Journal of Universal Computer Science, vol. 12, no. 9 (2006), 1104-1117
submitted: 31/12/05, accepted: 12/5/06, appeared: 28/9/06 © J.UCS

UML-Based Modeling of Data-oriented WEB Applications

Attila Adamkó
(University of Debrecen, Faculty of Informatics
Department of Information Technology, Hungary
adamkoa@inf.unideb.hu)

Abstract: Recently a growing demand has arisen for methods for the development of small- and medium scale Web Information Systems (WIS). Web applications are being built in a rapidly changing environment where requirements are usually unstable. Short-time design and implementation are needed in response to the new technologies. Our work focuses rather on the design and construction of Web applications, than management. Flexibility is a major requirement in such applications, and also in a database-backed environment for the structure and presentation of the sites.

We propose a systematic design method for Web applications which takes into account the data-oriented aspects of the application. The method is based on a UML profile adapted to the problem domain by means of stereotypes as well as a strategy for generating code templates from such models. We provide a method to derive the navigation model from the structural model of a Web application. We will also show guidelines for the development of the Data Layer of data-oriented Web application. Moreover, why to divide the business logic layer into two parts: the pure application logic for managing the workflow of the application and the storage logic responsible for the data structures.

Rapid development is enabled by providing roundtrip engineering capabilities with support for automatic code generation. We will show the role of XML: why to use XML to support both the reuse of content and context-dependent delivery.

An advantage of the proposed methodology is that several steps can be performed is a semi-automatic way providing rapid development and prototyping.

Keywords: Data models, MVC design pattern, UML, Web application, Web modeling techniques, XMI, XML, XSLT
Categories: D.2.2, D.2.3, D.2.10, H.4.3

1 Introduction

In several areas of life, a new form is available with the continuously growing availability of the Internet. With the evolution of technologies the early static web sites are changing into Web based distributed applications. However, reorganization and new developments require knowledge and integration of several technologies. For this reason the development of dynamic Web sites is often performed by teams consisting of graphics designers through software developers. Several complex methodologies are available to support their work, but these methodologies would introduce expensive overhead into these – mainly ad hoc – projects, which cannot be economically compensated by the process's benefits. Another important economical factor is the production time. To reduce time, development groups are often disregarding methodological approaches and system plans resulting incomplete documentation and difficulty maintainable systems. Moreover, as we could read in [Gingeri, 03], "*Most Web developers pay little attention to requirements elicitation and analysis, development methodologies and process, quality, performance evaluation, configuration and project management, and maintainability and scalability. Furthermore, application development heavily relies on the knowledge and experience of individual (or a small group of) developers and their individual development practices rather than standard practices. These systems also lack proper testing and documentation.*" The process of learning how to develop Web applications has just begun. Web Engineering is a new and still evolving discipline.

These aspects outline the demand for lightweight methods in the development of small- and medium size Web applications. An important factor in the development for such Web applications is the support of successful communication – using a common language – to avoid misunderstandings and expensive redesign. The UML (Unified Modeling Language) [UML, 01] fulfills these requirements by providing a family of intuitive notations and diagrams which are could be used to describe software systems at a high level of abstraction. These models have to be focused on the information relevant to the different roles in the development team. In many cases, we can distinguish between the domain expert, who has knowledge about the business processes behind the Web application, the graphic designer, who is in charge of the creation of the user interface, and the developer, who has to build a working software system based on the work of his partners. Furthermore, we have the customer, who has little knowledge about the technical realization of the project, but the diagrams and models helps to synchronize the requests with the software system in the early stages of the development.

Performing early demonstrations is another important factor in the course of the communication with the customer. Therefore, the methodologies should support code-generation, which could shorten production time too. The models should formulate the complete structural description of the website, resulting a working, but incomplete prototype.

Following these guidelines, the presented approach helps in the development of small- and medium sized data-oriented Web applications using UML. Use-cases, activity- and class diagrams are used to describe the behavior and the structure of the Web application. These models make it possible for the team members to investigate the system of different aspects. Naturally, UML requires additional stereotypes and data types to model complete Web sites.

1.1 Related work

There are numerous different approaches to the modeling of Web applications. Some of them focus on the modeling notations, while others focus on the development process. Conallen [Conallen, 03] defines a UML profile for Web application design introducing stereotypes for components, classes, methods and associations in order to distinguish, for example, server components, client components, etc. The Web Application Extension (WAE) to UML enables to represent Web pages and other architecturally significant elements in the model alongside the "normal" classes of the model. WAE uses own graphical notations to express these components. His book explains a complete lifecycle for Web application development including requirements elicitation through user experience modeling, but this approach seems to be a very implementation oriented direction.

Koch and others [Hennicker, 00] described a hypermedia extension to the UML to model Web applications, focusing on the navigation and presentation aspects. The conceptual model consists of a class diagram identifying the objects of the problem domain and their relations. An extended class diagram specifying navigational nodes describes the navigation structure and links using own notations. The presentation model defines the structure of the user interface and their behavior by state diagrams. This approach seems to be oversized for small applications and moreover our presented methodology leaves this phase untouched, because the applied XML technologies makes it possible to separate the content from the presentation leaving user interface design to the graphic designer.

The Web Modeling Language (WebML) [Ceri, 00] is a notation for specifying complex Web sites at a conceptual level. It distinguishes between a structural model, a composition model, and the topology of links between pages. The structural model expresses the content of the application, in terms of the entities and relationships. Entities have attributes, with an associated data type. Properties with multiple occurrences are described using components. To certain extent, it is quite similar to the entity-relationship (E-R) model widely used in database design. The composition model states what the pages in the Web applications are and how the content units form each page. There are six different types of units. The navigation model illustrates the navigation links between pages and content units to form the hypertext. Further, WebML has a presentation model and a personalization model. Additionally, a software tool, called WebRatio is developed to support this modeling methodology.

WDSM (Web Site Design Method) [WSDM, 01] is a user-centered approach. The starting point is a set of potential visitors of a Web site classified into user classes. The available data is modeled based on the information requirement of the users. WSDM describes a four-phase process: user modeling, conceptual design, implementation design and implementation. Within modeling, the information requirements of different user classes with their different perspectives are formalized. How the different users can navigate through the Web site is described within the navigation model.

Naturally, Web application development is not only supported at the conceptual level, there are several software tools available, like Together Designer from Borland and Rational Rose from IBM. Although these tools claims to support the whole lifecycle in the development process, they offer only basic help for modeling the specialties of Web applications because they only include low level implementation elements like servlets, Active Server Pages or HTML pages. Several abstract modeling elements are missing, for example navigation, presentation and user interaction.

Our proposed methodology is using similar basis for modeling Web Applications like the previous methodologies but takes into account the data-oriented aspects. High level modeling support is achieved by UML diagrams and implementation level support is achieved by XML technologies. The advantage of our methodology is that several steps can be performed in a semi-automatic way to derive the corresponding code fragments and our UML profile is adapted to the problem domain by means of stereotypes and special attributes. Thus, the methodology is centered on three main aspects of Web applications: the content, the navigation and the presentation. In contrast with the other methodologies our approach does not requires special software tools to support the development process. The utilization of standards makes it possible to perform our approach with any available tool. Consequently, we need to design UML diagrams which are used in a semi-automatic way to generate the prototype of the Web application.

2 Data-oriented Web applications

Naturally, the exact specification of a Web application in the beginning of the development is impossible, because its structure and functionality evolves with time. Moreover, maintenance is an even more significant part of the lifecycle of Web applications than in traditional applications since Web technologies and user requirements change continuously. In addition, non-functional requirements, such as security have to be addressed by a Web development process. The objective is to develop quality Web applications produced through systematic construction of models. In the design of Web applications a useful way is indicated by the Model-View-Controller (MVC) design pattern which is adopted in several frameworks, like Apache's Struts [Struts, 05]. It can improve the application's usability, the creation of reusable code and helps to understand and clarify the functionality of the program. The MVC pattern is very simple, yet incredible useful. Its importance lies in the clear separation of the functional layers and their functionality. However, in data-oriented Web applications, the model component becomes a little simpler because small and medium sized Web applications mostly have simpler business logic. Additionally, in our case the main purpose is the preparation of the corresponding data-structure, which could be used in the subsequent steps. For such systems, the realizations of data input, processing, presentation consumes a considerable amount of production time. Furthermore, in the model construction stage, the primary principle is not the description of the corresponding object's behavior, rather than the effective access and management of the data.

To illustrate the difference, we could imagine a Web application that manages the workflow for online collaborative groups and, another one for managing a person registering system. We need to establish the connection between the web application layer's and the components of the MVC in both situations. The substantial is comes forward when we design the link between the model component of the MVC and the Data layer. In the case of the workflow management system, the Data access layer responsible only for the storage and the service of the required data, the validation part of the model component resides in the Business Logic layer, co-operating with the Controller's functionality. However, in the design of the registering system we could utilize the database management systems supporting functionality (stored procedures, view tables, triggers, etc). The model component's responsibility has to be reduced, the data management functionality has to be transferred to the Data access Layer, and as a result, the Business Logic Layer focused only on the application logic. Moreover, there are further possibilities, like automatic derivation of the relational database schema and the data management web pages from that logical model. Thus, these functionalities make it possible to build systems around the data structure. By considering these functional aspects, we could represent the first aspect of a Web application with UML class diagrams.

Naturally, we need to pay attention also for the design of the navigation. Additionally, a Web application may have more then one entry point in contrast of traditional applications, so the need of well-defined navigation is expected. In the case of data-oriented Web applications, we could achieve more advantage with the derivation of the navigation model from the structural model, because the navigation paths are based on the relations between the entities. Accordingly, the navigational model is represented as a class diagram that shows the objects that could be visited through the navigation. The associations, their multiplicity and role names establish the base for deriving a navigational model. In addition, some extension incorporated for the successful modeling.

The third aspect, the presentation is not influenced by our methodology because the system's model component will answer with an XML document for each request. That XML document could be easily transformed with XML technologies (XSLT style sheets) to the desired output format, achieving universal client access.

3 The methodology

The presently available methodologies are mostly using object-oriented approaches, cleanly separating each component's functionality. The design of Web applications builds on the requirements specification, just like the design of software systems in general. In these systems the conceptual design of the domain is based on use-cases and includes the involved objects that users will perform with the application.

Particular emphasis is placed on the information exchanged between the user and the system. The use case models serve as an input for modeling the content of the application. However, in data-oriented cases it has proved to be an effective strategy to build the object-oriented approach over the data-oriented, i.e. the data structure will be the basis and the application will be built around that structure.

We may have a better understanding of the meaning of data-oriented approach when we think about an existing database that needed to extend with a Web interface. In these cases, the use-cases highly depend from the data model, but the situation is the same when we need to develop new Web applications for managing registration and information systems.

In our approach these use-cases will be used to form the *application tier* in Web applications or if think on the MVC, we formalizing the Controller module with use-cases because this module is responsible to handle user operations. The structural model, a class diagram is used to describe the *data tier*. This diagram also could be used to automatically derive the corresponding relational database schema. The *presentation tier* of a Web application is formed by the Navigation and Presentation model together. The Navigation model is used to describe at each class which related classes can be visited from it. The navigational elements could be realized with menus and links. And the final step is described by the presentation model, how the requested data could be transformed together with the navigation structure to the client's language.

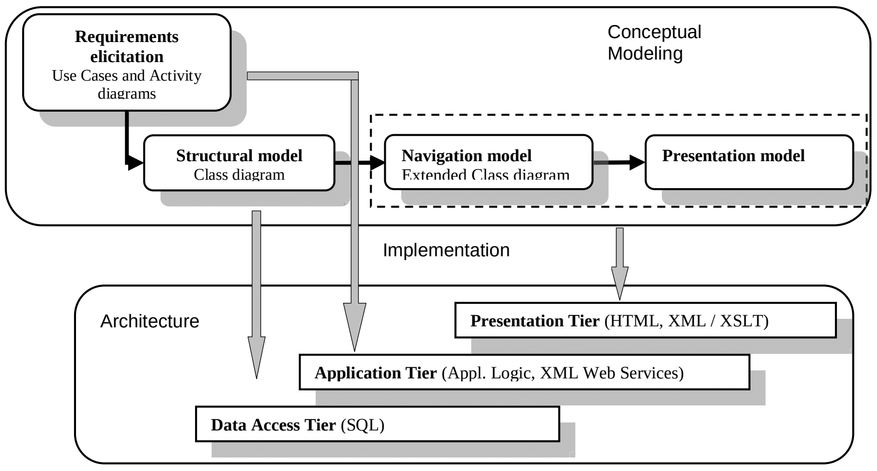

Figure 1: Methodological approach

In the following section our methodology is outlined.

3.1 Context analysis

The first step in the development process – following IBM Rational's software development strategy – is the context- and requirements analysis of the problem domain. Using use-cases and activity diagrams, we could determine the outline of the system, and describe the fundamental functional aspects of the system from the perspectives of different users. The RUP provide a user-centered approach that forces the developers to define who are the actors (users) of the application and offer an intuitive way to represent the functionality that an application has to fulfill for each actor.

As we have data-oriented approach, the generality of the available tasks are related by data management and manipulating activities. The actors are representing the different roles for users of the Web application. Use-cases are used to describe available operations, and have to be detailed by activity diagrams. These activities will produces the entry points of the navigation diagram for each user role and the use-cases will serve the layout for the opening page for each actor (containing the list of available tasks/entry points). The activity diagrams are used to describe business logic and furthermore, could be used to derive program modules and code skeletons. Of course, we have a simpler case when we need to develop a system for an existing database. In this situation, the structural model is deeply bounded for the data structure, and we need to focus on the performable data management tasks.

Additionally, further important factors are exists, as performance or availability. These aspects could influence our modeling method, and further researches will consider these factors. However, at this time we are focusing on platform independent models leaving untouched state models and page handling scenarios (sequence diagrams).

3.2 Structural model

The conceptual design aims to build a domain model trying to take into account as little as possible the navigation paths and presentation aspects. The main modeling elements used in the conceptual model are: class, association and package. For a common Web application the use-cases and the information-flow descriptive activity diagrams could be the base of the conceptual design of the domain.

We could use an incremental approach to identify classes. First, we need to identify the "active" entities in the system. At first glance, the actors identified in the use-case appear to be prime candidates for being listed as potential classes. Next, we need to identify the business domain ("passive") entities in the system. Usually, these business domain classes are mapped to either one or more database tables.

However, in data-oriented cases the base of the conceptual model should be the managed data, not the typical user activities and related use cases, as we could see in the earlier example. So the modeling sequence will be modified a little. At first, the information carrier classes and attributes are determined, and only after this point will be detailed the use-cases describing the user activities. The result of the first step will be a class diagram which will determine the structure of the system, the classes and their relations (like association, aggregation, …). In this stage we need to ensure only the data management activities to the users, so the tasks list will be shorten. The most obvious solution to preserve consistency is to leave the data management tasks to the database management system.

However, we could find that the data layer is responsible only for serving the data and the Business logic layer is responsible for the validation, some would agree our approach. The idea behind this modification comes from the nature of the area being modeled, namely data-oriented directions. Moreover, most of the Web based systems still using relational databases to store information about a domain. These facts confirm the point that a data-oriented approaches still useful to develop Web applications in an object-oriented world.

3.2.1 Division of the Business Logic Layer

To achieve simpler data management seems to be a logical solution to separate the Business Logic Layer into two distinct parts. To strengthen this statement we could think about the function of the primary and foreign keys, or the "not null" constraints which are associated closely with the data management functionality. One part of the Business Logic will be responsible for the data management, while the other part will manage the application itself. This separation will result the expansion of the data layer with data management tasks. After this separation the application logic associated only the workflow management (user authorization, session management, …).

As a consequence, the detachment of these data management tasks would include new functionalities into the class diagram. UML offers two modeling possibility to handle this situation: qualified associations (e.g. foreign key management) and the attachment of constraints (which could be a formula, an expression of a programming language or a simple text).

To utilize this approach successfully, we need to design our Web application using relational basis. We could still use many-to-many associations, but we should attach an association class these relations. This limitation makes it possible the automatic code generation for the relational database schema. The usage of directed associations to describe container-contained relationship is not required in this stage, but in the navigational model highly advised to use directed associations for indicating the navigation directions.

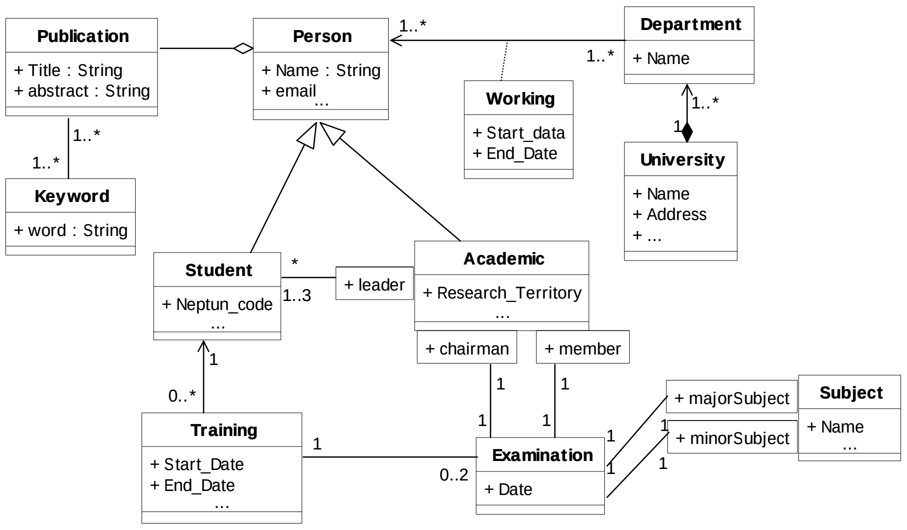

Figure 2: A part of a University's Structural model

3.3 Navigation model

The next stage in the development process is the navigation model design. The navigation model specifies which objects can be visited in a Web application. Moreover, it defines the availability of the objects. In the navigation model's building process the developer takes crucial design decisions, such as which view of he conceptual model is needed and what navigation path are required to ensure the application's functionality. The decisions are based on the conceptual model, use-case model and navigation requirements that the application must satisfy.

The navigation diagram takes a structural diagram copy as its starting-point, which could be extended with additional associations. Generally, these new associations should be added for direct navigation to avoid navigation path of length greater than one. However, there could be some conceptual classes that are not a visiting target in the use-case model. It is irrelevant in the navigation model and therefore it is omitted in the navigation diagram. The navigation inside the Web application occurs along the associations, which are used to describe the relation between navigation classes. These associations will appear as hyperlinks in the user interface.

3.3.1 Navigational structural model

Conceivable, that navigation model is not detailed enough to function as a basis of the navigation for Web applications. Think of that resolution of relations, or the website's menu structure.

We could find useful guidelines and graphical notations introduced by UWE, like

- index,
- query,
- and menu.

We could design proper models if we extend our navigation model with these notations, and this new model could serve as a background of the automatically generated navigation system. Unlike UWE, we do not use new graphical notations in the diagrams; instead we pick up new attributes extended with stereotypes to describe their functionality in the navigation. The approach could be seen in figure 3, corresponding for the structural model in figure 2.

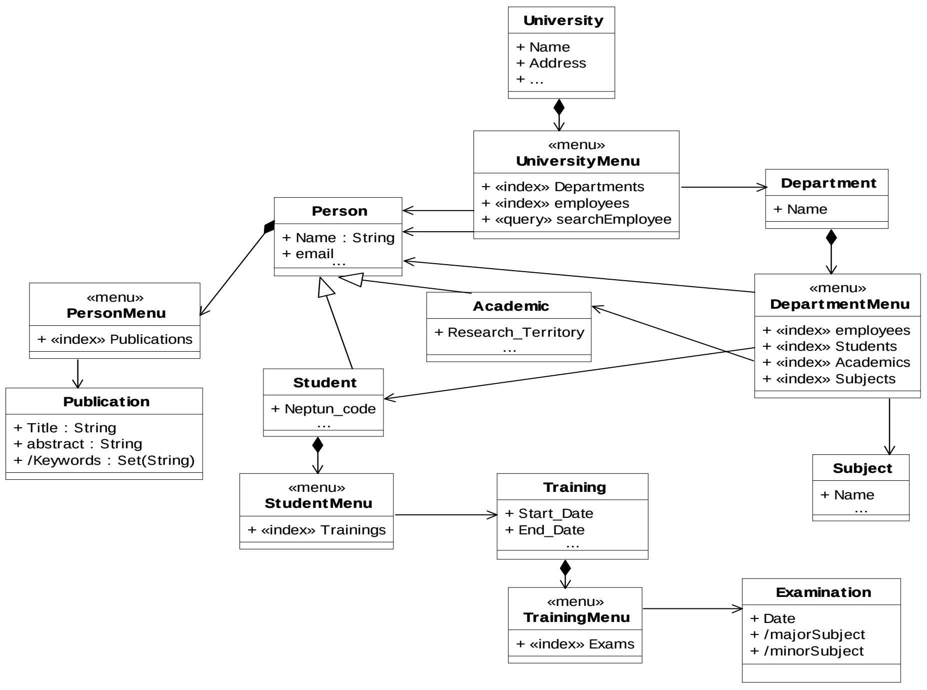

Figure 3: Navigation structural model

3.4 Presentation model

Our methodology does not deal with presentation aspects because the answer for each request is presented with an XML document containing the result and the navigation structure starting from this entity. This XML document could be transformed to the desired output format using XSLT. The structure of the opening page for each user also could be derived from to related use cases.

4 Code generation

These models help to comprehend the problem domain, but these models would be offer a more complex support if we could generate from them a working prototype of the expected Web application. Naturally we are agree that UML models supposed to be abstract, but it is not uncommon for a UML model to capture almost as much technical information. So, it is not a big deal if we extend our models with some implementation specific information. Note that many mappings are possible between an XML Schema [Schema, 00] and a UML model (think about associations and attributes). Accordingly we need to introduce some limitations to aid code generation (e.g. we need to create an association class for every many-to-many relation and there are no multiple inheritances).

Therefore, we could imagine an XML Schema as a specific rendering of a very detailed data model. To achieve this generation process XML technologies offers effective assistance with the support of the above discussed design strategies adjunct with modularity and reuse. This code generation process is performable with the help of the XMI (XML Metadata Interchange) [XMI, 01] interface worked out by OMG. XMI is an industry standard, supported by all the major modeling tools. So there is no problem in a collaborative environment if the teams are using different tools.

The first step in our generation process is the derivation of XML files from UML diagrams conforming for the XMI format as we could see in Fig. 4. The example shows the XMI representation of the `Person` class. This generated XMI file could be transformed into the XML Schema format serving as a base of the web application. Naturally, there is the possibility of the reverse procedure: from XML Schema to XMI.

```
<UML:Class xmi.id = 'Ia1efm1' name = 'Person' visibility = 'public'>
  <UML:Classifier.feature>
     <UML:Attribute           xmi.id='Ia1efm2'            name='Name'
visibility='private'>
        <UML:StructuralFeature.type>
           <UML:Class xmi.idref = 'Ia1efm3'/>
        </UML:StructuralFeature.type>
     </UML:Attribute>
    <UML:Attribute           xmi.id='Ia1efm6'            name='email'
visibility='private'>
       <UML:StructuralFeature.type>
          <UML:Class xmi.idref = 'Ia1efm10'/>
       </UML:StructuralFeature.type>
    </UML:Attribute>
  </UML:Classifier.feature>
</UML:Class>
```

Figure 4: Part of the generated XMI file

Using XSLT transformations [XSLT, 99], we could derive the appropriate XML Schema file. In figure 5, we could see a part of a schema illustrating the mapping of a UML class into an XML Schema element.

```
<xs:element name="Person">
   <xs:complexType>
          <xs:sequence>
                 <xs:element name="name"  type="xs:string" />
                 <xs:element name="email" type="xs:string" />
          </xs:sequence>
   </xs:complexType>
```

Figure 5: The corresponding XML Schema part

From subsequent transformations we could derive the database schema for the database management system and moreover the XForms [XForms, 03] based web pages used to manipulate the data. The XForms standard allows us to define forms that are much more sophisticated than those of HTML. Perhaps more importantly, it makes it easier for web applications to grab and use the data entered into forms, because an XForms client can plug the data directly into any XML structure.

```
<xforms:model id="mdlPerson" xmlns:xxi="http://4xmpl.org"
 schema="./Conceptual.xsd">
   <xforms:instance id="dataPerson">
          <Person-container>
                 <Person>
                        <name></name>
                        <email></email>
                 </Person>
          </Person-container>
   </xforms:instance>
   <xforms:submission id="s1" method="multipart-port"
action="http://localhost/cgi-bin/test.cgi" indent="true" />
</xforms:model>
…
<xforms:repeat id="r1" nodeset="/Person-container/Person">
<table width="700" border="0" align="center">
   <tr>
          <td><xforms:input id="inpName" ref="name">
                 <xforms:label>name</xforms:label>
          </xforms:input></td>
          <td><xforms:input id="inpEmail" ref="email">
                 <xforms:label>email</xforms:label>
                 <xforms:hint>name@domain</xforms:hint>
          </xforms:input></td>
  </tr></table>
</xforms:repeat>
```

Figure 6: An XForms based data manipulation page

The entered data stored in a local XML document and validation supported by XML Schema files that derived from the UML models. Its main advantage is the separation of the presentation model from the data model. The data model part of XForms enables you to declare data items and structure separated from any set of widgets used to display the data values. XForms defines the means for binding these data items to the display widgets separately from the declaration of the data model itself. In addition, declarative means of acting on changes in value for any particular data item are also defined. XForms widgets are abstract in nature, so different platforms can choose to implement them in different ways.

To achieve this automatic derivation of these XFroms based web pages we need to extend the logical model with further additions, like implementation stereotypes and data types used by the XForms standard. For example, we could implement the secret input filed type as a restriction for the built-in string type or we could define the syntax of the email address restricting the string type too. At this point we could use these new types in our models and the generated XForms based pages will not display only regular input fields. Other possibilities in the XForms standard are the `hint` and `label` tags that are used to make the user interface more user friendly showing labels and helpful hints for the input fields. This features are also could be integrated into the most detailed view of the UML models.

These transformations and extensions are makes possible to generate a working prototype of the Web application from models. The generation process could be seen in figure 7.

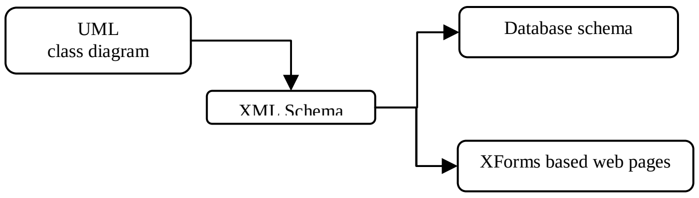

Figure 7: Data-oriented design strategy with UML and XML

4.1 Further extensions for the logical model

The above discussed models concentrate mainly for the data structure of a Web application, while the problem domain usually contains further constraints which could not be modeled only by classes and associations. For example, using a Personal Registration Web application we could have complex restrictions, like a person should not be a member of a committee. In a UML class diagram this relation appears in an association between the two classes. To express this restriction we could attach constraints to this association written in a textual form or using a programming language syntax.

In the implementation phase, these restrictions generally appear as constraints between components of an XML document which are known as co-occurrence constraints and are not expressible in the XML Schema language. To extend the descriptive power of our methodology we could combine the XML Schema language with the Schematron language [Schematron, 02]. Schematron is a rule based schema language that uses path-expressions instead of grammars. This means that instead of creating a grammar for an XML document a Schematron schema will make assertions applied to a specific context within the document. If the assertion fails, a diagnostic message that is supplied by the author of the schema can be displayed. Thus, the validation process consists of two steps; first, the XML document is checked against the Schematron rules using XSLT processor.

As a function of the validation result, an error message would be displayed or the normal validation continues against the XML Schema. However, Schematron is not the only possibility to describe semantic constraints. On the UML diagrams we could use directly the database management's system built-in language. We could utilize it in attached constraints or in tagged values. In the implementation, these constraints will be transformed into check constraints or stored procedures using e.g. the Pl/SQL [Oracle, 02] language.

4.2 Further possibilities

Related XML technologies offer several connecting possibilities. For example, we could generate diagrams from data utilizing the Scalable Vector Graphics (SVG) [SVG, 02] format or we could support a unified availability of the data with web services or we could generate RSS news feed.

5 Conclusion and future work

In this paper we have illustrated the complexity of data-oriented Web applications and that it is not at all a systematic task. We have introduced a new methodology to help in the development of effective Web applications based on UML and XML technologies to support data management tasks in small and medium sized projects. We have proposed some remarks in the implementation phase utilizing XML technologies to develop modular, scalable and expandable Web-based systems.

The method is based on a UML profile adapted to the problem domain by means of stereotypes as well as a strategy for generating code templates from such models. We provide a method to derive the navigation model from the structural model of a Web application. An advantage of the proposed methodology is that several steps can be performed is a semi-automatic way providing rapid development and prototyping.

Ongoing researches can go in several interesting research directions in the design and development phase. We are going to study the additional expandability of our UML based methodology.

References

[Baresi, 01] L. Baresi, F. Garzotto, and P. Paolini: Extending UML for modeling web applications. In Proc. of 34th Annual Hawaii International Conference on System Sciences (HICSS'34), Maui, Hawaii, Jan. 2001. IEEE Press.

[Booch, 99] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide, Addision Wesley, 1999.

[Ceri, 00] Ceri, S., Faternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites, Proc. WWW9, 2000.

[Conallen, 03] Conallen, J.: Building Web Applications with UML 2nd Edition, Addison Wesley, 2003.

[Gingeri, 03] Ginegi A., Murgesan S.: The Essence of Web Engineering, IEEE Multimedia, Vol. 8, no. 3

[Hennicker, 00] Hennicker R. and Koch N.: A UML-based Methodology for Hypermedia Design., UML´2000, LNCS 1939, Springer Verlag, 2000. 410-424

[Oracle, 02] Oracle 9i Database, 2002, http://www.oracle.com/

[Raman, 04] Raman, T. V.: XForms, XML Powered Web Forms. Addison-Wesley, 2004.

[Schattkowsky, 02] T. Schattkowsky and M. Lohmann: Rapid development of modular dynamic web sites using UML. In J.-M. J´ez´equel, H. Hussmann, and S. Cook, editors, In Proc. of 5th International Conference on UML 2002 - The Unified Modeling Language, pages 336–350. Springer, LNCS 2640, Oct. 2002.

[Schema, 00] XML Schema Part 0: Primer, 2000, http://www.w3c.org

[Schema, 00] XML Schema Part 1: Structures, 2000, http://www.w3c.org

[Schema, 00] XML Schema Part 2: Datatypes, 2000, http://www.w3c.org

[Schematron, 02] Robertsson, E., Combining Schematron with other XML Schema languages, 2002, http://www.ascc.net/xml/resource/schematron/schematron.html

[Silberschatz, 97] A. Silberschatz, H. F. Korth, S. Sudarshan, ``Database System Concepts (3rd Edition)'', *McGraw-Hill Co.*, 1997

[Struts, 05] Sruts, The Apache Software Foundation, 2005, http://struts.apache.org

[SVG, 02] Scalable Vector Graphics, Version 1.2, 2002, http://www.w3c.org

[UML, 01] The: UML Specification, version 1.4, 2001, Object Management Group

[W3C, 02] W3C – World Wide Web Consortium, http://www.w3.org/

[WSDM, 00] O.M.F. De Troyer, C.J. Leune: WSDM: a user centered design method for Web sites, 2000.

[Xforms, 03] XForms Part 0: Primer, 2003, http://www.w3c.org

[XSLT, 99] XML Transformations (XSLT) Version 1.0, 1999, http://www.w3c.org

[XMI, 01] The XML Metadata Interchange (XMI), 2001, Object Management Group

[XML, 98] Extensible Markup Language (XML) 1.0, 1998, http://www.w3c.org

[Zhao, 02] Zhao, W., Kearney, D. and Gioiosa G.: Architectures for Web Based Applications, AWSA, 2002.