website navigation is performed using a regular hyperlinks ("outside" the framework).

C. *Templating Module*

The templating module is responsible for a coherent look and fill of the entire website. It operates on a single master page which has a dynamic area fulfilled with some functional pages, i.e. a document repository, a forum, news, etc. For instance, the master page can contain a header, a navigation panel and a footer.

The process is triggered by Page Route rule. When a particular page is requested by a browser, the master page is applied, or which is more correct, the requested page is embedded in the master page and then returned to the browser.

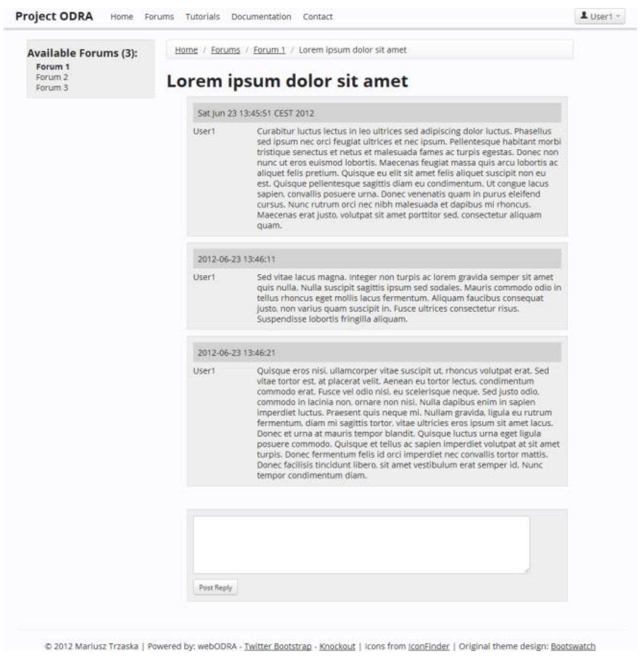
D. *Data Transformation Module*

When a Data Route rule executes a given SBQL method, the result could be any SBQL data type, i.e. a collection, a single object, a text. It needs to be processed to the format recognized by the Client GUI. The Data Transformation Module recursively converts the result into JSON [7] string, sends it back to the web browser where it is further processed.

V. EXAMPLE UTILIZATION OF THE FRAMEWORK

To verify usefulness of our approach, and the implemented library, we have decided to create a sample portal with a forum functionality (Figure 2). All business logic has been defined in SBQL language and the data is stored in the ODRA database. The prototype supports:

- Logon / logout with simple security model,
- Storing forums with topics and posts,
- Adding posts and topics,
- Responsive layout thanks to the Twitter Bootstrap.



Lorem ipsum dolor sit amet

Sat Jun 23 13:45:51 CEST 2012

User1 Curabitur luctus lectus in leo ultrices sed adipiscing dolor luctus. Phasellus sed ipsum nec orci feugiat ultrices et nec ipsum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec non nunc ut eros euismod lobortis. Maecenas feugiat massa quis arcu lobortis ac aliquet felis pretium. Quisque eu elit sit amet felis aliquet suscipit non eu est. Quisque pellentesque sagittis diam eu condimentum. Ut congue lacus sapien, convallis posuere urna. Donec venenatis quam in purus eleifend cursus. Nunc rutrum orci nec nibh malesuada et dapibus mi rhoncus. Maecenas erat justo, volutpat sit amet porttitor sed, consectetur aliquam quam.

2012-06-23 13:46:11

User1 Sed vitae lacus magna. Integer non turpis ac lorem gravida semper sit amet quis nulla. Nulla suscipit sagittis ipsum sed sodales. Mauris commodo odio in tellus rhoncus eget mollis lacus fermentum. Aliquam faucibus consequat justo, non varius quam suscipit in. Fusce ultrices consectetur risus. Suspendisse lobortis fringilla aliquam.

2012-06-23 13:46:21

User1 Quisque eros nisi, ullamcorper vitae suscipit ut, rhoncus volutpat erat. Sed vitae tortor est, at placerat velit. Aenean eu tortor lectus, condimentum commodo erat. Fusce vel odio nisi, eu scelerisque neque. Sed justo odio, commodo in lacinia non, ornare non nisi. Nulla dapibus enim in sapien imperdiet luctus. Praesent quis neque mi. Nullam gravida, ligula eu rutrum fermentum, diam mi sagittis tortor, vitae ultricies eros ipsum sit amet lacus. Donec et urna at mauris tempor blandit. Quisque luctus urna eget ligula posuere commodo. Quisque et tellus ac sapien imperdiet volutpat at sit amet turpis. Donec fermentum felis id orci imperdiet nec convallis tortor mattis. Donec facilisis tincidunt libero, sit amet vestibulum erat semper id. Nunc tempor condimentum diam.

© 2012 Mariusz Trzaska | Powered by: webODRA - Twitter Bootstrap - Knockout | Icons from IconFinder | Original theme design: Bootswatch

Figure 2. A sample forum developed using WebODRA framework

The example is also included in the library release available from our website.

VI. CONCLUSION AND FUTURE WORK

We have presented our approach to creating web applications using a single, coherent model utilized both for data and business logic. Thanks to the powerful query and programming language SBQL, a programmer stays on the same high level of abstraction, saving time and making less errors.

Our approach is supported by a working prototype framework called WebODRA [19]. Furthermore, we have created a sample portal with a forum functionality, proving that the idea is useful.

The contribution of this paper is based on quite new method for creating websites. To our best effort, we were not able to find a similar solution, directly employing power of a modern database to developing web portals.

We believe that this kind of solutions could be a valuable alternative to existing tools for creating data intensive web applications. Thus we would like to continue our research in that field, improving our framework to make them production-ready.

REFERENCES

[1] Duck Typing. http://rubylearning.com/satishtalim /duck_typing.html. Last accessed: 2012-08-18

[2] Trzaska, M.: The Smart Persistence Layer. ICSEA 2011: The Sixth International Conference on Software Engineering Advances. October 23-29, 2011 - Barcelona, Spain. ISBN: 978-1-61208-165-6. pp. 206-212.

[3] Perscheid M., Tibbe D., Beck M., Berger S., Osburg P., Eastman J., Haupt M., Hirschfeld R.: An Introduction to Seaside, Software Architecture Group (Hasso-Plattner-Institut), ISBN: 978-3-00-023645-7 (2008)

[4] ActiveRecord: http://ar.rubyonrails.org/classes/ActiveRecord /Base.html. Last accessed: 2012-08-20.

[5] CouchApp: http://couchapp.org/page/index. Last accessed: 2012-08-21.

[6] Anderson Ch., Lehnardt J., Slater N.: CouchDB: The Definitive Guide. O'Reilly Media, ISBN-13: 978-1449379681 (2010)

[7] JSON (JavaScript Object Notation): http://www.json.org/. Last accessed: 2012-08-19.

[8] Arraes Nunes, D., Schwabe, D.: Rapid Prototyping of Web Applications combining Do-main Specific Languages and Model Driven Design. Proceedings of the 6th International Conference on Web Engineering (ICWE'06; July 11-14, 2006, Palo Alto, California, USA).

[9] Ceri, S., Fraternali, P. and Matera, M. Conceptual Modeling of Data-Intensive Web Applications, IEEE Internet Computing 6(4), July/August 2002.

[10] Distante D., Pedone P., Rossi G. and Canfora G.: Model-Driven Development of Web Ap-plications with UWA, MVC and JavaServer Faces. Web Engineering Lecture Notes in Computer Science, 2007, Volume 4607/2007, 457-472, DOI: 10.1007/978-3-540-73597-7_38

[11] Williamson, J.: Oracle Application Express: Fast Track to Modern Web Applications (1st ed.), McGraw-Hill Osborne Media, ISBN 0-07-166344-4 (2012)

[12] Subieta K.: Stack-based Query Language. Encyclopedia of Database Systems 2009. Springer US 2009, ISBN 978-0-387-35544-3,978-0-387-39940-9, pp. 2771-2772

[13] Adamus R., Habela P., Kaczmarski K., Kowalski T., Lentner M., Pieciukiewicz T., Stencel K., Subieta K., Trzaska M., Wislicki J.: Overview of the Project ODRA. Proceedings of First International Conference on Object Databases (ICOODB) 2008, pp. 179-198

[14] Subieta K.: Stack-Based Architecture (SBA) and Stack-Based Query Language (SBQL). http://www.sbql.pl/. Last accessed: 2012-06-15.

[15] ODRA (Object Database for Rapid Application development): Description and programmer manual. http://www.sbql.pl/various/ODRA/ODRA_manual.html. Last accessed: 2012-06-15.

[16] Subieta K., Beeri C., Matthes F., Schmidt J.: A Stack-Based Approach to Query Languages. Proc. 2nd East-West Database Workshop, 1994, Springer Workshops in Computing, 1995, pp. 159-180.

[17] Jetty - Web Server: http://jetty.codehaus.org/jetty/. Last accessed: 2012-08-18.

[18] Knockout Framework: http://knockoutjs.com/. Last accessed: 2012-08-19.

[19] The WebODRA framework: http://www.mtrzaska.com/webodra. Last accessed: 2012-11-08.