Web-for-Web:
A Tool for Evolving Data-Driven Web Applications[*].

Olga De Troyer, Jo De Greef, Peter Stuer
Vrije Universiteit Brussel
WISE Research group, Department of Computer Science
Pleinlaan 2
B-1050 Brussel, Belgium
Olga.DeTroyer@vub.ac.be, Jo.De.Greef@vub.ac.be, pstuer@vub.ac.be

Abstract
The Web enables us to place large collections of information, stored in databases all over the world, at the disposal of people all over the world. In principle, it is not difficult to publish the information from a database on the Web because the technology exists. However, it may be a time consuming effort to define all the web pages needed for consulting and possible also for updating the information. If in addition the underlying database tends to evolve in time, web pages have to be defined over and over again. In this paper we describe a tool, Web-for-Web, that allows to build, in an easy and fast way, web applications for consulting and maintaining information stored in a database and in addition allows to readily adapt those web applications to changes to the database structure. To realize this, the tool makes use of Meta data, XML and XSL.

1 Introduction
Next to using the Web for marketing purposes or e-commerce, the Web can also be used to open innumerable collections of information to people all over the world. The availability of a compiled collection of information about a certain domain on the Web would significantly increase visibility and could advance research in this domain considerably. The TensiNet project [TensiNet, 2001] is such an initiative. TensiNet is a European Thematic Network funded by the EC within the 5th Framework Program; the work program is GROWTH (Promoting Competitive and Sustainable Growth) [EC-GROWTH, 2002]. The purpose of the project is to advance the use of Tensile Structures in architecture and urban planning. There was a need for this because the Tensile Structure industry had a number of problems in Europe. There was insufficient transfer of multi-disciplinary know-how between architects, engineers, the construction industry and research institutes; basic knowledge about Tensile Structures was not widespread and common reference data was unavailable; there were no uniform guidelines for design and construction of these structures in Europe; and there were a number of unanswered scientific questions about the material and the structural behavior of Tensile Structures.

Therefore, the purpose of TensiNet was to assemble, structure and analyze existing data and integrate this expertise in a knowledge base accessible to all partners of the project and later on also to a broader community. To disseminate the information in the knowledge base and to facilitate the collection of information a web interface would be provided. Already before the

[*] This work was supported by the European Commission in the context of "TensiNet" a Thematic Network in the 5th Framework Programme (GROWTH)

start of the project, it was clear that the structure of the database underlying the knowledge base could not be defined in a single step. A lot of knowledge was based on intuition and the needed information was only available in the heads of people. It would be a long and difficult elicitation process. As a consequence, it was decided to build the database for maintaining the information in an evolutionary way, meaning that the final version would evolve from a limited first version by expansion and adjustment in a number of iterations. This approach was needed to support the flexibility needed for the development of this knowledge base.

Because the structure of the database would evolve over time, it was clear that also the web interface of this database had to evolve. Using a classical web-programming environment (such as e.g. Microsoft Visual InterDev 6.0 [InterDev, 2002], Macromedia Dreamweaver Ultradev 4 [UltraDev, 2002]) would imply that each time the database structure changes or new tables are added, the code for the web interface has to be changed as well or extended by hand. To avoid this, we developed the Web-for-Web tool. This tool allows to describe the database and its web interface at a Meta level and to generate the web pages at run time from this description. In this way it is not only easy and very fast to develop the web application but also very easy to make changes to the database or the web interface: only some changes at the Meta level are needed and an updated web interface will be generated at run time. Web-for-Web dramatically reduces the time and effort required to develop data-driven web applications and allows to readily adapting them to real world needs. To realize this, Web-for-Web uses database technology in combination with XML and XSL. The tool is bootstrapped, which means that the application for maintaining the Meta level is also a web application, described and generated by the tool itself, hence the name “Web-for-Web”. The tool is not limited to the TensiNet project. It is already used for other projects as well.

The rest of this paper is structured as follows: In section 2 we give a general description of the tool and describe its architecture. In section 3 we give more details about the Meta Database underlying Web-for-Web. In sections 4, we describe the role of XML and XSL. The tool itself is illustrated in section 5 by means of two applications. Finally, section 6 draws conclusions and discusses further work.

2 Overview and Architecture of the Web-for-Web Tool

As already explained in the introduction, Web-for-Web allows building a web interface for a database in an easy and flexible way. Web-for-Web actually has two types of users. First there is the developer who uses Web-for-Web to specify the web interface needed for the database. In the rest of this paper we will call this type of user: *the designer*. On the other hand, we have the visitor or user of the web application who uses Web-for-Web indirectly because the pages he is viewing are generated by Web-for-Web. We will refer to this type of user as *visitor*.

Web-for-Web uses an object oriented approach for displaying and manipulation the information in the database. This means that the information is viewed as a collection of objects described by and grouped into object classes. E.g. a possible object class for the TensiNet web application would be “Project”. Attributes for Project could be: Name, Location, Owner, Year-of-construction, Constructors-involved, …. Using Web-for-Web, the visitor of the web site can (in principle) query, view, edit and delete instances of such an object class. New instances can be created for an object class and an overview list of instances of an object class can be obtained. Note however that the access to object instances and object classes can be restricted by access

rights (explained later on). If a visitor of the website has unlimited access, a possible way to start off is with a web page showing the list of all object classes (see figure 1). The object classes are given by means of their name and with each object class an Edit-button is associated. Clicking on the name of such an object class gives a page with a list of all instances of that class. (Clicking on the Edit button associated with such an object class will allow to add, delete or edit instances of that object class.) Clicking on one of the object instances in the list of instances will give a page with the values of the attributes of the selected object instance. Because an object class may have many different attributes, the designer can opt to group attributes together. E.g. Street, House-Number, ZIP-code, City and Country can be grouped as "Address". This allows displaying the information associated with an object instance in a more structured way (see figure 2).

Figure 1: List of all Object Classes in the TensiNet website

Figure 2: Information about an object instance in the TensiNet Website

If the visitor has the rights to update information, appropriated buttons (for adding, deleting or editing) will be available next to the information (which can be an attribute, an object instance or an object class) and appropriated input pages will be generated (see figure 3 for such an input screen).

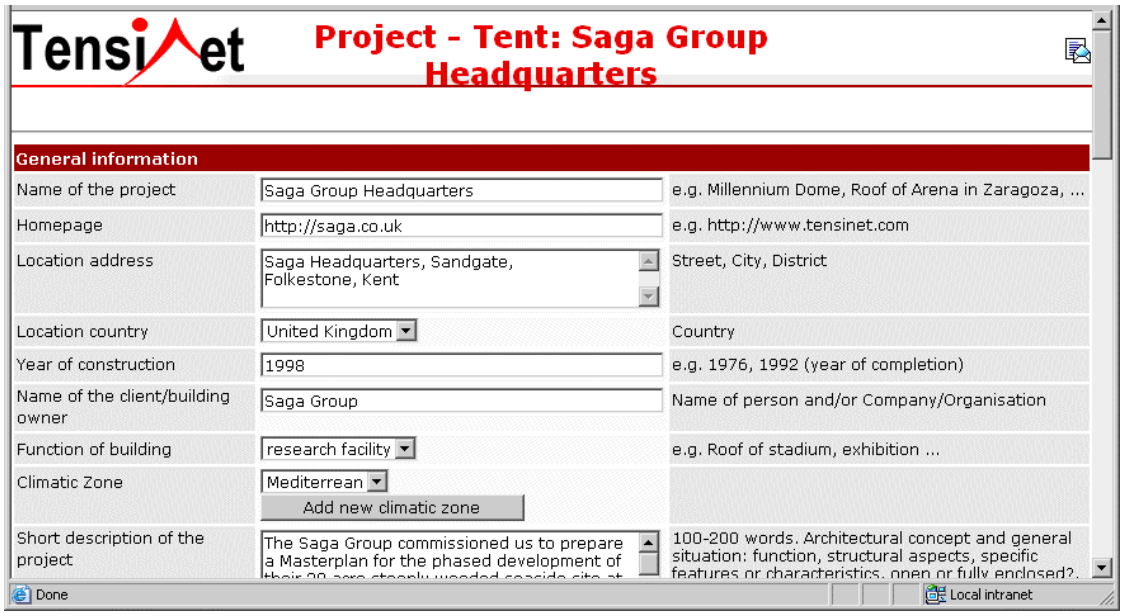

Figure 3: Page of the TensiNet website to update a Project object instance

Next to the possibility to group attributes, the designers can also associate captions, hints and help texts to attributes. Hints are displayed next to the attribute and can be used to give examples of expected input or a short description of the meaning of the attribute to the visitor (see figure 3). Help text will be displayed if the visitor uses the help button associated with the attribute. The designer can also define the order of displaying the attributes and indicate whether some attributes have to be omitted. All this together defines a “view” on an object class. The designer can define different views for a single object class. In this way it is possible to define a view for viewing an instance of the object class, a view for updating an instance of this class and even a view for printing the instance (as printing on paper may require a different format than viewing information on a screen).

To protect the information from unauthorized use, the designer can associate access rights with the information at different levels of granularity. Access rights can be associated to object classes, object instances and individual attributes. Different access rights can be defined for different groups of users.

Figure 4 gives an overview of the architecture of Web-for-Web. Web-for-Web uses a three-tier structure*: a data layer*, *a logic layer* and *a presentation layer*. The information to be viewed and manipulated through the web application is stored in a (relational) database. The Meta information (information on how the data is structured in the database and how it must be structured on the web-interface) is also maintained in a database, called the Meta-Database (see section 3 for a discussion on the Meta Database). Currently Microsoft SQL Server [MsSQL, 2002] is used for this data layer, but in principle any (relational) DBMS could be used.

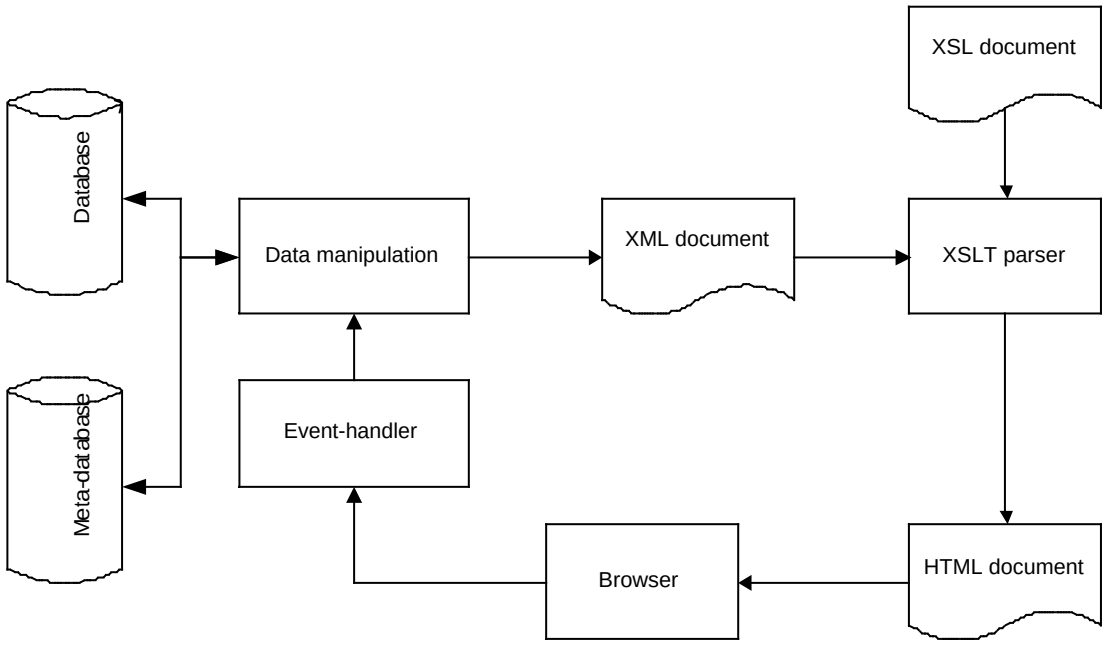

Figure 4: Architecture of Web-for-Web

For the logic layer, a server side scripting language (PHP [PHP, 2001]) is used. If the visitor of the web application requests a page, the Event Handler will pass the request to the Data Manipulation Module, which will extract the requested data from the Database using the Meta information in the Meta Database. Instead of directly generating HTML code, the output of the Data Manipulation Module is an XML document ([XML, 1997]) only containing the requested data and its structure. In this way, presentation issues are separated from structure and logic. How to display the information in the XML document on the web page (the presentation layer) is described in an XSL document ([XSL, 2002]). Next, an XLST parser ([XSLT, 1999]) will generate HTML for the XML document using the XSL document. This approach also has the advantage that different XSL documents (i.e. different style formats) can be used for the same XML document. Section 4 will discuss into more detail the role of XML and XSL in this tool.

For updating information in the database, the Event Handler passes the request together with the information to the Data Manipulation Module, which will update the Database accordingly using the Meta information in the Meta Database.

3 The Meta Database

In this section we will describe the structure and role of the Meta Database.
The Meta Database contains information about the data (in the database) needed to generate the web interface and to update the database. The Meta Database actually contains three types of Meta information:

- Information describing the structure of the data in the database;
- Information on how the data needs to be structured on the web pages;
- Information on how the data can be manipulated through the web interface.

Note that for the moment, the Meta database does not contain information on how the data should be displayed on the web pages. As already explained this is captured by means of XSL documents. In section 4 we explain why we have opted for this solution.

As explained in the previous section, we take an object-oriented approach to view and manipulate the data in the Database. Therefore, the Meta Database describes the information in the database from an object-oriented (OO) and conceptual point of view. The central concept in the Meta Database is the *object class*. For each object class, the *attributes* are described and possible *subclasses* and *parent classes* are given. We distinguish between *value attributes* (e.g. number, string, text, URL) and *object attributes*, which in fact model relations between object classes. Two examples of object classes for the TensiNet website are: Company and Project. Some of the attributes for Project are: name, homepage, country of location, year of construction, building owner. Name, homepage, country of location, and year of construction are value attributes; building owner is an object attribute because the building owner of a project is a company (which is represented as an object instance). Two of the subclasses for the object class Company are Architect's Firm and Engineering Firm. Company is called the parent class of Architect's Firm and Engineering Firm.

Next, this conceptual OO description of the type of data stored in the database is linked to the actual structure of the database (multiple databases are possible). We suppose that the database is a relational database because this type of database is the most popular one. In a relational database, object classes are stored into tables, and the attributes correspond to columns in the tables. This means that in the Meta Database, we capture for each object class, the name of the corresponding table in corresponding database; and for each attribute, the name of the corresponding column (s). For object attributes, also information is captured to be able to locate the associated objects and their attributes in the database (i.e. the foreign key).

Next to this type of Meta information (describing the type of data and how it is structured in the database), the Meta Database also contains information on how the data needs to be structured on the web interface. As already explained, the web interface is object-oriented as well. Therefore, the description how the data must be structured on the web pages is also associated with the object classes. This means that with each attribute of an object class the following information can be associated: a caption, a hint, a help text, and an order number. An example caption for the attribute "building owner" of the object class Project is "Name of the client/building owner" (see figure 3 – on this figure also examples of hints are shown). The order number is used to indicate the order in which the attributes need to be displayed. Furthermore, the attributes of an object class can be grouped into *categories* and these categories can be given names and also an order (see figure 2 for the use of categories – two categories are shown: "Function" and "Material of the cover").

To allow to manipulate the data in the database, the Meta Database also contains information on who is allowed to do what. This type of information is expressed in the form of access right on groups of users. Again the access rights are not expressed at the relational level but at the OO level. Access right can be defined for each object class, for individual attributes of an object class, and even on the level of the individual object instances. This information is sufficient to generate (later on) the appropriate pages. E.g. if a visitor has no update rights, no update buttons or input pages will be generated (he only will be able to query the information); if a visitor only has edit right but no delete or add right, only an edit button will be generated.

The fact that all information in the Meta Database is linked to the conceptual level (OO) has the advantage that most of the tool is independent on how data is maintained and stored in a relational database. E.g. suppose we want to adapt the tool in such a way that it can also build web interfaces for data stored in an object-oriented database then we only have to adapt the part of the Meta Database that describes how the object classes are mapped to the database structure.

4 The Role of XML and XSL

Because it was important that the tool could be used for more than one project, it was needed to allow for different styles of presentations. Even within one project, it is possible that different styles are needed for the same data. E.g. suppose you want to use different colors and fonts for different types of users. We could have solved the problem by also storing information about presentation styles into the Meta Database and let our PHP-application directly generate HTML. However, this would have complicated the application. In addition, there was no direct reason for doing this because this functionality was already available in the XSLT parsers. Thus, we limited the functionality of the application to extracting and structuring the information needed on a particular web page. To capture the extracted information and its structure it was clear that XML was most appropriate. Therefore a DTD was defined. Again the structure defined by the DTD is an object-oriented one (central element in the DTD is an object instance) and in addition independent of the project under consideration.

Next, a set of XSL documents is used to express the particular presentation style needed. In principle only one set of XSL documents needs to be defined for a project. A set of XSL documents per project is used to obtain a maximum of flexibility. E.g. common elements such a the header of the pages are defined in one XSL document, another XSL document is used for define input pages, and still another XSL document for defining output pages. An existing XSLT parser is used to generate HTML code. In principle it is also possible to store the information contained in the XSL documents in the Meta Database (and thus also maintain presentation information in the Meta Database). This would not change the functionality of the system, but it would simplify the definition and maintenance of the presentation styles for people not acquainted with XSL. Because of lack of time this has not yet been done.

Using XML and XSL has considerably simplified our application and as a consequence shortened the implementation time. In addition, the tool is more flexible because presentation styles can be changed without having to modify anything; only another XSL document is needed.

5 Example Applications

In this section we will illustrate the use of the tool with two examples. The first one is the TensiNet web application described in the introduction. Figures 1, 2 and 3 already illustrated parts of the Web interface generated by Web-for-Web for this application. Figure 5 shows the Web Interface for maintaining the Meta Database. The complete website of TensiNet (generated with Web-for-Web) can be found on http://www.TensiNet.com. However notice that currently most of the website is only accessible by the members of the project and therefore is protected by means of a password.

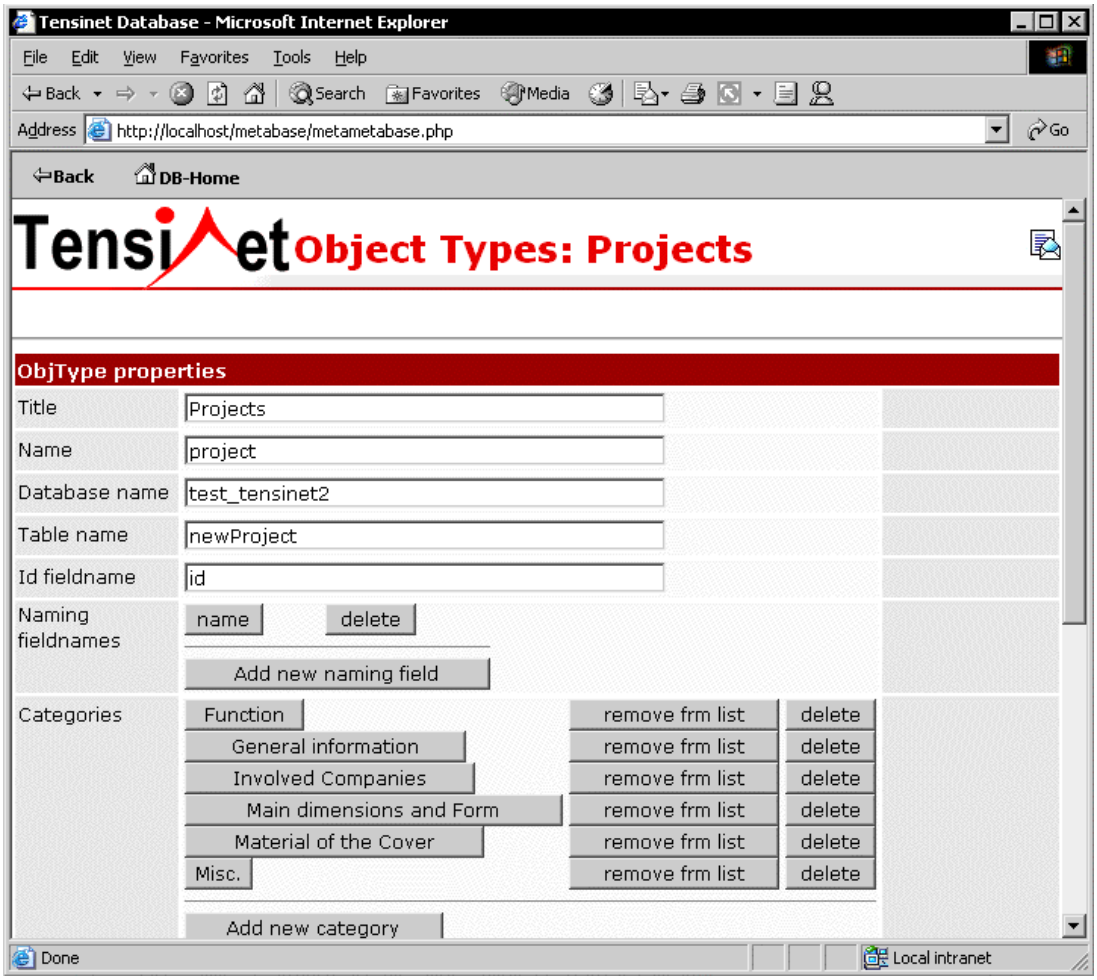

Figure 5: Page for updating the Meta Database for the TensiNet project

The second application is the BEOND project. In this project a web application is developed to allow lecturers to publish information about their courses on the web in a structured way and to allow students to consult this information and submit deliverables for projects and courses electronically. In this case, the structure of the database used for maintaining the information is less sensitive to changes than in the TensiNet project. However, this is not a reason for not using the tool. The main issue in this project was the language. Due to the fact that our university has a large number of foreign students, it must be possible to either generate an English version or a Dutch version (the local language) of a course web site. See figure 6 for an example page.

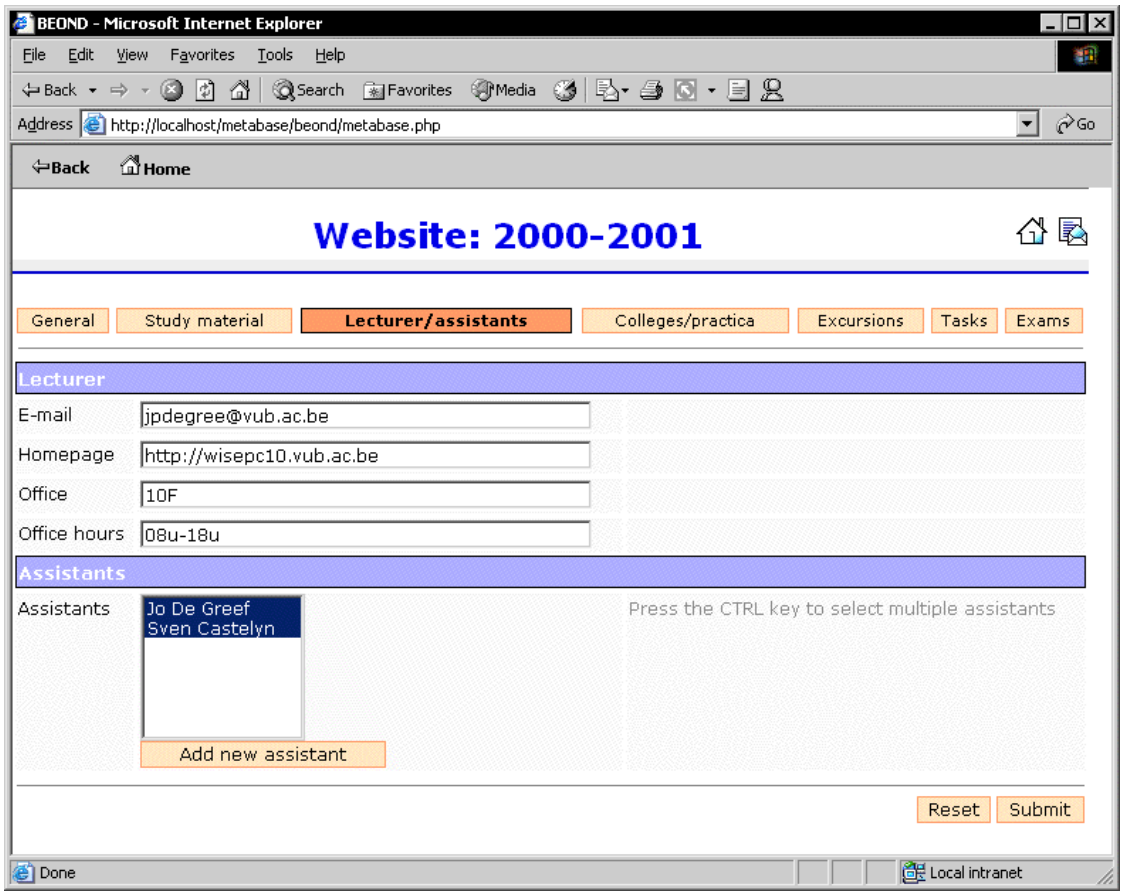

Figure 6: Page of the BEOND website

6 Conclusions and Future Research

Publishing information on the Web is said to be easy. However if lots of different types of information need to be published and maintained through the Web, many different web pages need to be developed. Although this is not a difficult task it may be a time consuming one. If in addition the database used to maintain the information is evolving very much, the web pages need to be adapted and new pages need to be added constantly. To reduce the time needed to develop and maintain such data driven web applications, the Web-for-Web tool was developed. The tool exploited the fact that both the way the data is structured in the database and the way the data should be structured and manipulated on a web page can be described at a high level in a so-called Meta Database. If information from the database needs to be displayed on a web page it can be extracted from the database using the information in the Meta Database. The information together with information how to structure it on the web page, is returned as an XML document. Next an XSLT parser will generate HTML code for the page using an appropriate XSL document. If the structure of the underlying database changes, it is only needed to specify the changes in the Meta Database. No additional coding is needed. The implication of the approach on performance is small. Most time is lost in checking the authorization rights. This is due to the fine level of granularity that we allow for specifying access rights. However, the performance is still very acceptable.

The Meta Database can be filled using a web interface, of course generated by the tool itself (bootstrapping). In the current version of the tool, the database maintaining the data still needs to be created in a classical manner (e.g. using SQL create table statements). However, it is possible to generate the database from the description given in the Meta Database. Also updates to the structure of the database can be derived from updates made to the Meta Database. Experience in generating a relational database from a conceptual description is available in the group (see e.g. [De Troyer, 1986], [De Troyer, 1989], and [De Troyer, 1993]) and we plan to do this in the future. On the other hand, if the database already exists it is, in principle, possible to generate parts of the Meta Information from the description available in the DBMS's repository. However, some reservation is needed here. Currently, the tool works under the assumption that the database is built according to a number of rules. In a next version of the tool, we plan to eliminate those restrictions.

References

[De Troyer, 1986] De Troyer O., Meersman R., Transforming Conceptual Schema Semantics to Relational Data Base Applications. In: Information Modeling and Data Base Management, ed. H. Kangassalo, Springer Verlag 1986.

[De Troyer, 1989] De Troyer O., RIDL*: A tool for the Computer-Assisted-Engineering of Large Databases in the Presence of Integrity Constraints. In: Proceedings of the ACM-SIGMOD "International Conference on Management of Data", eds. J. Clifford, B. Lindsay, D. Maier, ACM Press, 1989 (pp. 418-430).

[De Troyer, 1993] De Troyer O., On Data Schema Transformations. ISBN 90-9005913-X. Ph.D. Thesis, 1993, pp.1-301.

[EC-GROWTH, 2002] CORDIS (2002), EC- 5th Framework Programme (GROWTH) http://www.cordis.lu/growth/

[InterDev, 2002] Microsoft Coporation (2002), "Microsoft Visual InterDev", http://msdn.microsoft.com/vinterdev/default.asp

[MsSQL, 2002] Microsoft Coporation (2002), "Microsoft SQL Server", http://www.microsoft.com/sql/default.asp

[PHP, 2001] The PHP Group (2001), "PHP: Hypertext Preprocessor", http://www.php.net/

[TensiNet, 2001] TensiNet (2001), "TensiNet", http://www.TensiNet.com/

[UltraDev, 2002] Macromedia, "Macromedia Dreamweaver UltraDev 4", http://www.macromedia.com/software/ultradev/

[XML, 1997] W3C (1997), " Extensible Markup Language (XML)", http://www.w3.org/XML/

[XSL, 2002] W3C (2002), "The Extensible Stylesheet Language (XSL)", http://www.w3.org/Style/XSL/

[XSLT, 1999] W3C (1999), "XSL Transformations (XSLT)", http://www.w3.org/TR/xslt