WebLang :
A Language for Modeling and Implementing Web Applications

Olivier Buchwalder, Claude Petitpierre

Networking Laboratory
Swiss Federal Institute of Technology in Lausanne
CH-1015 Lausanne EPFL, Switzerland

E-mail: `{olivier.buchwalder, claude.petitpierre}@epfl.ch`

Abstract

Nowadays Web applications are being developed by the thousand, most of the time successfully. However, too many development projects exceed deadlines and budget, or become unreliable. The large number of existing technologies and the difficulty involved in the practical application of current development methodologies are partially responsible for this situation. Even, the main existing approaches, such as MDA, based on graphical models, do not fully define the architecture and the business behavior. In this paper, we describe a DSL language, WebLang, that abstracts the application components with a useful model, and remains sufficiently close to the technology to reduce the enormous gap that exists between UML models and implementation. The WebLang solution provides a compiler and an editing tool, which produces easily usable prototypes. Thus, the validation of an architecture is possible early in the development process, avoiding certain implementation and integration problems.

1. INTRODUCTION

Web applications and distributed systems are currently booming, and the developer community is very active in this domain. Application servers and specialized frameworks provide the essential components for Web application development.

Three-tier architecture has been adopted by many application and framework designers. However, despite the fact that this global architecture is known and adopted by many companies, there are too many development projects that exceed deadlines and budget, or become unreliable. This problem is not exclusive to the domain of Web applications, but the large number of existing technologies and specialized frameworks, developed for this domain, highlight the weakness of the available development methodologies.

Currently, there is no universal development process with a designing language that is commonly used and mastered by the majority of the developers. The available methods based on the graphic UML notation bring rules and structure, but they don't respond entirely to the practical needs of the companies [19, 7]. Frequently, the latter elaborate homemade methods, specialized frameworks or code generation tools, to fully master their business.

However, the development methods mainly converge to a consensus about the use of models to abstract the system details, and to provide simpler views of the system structure and business behavior in the early development stage. The general approach where models are used as key elements during the whole development process is called Model Driven Development (MDD [1, 2]). Recently, new development methodologies or approaches, based on this trend, turned up, such as the Model Driven Architecture (MDA [13]), Executable UML [16] or WebML[4]. The main differences between the existing modeling approaches are the level of abstraction of the models and the nature of the notation language. Moreover, the methods require powerful CASE tools for the edition of the models, and for their transformation to executable code.

The UML modeling notation is an important actor of software engineering; most approaches are based on UML, subsets of UML, or UML 2.0 extensions for a specific domain. UML provides a graphical notation and includes specific representations for describing the architecture or the behavior concerns in an abstract way. The recent UML 2.0 version provides user-defined extensions through the use of tagged value, stereotypes and constraints. They enable to insert new notations or terminologies, while keeping the basis of UML generic. In practice, the compatibility between different extended UML diagrams is not guaranteed anymore, and these extensions lead to the apparition of incompatible UML dialects [18]. Moreover, UML is a complex language and the extension mechanism is also quite complex. It is therefore not easy to understand how they will work in practice and how they will be manipulated and interpreted by tools to generate the code [8, 15, 17, 20].

The early implementation of Executable UML, such as Nucleus BridgePoint, uses a subset of UML plus certain rules to link the elements together. This extreme application of the MDD approach formalizes requirements and use cases, into a set of verifiable diagrams that can be executed, and expresses the specification of business behavior using action semantics, action languages and OCL constraints tags.

Several research efforts addressed the development of UML extensions for the domain of Web applications, such as the Conallen's approach [5], which defines UML extended diagrams for describing the client and server concerns of a Web application. Recently, the OMG released MDA, a standardized MDD approach, which is practically based on the UML extension mechanism. The MDA is a new way of developing applications and writing specifications, based on a platform-independent model (PIM) of the application and on automatic transformation into platform-specific models (PSMs) and into code. Several MDA tools exist, such as ArcStyler and OptimalJ, each of which introduces UML extensions for defining applications for the J2EE platform. However it's nontrivial to support and evolve correct PSMs for platforms such as J2EE or .NET. These platforms contain thousand of APIs and many of them are poorly documented. The application layer of middleware or additional libraries increase the difficulty to catch comprehensive models on which application can be built [20].

Beside UML extensions, more specific notations appeared for defining Web applications, such as Web Modeling Language (WebML), which was built on several previous Web design language proposals, including HDM[10], RMM, OOHDM, and Araneus. WebML provides a high level graphical notation and orthogonal models for designing structure, composition and presentation of a Web site; this approach is supported by the CASE tool WebRatio. WebML uses its own restricted notation and fails to express advanced composition and navigational constructs [11]. However, some research efforts have addressed the interaction between UML and WebML for defining behavior [14].

In this paper, we present WebLang, a Domain-Specific Language (DSL), which provides an adapted notation to define the architecture of J2EE Web applications. A DSL is a language designed to be useful for a specific domain, in contrast to general-purpose language (GPL), such as Java. WebLang brings abstraction for defining the structure and the business behavior of a Web application, and is intentionally closer to the target technologies than a general-purpose modeling language, such as UML. Indeed, in our opinion, the platform independent model is a theoretical good idea, but is practically difficult to apply for the wide area of existing technologies, and to maintain during the whole development process of an application. Our solution is designed to be used by architects and developers, who need to design well-defined architectures, and to generate rapidly testable prototypes on a specific platform.

The paper is organized as follows: in Section 2, the WebLang approach is presented. Section 3 shows the language specification, and finally Section 4 presents an example of a realistic application defined with WebLang.

2. WEBLANG APPROACH

WebLang is a Domain-Specific Language that makes it possible to define Web applications. The main motivation behind WebLang is to abstract the application components with a useful model, but to remain sufficiently close to the technology to reduce the enormous gap that exists between a PIM model and implementation. The WebLang approach expects to provide a simple and realistic method for designing the architecture of a Web application and a usable tool for generating a testable prototype.

The WebLang development process follows the MDD approach and brings a language model as key element of the development. The language syntax is oriented towards being naturally editable for a human in comparison with XML, which is more adapted to the machine. A WebLang application is defined by the assembling of several components that can specify each structural properties, business logic, and interconnections with other components.

The WebLang tool checks and compiles the model, and then generates the application in one atomic action. This approach is easier to implement than the incremental generation of application fragments. Furthermore, it provides the developers with a well-defined environment, where the whole application is defined with a unique and centralized model. All the generated files are standard and can be freely modified by the developer. The tool is integrated in the IBM Eclipse IDE, and is currently available for the J2EE JBoss platform, but the approach is extendable to other servers or technologies, by extending the templates or implementing new adapted modules.

2.1. Language Structure

The use of our own human-usable language allows us to freely specify the properties inherent to the technology without being limited by existing standards, such as the OMG Human-Usable Textual Specification (HUTN [12]), which is based on the OMG standards. We call a WebLang component a module, and a valid WebLang architecture is defined by a set of module instances. The following grammar presents the syntax of a typical active component of a WebLang application.

```
<module_type> <moduleName> {
  <destination> <path>;
  ( <specialized_feature>; )*
  ( <field_type> <field_name>; )*
  ( <method_declaration> )*
}
```

The global syntax looks like well-known GPLs, such as Java or C, while the inner specification of a WebLang module is based on the target nature, in abstracting and simplifying the technology details.

2.2. Introduction of Business Logic

One of the benefits of using a text language as input model is the possibility to introduce the business logic directly and naturally into the WebLang source code. Currently, the language is dedicated to the J2EE platform; therefore a full Java 1.5 parser has been integrated into the tool by using JavaCC. The workflow, the Web-flow (3.2.3) or the business behavior can be inserted directly into the module declarations using the target platform language as shown in Section 3.1.

When the language syntax is parsed, the Java zones are checked and processed by the tool before being introduced into the generated files. During the parsing phase, certain relevant tokens or groups of tokens are recognized and stored. Then, when the module instance is processed, this Java-related information can be easily handled. Compared with Wizard-based and graphic MDA tools, this approach is simpler and offers greater flexibility. An early implementation of some .Net modules has shown that a C# parser is not essential, but brings facilities for implementing the tool, especially for checking the syntax.

2.3. Model Nature

The WebLang language is clearly a Platform Specific Model, which abstracts the implementation details of the target platform, but specifies precisely the relevant features of the target technologies. This model specification is text-based, and is defined in a constant and centralized way, contrary to several UML graphic models.

The designer can create WebLang models in which the full application prototype is defined without ambiguities. For instance, with WebLang, the architect decides at an early stage whether a service must be handled by a Servlet (3.2.3) or by a Session Bean (3.2.2), and the definition of the corresponding business behavior is easily and precisely defined in both situations. In our opinion, many implementation and integration problems result from models that are too abstract, incomplete or not fully mastered by the developer. When the components of the application are fully defined and when a usable prototype can be tested initially, the development process becomes less risky.

WebLang does not provide a platform independent model, such as the attractive MDA PIM. The latter provides facilities for switching between different technologies at any moment of the development. However, we think that currently the definition of a Web application with a PIM is too difficult to master and is not efficient enough. A PIM does not allow the specification of a significant and crucial part of most applications. For instance, the PIM cannot define the simplest application that every developer begins to write, *Hello World*, because there are no I/O libraries defined in the OMG standards [8].

On the other hand, the PSM and the WebLang approach also provide facilities for switching between different technologies, because the model synthesizes both the application structure and the business behavior; the conversion between PSMs is trivial for most cases. Moreover, when the PSM conversion would require a deep reengineering, the corresponding PIM would certainly be not trivial to specify. The definition of a PIM requires the additional intellectual work of an architect or a developer, and there is no assurance that a defined PIM would be reusable in the context of a new platform.

The WebLang model is defined with a unique input structure; the whole information concerning architecture and behavior is centralized in the same model. On the contrary, UML diagrams are not linked together by nature; each of them defines a fragment of structure or behavior of the application. This fragmented graphic approach is interesting for documentation or discussion on a specific part of the system, but it implies difficulties for developers and architects, who must deal with a set of diagrams, between which the interconnections are not trivial.

3. WEBLANG LANGUAGE

3.1. A Simple Example

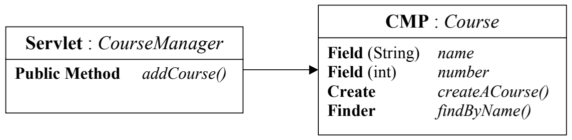
Servlet : *CourseManager*
Public Method *addCourse()*
CMP : *Course*
Field (String) *name*
Field (int) *number*
Create *createACourse()*
Finder *findByName()*

Figure 1. Two basic Web components

The figure 1 presents the abstract structural schema of a simple J2EE application. This application contains a Servlet instance (3.2.3), which is a Java Web service that usually returns an HTML response page, and a J2EE CMP element that provides an object abstraction of a database table. The corresponding WebLang model is presented in the following code sequence. This definition not only abstracts the application architecture, but also specifies the application business behavior in the form of Java business code.

In this example, the Servlet method searches for a *Course* CMP bean instance. If the instance is found, its id is displayed on the output page; otherwise the *Course* bean is created.

```
servlet CourseManager {
  package pack;
  public void addCourse(String name, int id,
                                  PrintWriter out){
    try {
      Course c = courseHome.findByName(name);
      out.println("Course finded:" + c.getId());
    }catch (javax.ejb.FinderException fe) {
      courseHome.createACourse(name, id);
      out.println("Course created:" + id);
    }
  }                                                   Public Methods
}
cmpbean Course {
  package pack;
  String name;
  int id;
  public Course createACourse(String name, int id);
  public Course findByName(String name) {
    query = "SELECT OBJECT (o) FROM Course o
             WHERE o.name = ?1"
  }                                                   Fields + Methods
}
```

The WebLang generator reads these model inputs and produces mainly Java classes annotated with XDoclet tags, HTML or JSP pages, and XML script files.

3.2. Module Details

This section presents a syntax overview of the main modules. For the sake of simplicity, the specification of the modules is presented by examples rather than using BNF grammars.

3.2.1. Java Common Processing

The parsing of the following modules expects some mandatory or optional tokens, and is able to run an external Java 1.5 parser on specific blocks. The main Java parsing functions used are *BlockStatement*, *ClassBodyDeclaration* and the more localized *FieldDeclaration* and *MethodDeclaration*. Moreover, some parser extensions have been made for handling some specific declarations, such as the bean finder and creation methods.

```
Course c = courseHome.findByName("math");
Course c = courseHome.createACourse("math",12);
```

Each module processes the parsed Java information differently according to its needs, but a common processing is applied to most supported modules. The preceding code sequence shows the abstracted WebLang syntax for finding and creating a bean instance. These expressions can be used in all Java blocks, and are transformed by our tool into executable code compatible with the module in which they are defined. The bean type is automatically replaced by a reference to the remote or the local object proxy, and the bean home identifier by a reference to the corresponding home proxy object.

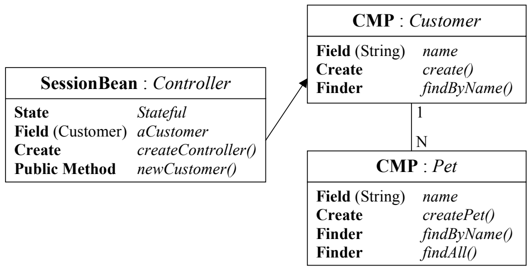

Figure 2. Interaction between three EJB modules

3.2.2. J2EE Enterprise Java Beans

WebLang is able to produce the J2EE EJBs classes under the specification of EJB 2.0. The figure 2 and the following code sequences show a typical interaction between three EJB modules, a Session bean and two CMP beans.

CMP Bean

A CMP bean (Container Managed Persistence Entity Beans) represents the object abstraction of a table in the database. Each instance of a defined CMP bean can be created, deleted and retrieved like a row of a database table. Some relations to other CMP beans can be defined after the package entry.

```
cmpbean Pet {
  package data;
  String name;                                    CMP Fields
  public Pet createPet(String name);
  public Collection findAll() {
    query = "SELECT OBJECT (o) FROM Pet o"
  }
  public Pet findByName(String name) {
    query = "SELECT OBJECT (o) FROM Pet o
             WHERE o.name = ?1"
  }                                               Finders
}
cmpbean Customer {
  package data;
  relations (Pet = 1:N);                          Relation 1:N
  String name;
  public Customer findByName(String name) {
    query = "SELECT OBJECT (o) FROM Customer o
             WHERE o.name = ?1"
  }
  public Customer create(String name);            Create
}
```

CMP Relations: a relation must define the target bean name and the relation nature: *1:1, 1:1 target, 1:N* or *N:M*. When a relation is defined, the database table is configured with XDoclet tags, and getter-setter functions are generated in the class file.

CMP Fields: the Java class members are registered as CMP bean fields; they are mapped as table attributes and getter-setter methods are inserted in the CMP class.

CMP Finder Methods: some EJB finder methods are definable as specific method declarations. Their names must begin with the keyword *find* followed by an optional identifier name. The parameters may have either simple types or fully qualified types. This method declaration expects the definition of an EJB-QL query (see Sun EJB specification, Chapter 11).

EJB Creation Methods: some EJB creation methods are definable as specific method declarations. Their names must begin with the keyword *create* followed by an optional identifier name. The parameters must correspond to a subset of the attributes, placed in any order.

Java Methods: the method declarations are parsed and inserted in the generated class with the common transformations 3.2.1. The public methods are referenced in the bean's proxy and become accessible from the other modules.

Session Bean

The SessionBean is a service supplier, managed by the server and associated with a client session.

```
sbean Controller {
  package control;
  state Stateful;                                      Stateful
  Customer aCustomer;
  ControllerSession createController();
  public void newCustomer(String name){                Public Method
   aCustomer = customerHome.create(name);
  }
}
```

Session Type: the session can be declared *Stateful* and assigned permanently during a client session or *Stateless* and reassigned to other clients.

EJB Creation Methods: some EJB creation methods are definable in the module declaration, like for the CMP beans.

Java Methods: the method declarations are parsed and inserted in the generated class, like for the CMP beans.

3.2.3. Web Server Modules

The following client modules, Servlet and Struts, are by nature local to the server; they are typically hosted in a Web server, such as Apache Tomcat. Some others modules are remote such as the Java Client (see 3.2.4).

Servlet

A Servlet module is presented in Section 3.1. The generation of a module produces a standard Java Servlet class and a requesting HMTL page.

Java Methods: the method declarations are parsed and inserted in the generated class with the common transformations 3.2.1. Moreover, the public methods are processed specifically; they become accessible by calling the generated Servlet. For each public method, a requesting HTML form with all method's parameters, is included in the generated page.

Struts

A Struts module defines a state chart based on the Apache Struts framework. The following code sequence represents the business and the presentation layers of the application example, reviewed in Section 4.

```
struts Control {
 package store;
 stateMachine {
  statedef State_0 (pList);                          States definition
  statedef State_1 (pConfirm);
  switch (sessionState) {                            Java Block - FSM
   case State_S:
     makeForm(browseForm);
     sessionState = State_0;
     break;
   case State_0:
     clientForm.setNbOfItems(
       browseForm.getPetFormN().size());
     sessionState = State_1;
     break;
   case State_1:
     if (submit.equals("Confirm")) {
       String cfname = clientForm.getName();
       Customer cust;
       try {
         cust = customerHome.findByName(cfname);
       }catch(FinderException fe) {
         cust = customerHome.create(cfname);
       }
       for(PetForm pform: browseForm.getPetFormN()){
         if(pform.getFlag()){
           String pfname = pform.getName();
           cust.addPet(
             petHome.findByName(pfname));
         }
       }
     }
     sessionState = State_0; break;
  }
 }
 page pList {forms(browseForm.petFormN[](OK)); }   Pages
 page pConfirm {forms(clientForm(Confirm),
                      cancelForm(Cancel)); }
 private void makeForm(BrowseForm browseForm){      Java Method
   for(Pet pet : petHome.findAll()){
     PetForm pForm = new PetForm();
     pForm.setName(pet.getName());
     browseForm.getPetFormN().add(pForm);
   }
 }
}
form BrowseForm {              form PetForm {              Struts Action Forms
  package pack;                  package pack;
  forms (petFormN[]);            fields(String name,
}                                       boolean flag);
form ClientForm {              }
  package pack;                form CancelForm {
  fields(String name,            package pack;
         int nbOfItems);       }
}
```

The WebLang Struts module provides a safe Web application controller, in which the state of the client is preserved during the whole session. Some additional Java class files are automatically included in the project to manage the state machine. The generation of the module produces a Struts Action class, which contains the *state chart*. A JSP page is also produced for each declared *page*, and a Struts ActionForm class for each defined *form*.

State Definition: The definitions of the *states* are declared with a state name and an associated page name; $State0$ is reserved as the starting state.

Java State Chart: the state chart code is parsed by the Java *BlockStatement* function. Typically, a Java switch statement is anticipated to assign a new state according to the requested *form* values. However, the developer is free to manage the state evolution and to initialize the *form* values with the Java language.

Page Definition: each *page*, referenced in the state definition, must be declared formally, and is generated as a JSP page. A *page* must define a name and the list of included *forms*.

Forms: a Struts Action Form is basically a Javabean; it is used to host the *page* input and output data. The *form* defines some local fields and some included *sub-forms*.

3.2.4. Other Supported Modules

The WebLang language defines other modules by following the same approach: Message Driven Bean (MDB), Java Class, Java Client, RMI Object, Synchronous Java Class, Hibernate bean, JSP or HTML.

In addition to the presented proxy connections, WebLang supports the asynchronous exchanges between most modules in using the Java Messaging Service (JMS) specification.

4. DESIGN WITH WEBLANG

In order to demonstrate the value of our solution, we present in this section a realistic prototype of an elementary Web-store application. This three-tiered architecture provides a Web access to a *pet* database, and allows customers to make orders.

This application is composed of a database for storing persistent data; it mainly implements a Web service for handling the client requests, which accesses the system by using a Web browser. The full source code of this application is composed of the Struts and the CMP preceding code examples (3.2.3, 3.2.2). This application prototype is fully defined by WebLang with approximately fifty lines.

4.1. Scenario

A client accesses the *pet* database with a Web browser. The first page, displayed by the server, shows the list of all available *pets*. The client selects the elements that he wishes to order, and then submits his selection to the server. The server temporarily keeps the client selection, and responds with a confirmation page, showing the number of ordered *pets* and an empty field name. The client enters his name and clicks on the confirm button to finalize the order or on the cancel button to return directly to the starting page. If the order is confirmed, the selection is stored into the database; the customer instance is created if necessary and all selected *pets* are added to the customer's list.

4.2. Application Components

This application defines two CMP beans to store persistent data; the *Pet* table must be filled externally. The *Customer* CMP is linked to the *Pet* CMP with a 1:N relation. The Form components are used to handle the temporary data from and to the Pages. The Forms are basically Javabean objects with some Struts additions. The heart of the application is managed by the Struts component *Control*. It is mainly composed of a state chart that processes the client requests and accesses the database in a reliable way.

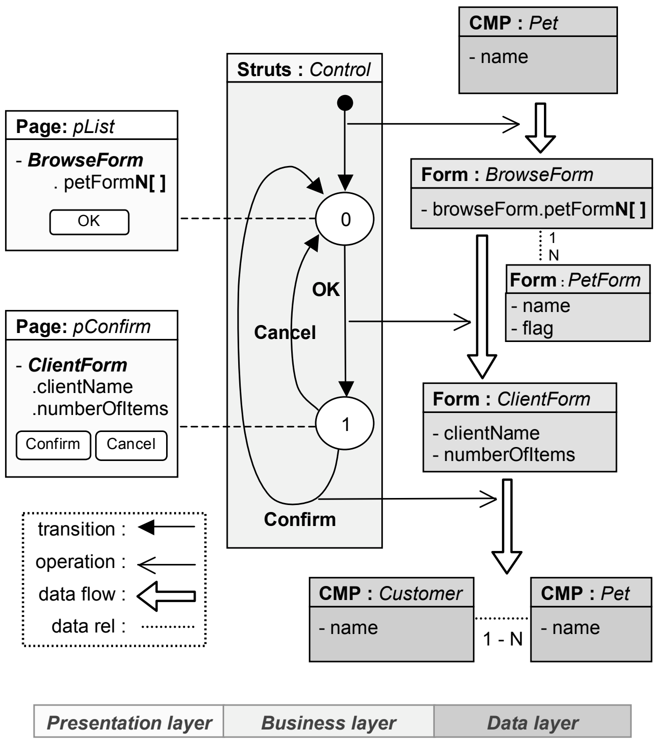

Figure 3. Elementary Web-Store

4.3. Diagram Overview

The figure 3 represents the architecture of this application. The WebLang components are all represented in the same diagram, as well as the principal behavior of the application. Currently, this diagram notation is informal, but it is for the most part usual for a developer. This schema is based on the following UML diagrams: state-chart, dataflow and collaboration diagrams. The boxes represent the

WebLang strongly typed components. The diagram is separated into the common three layers (presentation, business and data). The Struts instance is clearly the controller of the application, and each state, declared in the Struts module, is linked to a page. The data-flow evolution, represented in the schema by double arrows, is controlled by the transition of the state chart.

The business functionalities need to be extended to include the requirements of a commercial store, but the architecture defined by WebLang is really close to the final product. The generated prototype can be tested early in the development process, and the business functionalities can be developed on concrete and validated foundations.

Although we think that the edition of graphical models is less efficient and less flexible than editing a textual language with a good editor, the diagram visualization is interesting for documentation and development discussion, and we plan to release this tool extension.

5. CONCLUSION

We have presented a new approach for designing Web applications. The method is based on a language, WebLang, that is intentionally close to the technology. Our goal is to abstract the behavior and the structure of the application by providing well-defined components.

The advantage over some other modeling methods is that the WebLang language can specify the whole architecture of an application. The model is natural for a developer and more flexible than some graphic editions. The associated tool can compile and produce a usable prototype in one action. The architecture can then be tested early in the development process, thus avoiding certain implementation and integration problems.

Currently, WebLang only defines architecture for the Java J2EE platform. However, the concept is easily extendable to other technologies by adding new module declarations. We plan to support Web services soon, and to provide adapted modules for the Microsoft .Net platform in the future. Since the EJBs have no equivalent in the Microsoft specification, a .Net architecture is more difficult to devise in a standard and powerful way. The WebLang approach can provide a useful components set to help the developers to compose reliable applications.

The imminent introduction of Web services will transform WebLang into a SOA (Service Oriented Architecture [3]) designing tool, which will allow to compose heterogeneous application in a simple way.

5.1. Future Work

We are also implementing a generic designing tool, which extends the WebLang modeling approach to all kind of applications. This tool will allow the developers to specify and create their own DSL language, parser, generator and editor adapted to their needs.

This approach enables Language Oriented Programming [6, 9], which is a style of programming in which, rather than solving problems in GPLs, the programmer creates adapted DSLs and solves the problem in these languages.

References

[1] C. Atkinson and T. Kuehne. Aspect-oriented development with stratified frameworks. *IEEE Software, Volume 20, Issue 1, Jan.-Feb. 2003 Page(s):81 - 89*, 2003.

[2] C. Atkinson and T. Kuehne. Model-driven development: a metamodeling foundation. *IEEE Software, Volume 20, Issue 5, Sept.-Oct. 2003 Page(s):36 - 41*, 2003.

[3] J. Bloomberg. Principles of soa, 2003. ZapThink - ADT-mag.com.

[4] S. Ceri, P. Fraternali, and A. Bongio. Webml: a modeling language for designing web sites. *Computer Networks (Netherlands)*, 33(1–6):137–157, 2000.

[5] G. Costagliola, F. Ferrucci, and R. Francese. Web engineering: Models and methodologies for the design of hypermedia applications, 2002.

[6] S. Dmitriev. Language oriented programming: The next programming paradigm, 2004.

[7] M. Fowler. *UML distilled: A brief Guide to the Standard Object Modelling Language*. Object Technology series. Addison Wesley, 3rd edition, 2004.

[8] M. Fowler. Language workbenches and model driven architecture, 2005.

[9] M. Fowler. Language workbenches: The killer-app for domain specific languages?, 2005.

[10] F. Garzotto, P. Paolini, and D. Schwabe. Hdm a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.*, 11(1):1–26, 1993.

[11] E. Gorshkova and B. Novikov. Exploiting uml extensibility in the design of web applications, 2003.

[12] HUTN. *Human-Usable Textual Notation (HUTN) specification (OMG)*, 2002.

[13] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture-Practice and Promise*. Addison-Wesley, 2003.

[14] M. Matera, A. Maurino, S. Ceri, and P. Fraternali. Model-driven design of collaborative web applications. *Softw. Pract. Exper.*, 33(8):701–732, 2003.

[15] A. McNeile. Mda: The vision with the hole?, 2003.

[16] S. J. Mellor and M. J.Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley, 2002.

[17] C. Petitpierre. *Software Engineering: The Implementation Phase*. EPFL-Press, 2006.

[18] B. Rumpe. Executable modeling with uml. a vision or a nightmare? *Issues and Trends of Information Technology Management in Contemporary Associations, Seattle*, pages 697–701, 2002.

[19] D. Thomas. Uml - unified or universal modeling language? *Object Technology, vol. 2, no. 1, January-February*, 2003.

[20] D. Thomas. Mda: Revenge of the modelers or uml utopia? *IEEE Software, vol. 21, no. 3, pp. 15-17*, 2004.