**Vrije Universiteit Brussel**
**Faculty of Science**
**Department of Computer Sciences**


# An Audience-Driven approach to Web-for-Web

by
Michael Mattan
Academic Year
2002-2003

Essay brought forward to achieve the degree of Licentiate in the Applied Computer Sciences

Promotor: Prof. Dr. Olga De Troyer

# Abstract

Nowadays a lot of websites offer a huge amount of information, which is not always presented in a suitable way towards the audience of the website. WSDM offers a feasible solution for this problem because it offers us the opportunity to identify the different Audience Classes of the website and separate the information of the website according to these Audience Classes.

In this thesis we will give the design of a tool that gives the opportunity to the designer to develop an Audience-Driven web application. This tool will be built as a layer around Web-for-Web so that we can reuse its functionality concerning the well-structured storage of the website's information in the database. Finally, after the design is given, a Case Study will be discussed, which will proof the correctness of the design and the implementation of the tool.

**Vrije Universiteit Brussel**
**Faculteit Wetenschappen**
**Departement Informatica**


# Een Doelgroepgerichte Benadering van Web-for-Web

door
Michael Mattan
Academiejaar
2002-2003

Verhandeling voorgedragen tot het behalen van de graad van Licentiaat in de Toegepaste Informatica

Promotor: Prof. Dr. Olga De Troyer

## Samenvatting

Tegenwoordig hebben bieden veel websites een enorme hoeveelheid aan informatie, die niet altijd op een overzichtelijke manier gebracht wordt naar de bezoekers toe. WSDM geeft een oplossing voor dit probleem, daar het ons in staat stelt om de verschillende doelgroepen van de website te identificeren en de informatie van de website op te delen naargelang deze doelgroepen.

In deze thesis zullen we een design geven van een tool die de designer in staat stelt om een doelgroepgerichte web applicatie te ontwerpen, aan de hand van de design methode van WSDM. Deze tool zal als een laag rond Web-for-Web gebouwd worden zodat men gebruik kan maken van diens functionaliteit voor een goed gestructureerde opslag van de gegevens van de website in de databank. Uiteindelijk zal na het design een Case Study besproken worden die de correctheid van het design en de implementatie van de tool aantoont.

## Acknowledgements

First of all I would like to thank Prof. Dr. Olga De Troyer for making it possible to realize this thesis and for the time she spent on reading the drafts of my thesis several times. The critical remarks she gave on the drafts have helped me a lot to come to a higher quality document for the final version of this thesis.

I would also like to thank my girlfriend Evy for the support she gave to me the last year and the patience she had when I was busy writing this thesis. She meant a lot to me last year. Furthermore, I would also like to thank Karl Evenepoel for proofreading the draft of this thesis and giving some suggestions for improvement.

And last but not least, I should be gratefull to my parents for their support during the year and for giving me the chance to study.

# Contents

# List of Figures

# Introduction

Nowadays, there is a growing realization that web designers should focus on the needs of the different types of visitors of a website. When a visitor has to search too long before he can find the information he is looking for, he will certainly leave the website and will never come back. This is a tendency we see in many large web applications. The amount of information these web applications offer is huge. Most of the visitors are mainly only interested in a small part of this information, not in the whole package. When the web designer divides the visitors into different categories, and groups the information that is suitable for that category, the visitor would find his information more easily, which means (directly or indirectly) more benefits for the company. This approach is called audience-driven [1].

In this thesis, I want to focus on websites that need to present a large amount of information that is stored in a database, in such a way that this information can also be used in other applications. For this purpose, the Web-for-Web [2] software was developed, which is an easy to use web interface to define and use a web interface for a database. However, since usually, a database contains a large amount of information, we also want this information to be presented in an Audience-Driven way, so that the visitors of the website will be able to find their information quickly. But this is not supported by Web-for-Web, which presents the information in an object-oriented way, not in an audience-driven way. Therefore, the goal of this thesis was to investigate how Web-for-Web could be turned into a so called Audience-Driven Web-for-Web, thus presenting the information to the user in an audience-driven way, with as less effort as possible. To achieve this goal, we have implemented the Audience-driven Web-for-Web as a layer on top of Web-for-Web, so that it can make use of its power of managing the content and structure of the database. This approach has several advantages: (1) The kernel functionality of Web-for-Web could be reused; (2) the original Web-for-Web could still be used for applications where an object-oriented approach is more appropriate; (3) Maintenance of the kernel functionality is localized; (4) Web-for-Web itself can still be used to edit and browse the data that has been used in the Audience-Driven web application.

The first chapter of this thesis will give a complete overview of Web-for-Web, combined with an overview of the technology that was used for its implementation. This is needed for the reader to understand the rest of the thesis. The next chapter will give an overview of WSDM (Web Site Design Method), the method that will be used as a guideline for the implementation of the Audience-Driven Web-for-Web application. In the third chapter, an overview of the main problems encountered in Web-for-Web, that would matter the adaptation to an audience-driven approach, are discussed. In the rest of the thesis, we explain the solutions for these problems.

The fourth chapter is the most important chapter of this thesis. It gives the design of the Audience-Driven Web-for-Web application. For each of the concepts we will use in the application, we will give a complete specification, a Database Diagram and a Class Diagram. With these explanations, the reader should have a good overview on the complete application and how it will work.

The design has been implemented, resulting in a working tool. In chapter five, a Case Study about a Primary School is elaborated to illustrate the new approach and the working of the tool. Finally, a description of some related work is given, which illustrate the need of the design of this new Audience-Driven Web-for-Web application.

# 1 Web-for-Web

In this chapter I will give an overview of the current Web-for-Web application [2] and the technology that has been used to develop it. First I will give an overview of the functionality of the current version of Web-for-Web and how it is organized. Afterwards I will give a description of the Three-Tier Architecture, which is the underlying architecture of the application and finally an overview of the technology that has been used in Web-for-Web.

## 1.1 Current Version

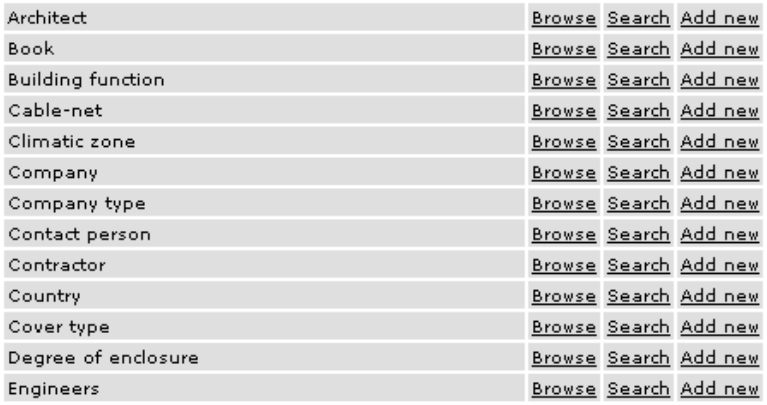
Many data intensive web applications use an underlying database to maintain the information on the web site. The different pages of such a web application will gather information from this database and display it in a pre-defined way on these pages. Such an approach has many advantages, but it has the disadvantage that when somebody suddenly decides that the current structure of the database is no longer adequate and that some tables should be added, modified or deleted, this may require considerable rewriting of the web application. This may be a lot of work and it would be easier when one would have a tool that allowed modifying the database and that was able to adapt the web application accordingly. It was for this purpose that Web-for-Web was given birth.

Web-for-Web gets is name from the fact that it provides a web interface for creating a web interface for a database application. Web-for-Web is a standalone tool, written in PHP, which is able to create tables and columns in a relational database and according to these tables the layout of the web interface is generated at run-time. The designer is able to specify which information from the database should be shown on the different pages. Web-for-Web uses an object-oriented approach for displaying and manipulating the information in the database. This information is viewed as a collection of Objects described by and grouped into Object Classes. An Object Class is mapped onto a table in the database and that table will be used to store the information related to that particular Object Class. For each Object Class, the designer of the web application is able to specify some Attributes. These Attributes may need some more explanation since they are the basic building blocks of the Web-for-Web pages and they will play an important role in the rest of this thesis.

An Attribute represents a piece of information that will be displayed on the page. Such an Attribute is mapped onto a particular column of the table of the relational database that is related to the Object Class the Attribute belongs to. The designer can specify different types of Attributes, which will represent different types of information. A small selection of these types is: a text-Attribute that represents plain text; a number-Attribute that represents a number; an email-Attribute that represents an email-address. There are more types of Attributes but we will return on this later on. The designer will group Attributes into Categories. The Categories are not mapped onto a table of the relational database. They are only used to group a number of Attributes into a larger collection. Categories will be used to indicate that some Attributes logically belong together, e.g. Street, City, Postal code and Country may be grouped in the Category Address. An example of this category is given in figure 1.1

| Address | |
|---|---|
| Street, nr | Kazernestraat, 9 |
| City | Gent |
| Postal code | 9000 |
| Country | Belgium |

Figure 1.1: Example of Web-for-Web Category



| | | | |
|---|---|---|---|
| Architect | Browse | Search | Add new |
| Book | Browse | Search | Add new |
| Building function | Browse | Search | Add new |
| Cable-net | Browse | Search | Add new |
| Climatic zone | Browse | Search | Add new |
| Company | Browse | Search | Add new |
| Company type | Browse | Search | Add new |
| Contact person | Browse | Search | Add new |
| Contractor | Browse | Search | Add new |
| Country | Browse | Search | Add new |
| Cover type | Browse | Search | Add new |
| Degree of enclosure | Browse | Search | Add new |
| Engineers | Browse | Search | Add new |

Figure 1.2: Example of a List of Object Classes

Next to Object Classes, Attributes and Categories, the designer can also specify Views on an Object Class. A View is a standard feature that is automatically defined by Web-for-Web by defining Attributes and Categories for an Object Class. In fact, it is just another word for the page you will be viewing. The designer has the option to specify captions (the name of the Attribute as it will be displayed on the page) and hints (small piece of text that is be displayed on a page next to the Attribute to give some help to the user on how to use the attribute) for the different Attributes. The designer also has the possibility to specify the order in which the Attributes are displayed. This will give him more flexibility to define the layout of the pages. All these options together define a View on an Object Class. It is possible to define multiple views for an Object Class, which can be done by specifying the categories that should be displayed within a particular view (this is typically done when a lot of attributes need to be displayed and when it is better to distribute them over several pages). The single view will then be divided into multiple Views, which are provided to the Visitor as multiple pages for one single instance. It is not only possible to define Views for viewing an instance of an Object Class, but the designer can also define views for editing an instance or printing an instance.

I will now briefly discuss how a web-for-web application works. The visitor of the web application has two possibilities: he can only view the instances of the Object Classes or he can update them. When he chooses the first option, he will get a list of available Object Classes. For each of these Object Classes, some default links are provided. These links are: 'browse', 'search' and 'add new'. Figure 1.2 gives an example of this list of Object Classes, taken from the

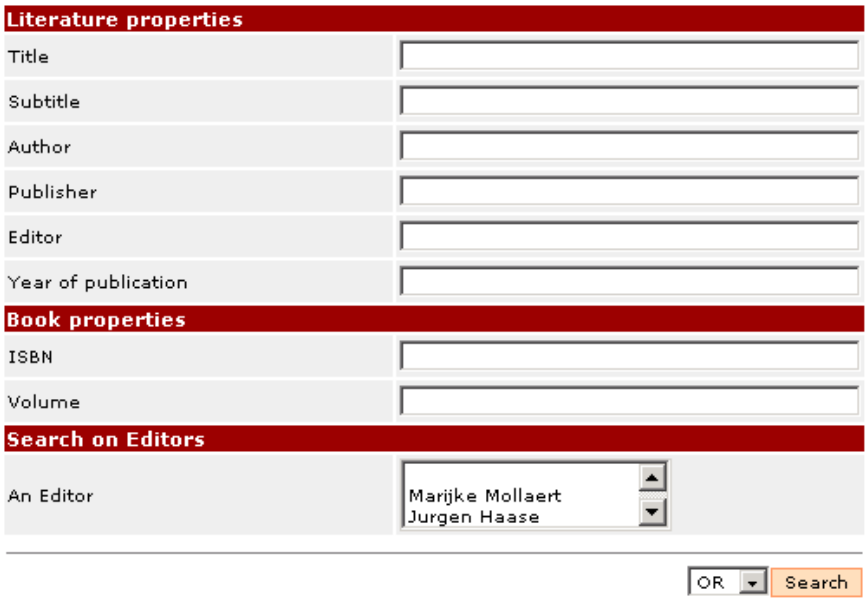



Figure 1.3: Example of a Search Page

| Name of the project | Year of construction | Project type | | | |
|---|---|---|---|---|---|
| Sony Center Forum Roof | 2000 | Cable-net | View | | |
| Hong Kong Park Aviary | 1991 | Cable-net | View | Edit | Delete |
| Sport centre at the King Abdul Aziz University | 1981 | Cable-net | View | | |

Figure 1.4: Example of a List of Instances


Figure 1.5: Example of a part of a View on an Instance

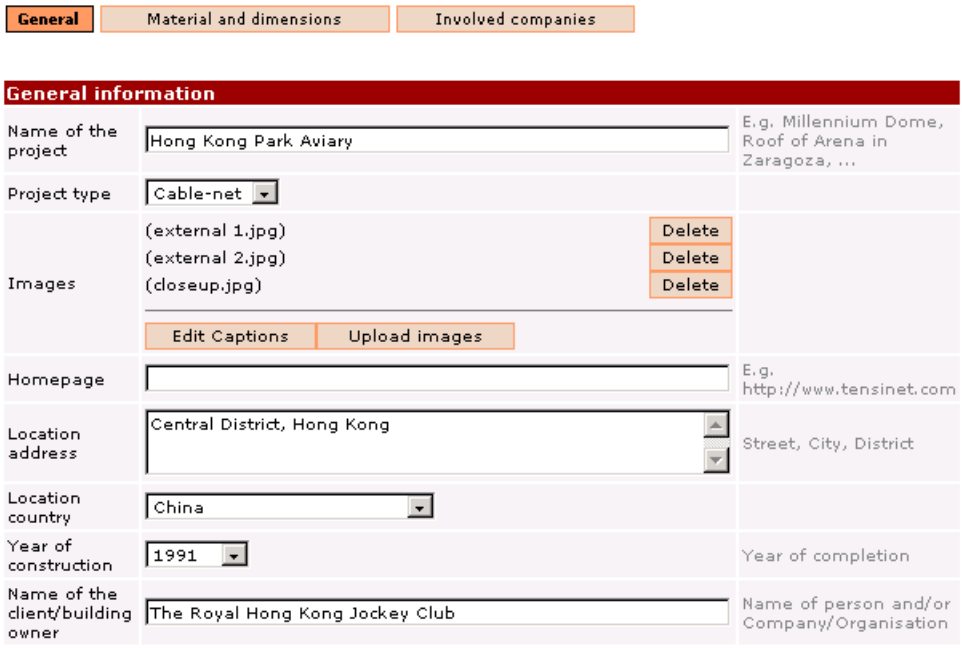



Figure 1.6: Example of a part of Editing an Instance

Tensinet website (www.tensinet.com). The last link, 'add new', is only available when the user is logged into the system and when he has permission to add new instances to that Object Class. The second link, 'search', will bring the user to a page with a form in which he may fill in some values for the attributes that are specific to that Object Class in order to find the instances in which he is interested. An example of this search-page for the Object Class 'Book' is given in figure 1.3. The first link, 'browse', gives him a list of all the instances that are available for that particular Object Class of which an example list of all Cable-Net projects is given in figure 1.4. For each of these instances, the following links are provided: 'view', 'edit', 'delete' and 'permissions'. These links are also available according to the permissions that are set by the owners of the instances. When the user has no permission to perform a given operation on a particular instance, the link will not be shown to him, in order to avoid abuses. The visitor can view the instance of the Object Class by clicking on the view-link, which will take him to View(s) of that instance. On this page he can see the different Categories with their Attributes and the values for each of these Attributes as stored in the database. An example of a View on the Instance 'Hong Kong Park Aviary' is shown in figure 1.5. One can also have the possibility to edit the instance, which will give him a View of the instance together with a form on which he can fill in or change the values for each of the different Attributes defined for the instance of that particular Object Class. An example of such a page is shown in figure 1.6.

The preceding part discussed how the Web-for-Web application works when the structure of the Object Classes is completely defined. But how can we define an Object Class? This is done at the Metabase level (the explanation of this

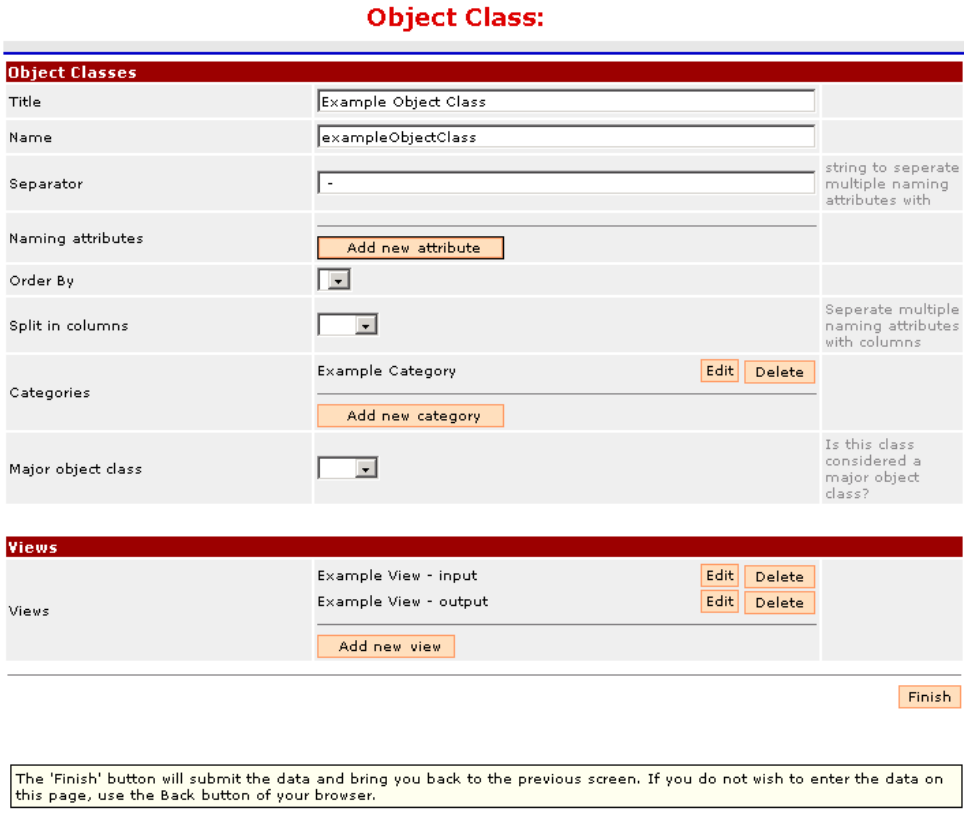



Figure 1.7: Add New Object Class

level is discussed in the following part about the Object Classes) by the designer, where he can manipulate (view, add, update) the Object Classes themselves. How can he do this? A picture of the form he gets when adding a new Object Class, is given in figure 1.7.

When adding a new Object Class, first of all, the user will have to specify a 'Title' (name) for the Object Class. This title will be displayed in the List of Object Classes that is given to the user when he enters via the Database level. The name of the Object Class is used to generate a valid name for the table on which the Object Class will be mapped in the relational database. The 'Separator' which can be specified is a string that will be used to separate multiple 'Naming Attributes'. A Naming Attribute is one of the Object Class's


Figure 1.8: List of Instances with Multiple Columns


Figure 1.9: List of Instances with separator '-'

Attributes that is used to identify the instances of the Object Class. One could use multiple Naming Attributes which can be separated by using columns or by using a separator. Examples, taken from the Tensinet website, for the two possibilities for the List of Instances are shown in figures 1.8 and 1.9. These figures also give an example of the use of the 'Order By'-field. The designer has the possibility to specify a default Attribute for the ordering of the Instances which can be descending or ascending according to his preferences. In figure 1.8, the 'Year of Construction'-Attribute was chosen with a descending ordering, so that the last year will be on top of the list. Only the Attributes that are associated with an Object Class can be used as Naming Attributes for this Object Class. These Attributes belong to a Category. So first of all, we have to add a new Category to the Object Class. A name has to be specified for this Category. In our example, we chose 'Example Category'. The properties that need to be specified for a Category are shown in figure 1.10. For each Category we can add as many Attributes as we want. We choose the desired type of Attribute for which we will have to specify the name of the Attribute, the order of it, a hint (we can see a hint in figure 1.7 for the Separator-Attribute) and some properties that are specific to that type. Finally, the designer will be able to specify the different Views for the Object Class. In this example, there are two: one for input and one for output. The system will create the Views in the appropriate way.

We have now discussed the fact that the designer is able to define Object Classes and specify the Attributes and Categories that belong to these Object Classes, but where is all this information stored? That is where the Meta Database comes up, of which its structure and its role is discussed in the next section.

### 1.1.1 The Meta Database

In Web-for-Web, two databases are used. The first database is the regular database, which contains all the information that will be shown on the different pages of the web application (further on referred to as the ***Database***). Also the accounts for the different users, used for the security system, are stored inside this Database. The second database is the ***Meta Database*** that contains a description of the data in the Database and how this data should be fetched to generate the appropriate pages for an instance. The Meta Database contains no information on the presentation of the information to the user. This is done by means of XSLT documents, which will be explained later on. The Meta Database contains three types of information:

- Information describing the structure of the data in the Database;

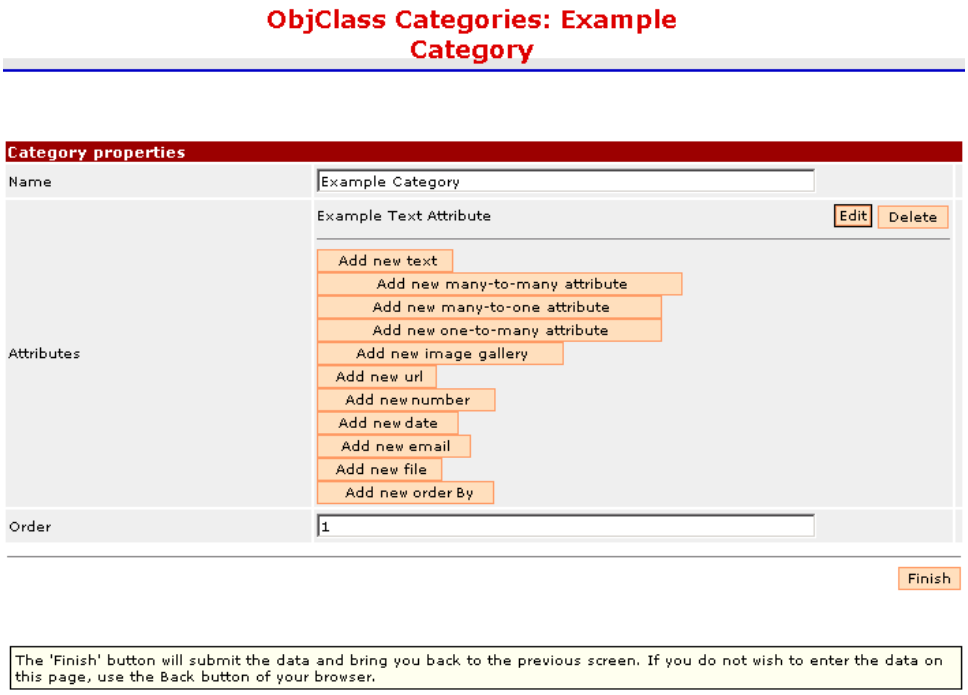



Figure 1.10: Add new Category

- 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.

The information that is stored in the Meta Database has been structured from an object-oriented point of view. An overview of the table-structure of the Meta Database is given in figure 1.11. The table, ObjClasses, contains the Object Classes. Starting from this table, we can get all the properties defined for a particular Object Class: the Categories and all the Attributes. Each Object Class is mapped onto a particular table in the Database. The name of this table is stored in the tableName-attribute of the ObjClasses-table. Each Attribute, stored in the Attributes table, corresponds to a particular attribute of the Database table. The name of the column is specified in the attribute: fieldname. To find the Attribute information for a particular instance, we just have to fetch the information that is stored in that particular attribute of the corresponding Object Class table.

Also the information on how the data can be manipulated is stored in the Meta Database. This is merely done by specifying who can update the information, specified by the security system of Web-for-Web. This is done by means of access rights, specified at the object level. The user can set access rights for each Object Class, for the Attributes that belong to the Object Classes and even for particular instances of an Object Class. These access rights will later on be used to generate the correct pages, based on the fact whether the user

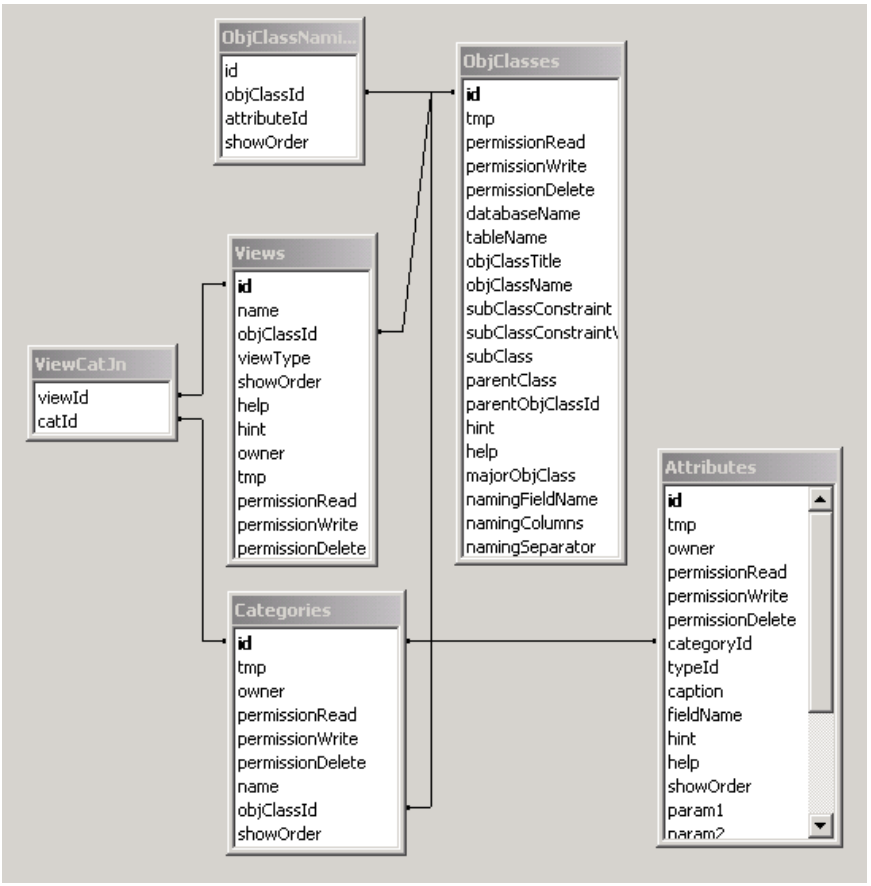



Figure 1.11: Table-structure of the Meta Database

has access to a given instance or not.

The advantage of the Meta Database is that the tool is almost independent on how the data is maintained and stored in the relational Database. When we would like to use e.g. an object-oriented Database, we would only have to rewrite the part of the Meta Database that describes how the Object Classes are mapped to the database structure. Also, by using two databases for the description of the data and the data itself, we will be able to use the data from the Database in other applications as well.

### 1.1.2 Conclusion

The current version of Web-for-Web is not very flexible. The way a visitor can consult the information in the database is rather fixed: first the visitor sees a list of Object Classes, from which he can select an Object Class. For this particular Object Class, a list of instances will then be displayed, from which he can choose one instance to view more details of that instance. This fixed structure fits well with an object-oriented view, where the focus is on one object (or object class) at the time and on how the data can be displayed in a uniform way, but for an Audience-Driven web application, much more flexibility is needed. In an Audience-Driven web application, the data should be grouped and displayed on a web page as needed by the users (audiences) of the website. This is not necessary in an object-oriented way.

Therefore to be able to generate an Audience-Driven web application, we need to adapt Web-for-Web. How this is to be done will be explained in more detail in the rest of this thesis.

## 1.2 Three-Tier Architecture

When designing a large web application, it is clear that this application should be easily maintainable. To ensure this, Web-for-Web was designed using a Three-Tier Software Architecture, where we encounter three levels of abstraction: the presentation level, the application level and the data level. The main purpose of this architecture is that these levels should assure that it is possible to change one them without having to change one of the other levels. This is a great aid towards maintenance of the web application, and is the most important advantage of this three-tier architecture.

### 1.2.1 Technical Details and Advantages

The three levels of abstraction are presented in figure 1.12 [2]. The first level, the presentation level, handles everything that is related to the presentation of the information. This level represents the user interface of the web application the user is browsing. Through this level, the user can interact with the application. He is presented some web pages, optionally with forms, on which he can click some links or enter some information and based on the event that he performs, the information will be sent to the application level, which will process the events and perform the necessary operations. Everything that happens within the application is triggered by this presentation level, which makes it

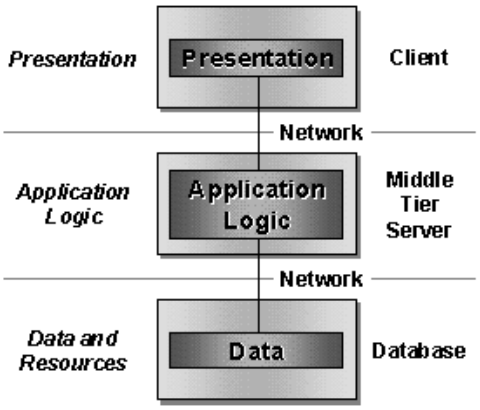



Figure 1.12: Three levels of abstraction

possible to change the layout of the web application without having to change the application's implementation.

The second level, the application level, deals with the correct functioning of the web application. It gets an event from the user interface, triggered by the user, and will take care of the correct handling of that event. E.g., when the user has sent some data through a form on a web page, the application level will capture this data and check for the correctness of it. Afterwards, this data will be processed by the application and the appropriate operations will be executed. When necessary, database queries are formulated and sent to the data layer which will ensure that the correct data is written to the database. When all the processing for the incoming data is done, this level will generate the necessary data to send this to the presentation level, which, in turn, will present the new page to the user, so that he can continue browsing the web application.

The third level, the data level, takes care of the operations that deal with all the operations concerning the database level. This data level gets a command from the application level to send a query to the database and will process this command according to the type of database that is used. The application level does not know about the type of database, so when the programmer decides to change the type of database, the only changes that will have to be made is at the data level. This level will ensure that the correct connections to the database are set so that they can be used to perform the correct actions.

These three tiers may all be separated on different machines, to improve the performance of the application, but this is not necessary. The presentation of the web application will be handled on the machine of the user, by the Internet Browser. Also the application level and the data level may be separated, since they do not run on the same level, they do not need to be on the same machine to run correctly. This implies that it is possible to use different machines for running the web application. The application level and the database level may

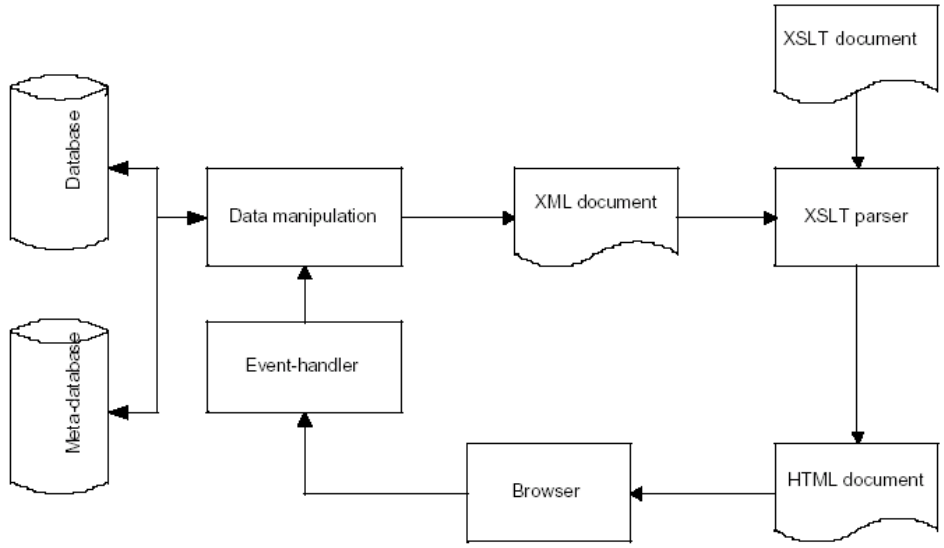



Figure 1.13: Web-for-Web Architecture

run on medium- or high-powered servers or on mainframes since they will have to be able to serve multiple connections to the application. And the presentation level will run on a low-end, single-user desktop, which is cheaper since it does not need all those capabilities of the high-powered servers. These possible separations of the levels over different machines give a great advantage towards the cost of the application, since no expensive servers are needed for each layer of the application.

### 1.2.2 The use in Web-for-Web

In Web-for-Web, the Three-Tier Architecture is also used. In figure 1.13 [3] a picture shows the architecture of Web-for-Web. I will now discuss the components that make up the different levels of the Three-Tier Architecture.

The first level, the presentation level, is represented by the parts: XSLT document, XSLT parser, HTML document, Browser, and Event-Handler. This part of the Web-for-Web application gets an XML document as input and converts the XML to an HTML document, using an XSLT document. The XSLT document contains information about how the XML document should be converted to regular HTML. The XSLT-parser takes the XML-document and the XSLT document as input and generates the HTML document that is returned to the browser. The browser returns the visual representation of that HTML document as a web page to the user, which he will use to continue browsing by clicking on a link or submitting form data to the application. When the user requests a page, his request is sent to the Event-Handler, which interprets the request and sends the right command to the application level.

The second level, the application level, is represented by the Data manipula-

tion part and the XML-document. The Data Handler, which is represented by the Data Manipulation part on 1.13, gets a request from the presentation level and handles this request by extracting the data from the Database according to the Meta information that is stored in the Meta Database. The fetching of the data from the Databases is done by sending a request to the data level, which will return the requested data. As a result of these operations, an XML-document is returned which only contains the data for a particular page and the way this data will be structured on the page. In this document there is nothing mentioned about how this data will be presented to the user, which follows the separation between the application level and the presentation level.

The third level, the data level, is represented by the Database and the Meta Database. In fact there is still a small part, called db-library, which provides all the functionality to access these databases, but this part is not shown explicitly on this figure. The current version of Web-for-Web makes use of Microsoft SQL Server, but in fact any DBMS could be used. When the implementation of the db-library is changed, a MySQL server or anything else could be used without having to change the implementation of the application level.

## 1.3 Technology Used

In this section we will give an overview of the technologies used to implement the Web-for-Web application. For each of these technologies, a short description will be given together with its role in Web-for-Web. The technologies that will be described are: PHP, XML and XSLT.

### 1.3.1 PHP

PHP (recursive acronym for: "PHP: Hypertext Processor") is a server-side, cross-platform, HTML-embedded scripting language. Much of PHP's syntax is borrowed from C, Java and Perl with a couple of unique PHP specific features. The goal of the language is to allow web developers to write quickly dynamically generated pages. PHP eliminates the need for numerous small CGI programs by allowing to place simple scripts directly in HTML files.

PHP is an excellent alternative to similar programming solutions as Microsoft's ASP and Macromedia's Coldfusion. As mentioned before, PHP is a cross-platform language and this is not only the case for the core PHP code, but also for all the libraries of PHP. It can run on all major operating systems, including Linux, many Unix variants, Microsoft Windows, Mac OS X and has also support for most of the web servers today, including Apache, Microsoft Internet Information Server, Personal Web Server ... For the majority of the servers PHP has a module, for the others PHP can work as a CGI processor. With PHP you have the freedom of choosing the operating system and web server that you like, which is not the case for ASP and Coldfusion.

Figure 1.14 [4] shows an example of a PHP-script. This example script is different from writing a script in a different language like Perl or C, since in those languages you have to write lots of commands to output HTML, where in PHP you write an HTML script with some embedded PHP code in it, which

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
        echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

Figure 1.14: Example of a PHP-script

will perform some operations. The PHP code is enclosed in special PHP tags: <?php ... ?> which show us when the PHP code start and when it ends.

### What can PHP do?

PHP is mainly focused on server-side scripting, so you can do anything that any other CGI program can do, such as collecting form data, generating dynamic page content or sending and receiving cookies. But PHP is capable of doing much more. There are three main areas where PHP scripts are used:

- *Server-side scripting.* This is the most traditional and main target area for PHP. Three things are needed to make this work: the PHP parser, a web server and a web browser. The PHP program output can be accessed with a web browser, and the PHP page can be viewed through the server.
- *Command line scripting.* A PHP script can be made to run without any server and any browser. The only thing that is needed is the PHP parser made to use it that way. This type of usage is ideal for scripts which are regularly executed.
- *Writing client-side GUI applications.* PHP is not the very best language to write windowing applications, but for those who know PHP very well, and who would like to use advanced PHP features, PHP-GTK can be used to write such applications. In this way, you also have the ability to write cross-platform applications. PHP-GTK is an extension which is not available in the main distribution, but that can be easily installed.

A programmer of PHP applications also has the ability to choose between procedural and object-oriented programming or he can use a mixture of both programming styles, which is a great benefit since not every programmer likes the procedural programming style most of the scripting languages are working with.

With PHP it is not only possible to send HTML output to the browser, but it also supports the output of images, PDF files and even Flash movies using specialized libraries that can be included in the PHP package. It is also possible to output any text such as XHTML and any other XML files. PHP can auto generate these files and save them in the file system instead of printing them on the screen, forming a server-side cache for the content of the page.

The strongest feature of PHP is the support for a wide range of databases. This makes it very easy to write database-enabled web pages. A small set of the supported databases is: mSQL, MySQL, Sybase, Oracle and many others. Together with the standard functions that are provided to access each of these databases, PHP also supports the Open Database Connection standard (ODBC), so it is possible to connect to each database that supports this world standard.

But these are not the only things to which PHP can connect. PHP has also the support to connect to other services using protocols such as LDAP, IMAP, POP3, SMTP, HTTP and many others. Next to these connections, it is also possible to open a raw network socket and interact with any other protocol.

All these possibilities are just a glimpse of what we are able to do with PHP. We will not go into deeper detail since this is out of the scope of this thesis and there are many good books [5, 6, 7] explaining which fancy things you can do with this language and how you should start programming in it. Here, we only provided a short description of it so that the reader of this thesis, who has no knowledge about PHP, gets a good view on PHP's capabilities.

**The use in Web-for-Web**

PHP is the core programming language that is used for the implementation of Web-for-Web. Not only the approach of Web-for-Web is object-oriented, but also its implementation is mainly object-oriented (some small parts are implemented using a procedural programming style). The db-library that takes care of the connections to the databases and the execution of the queries on these databases is just a set of functions that can be used throughout the complete tool. This library is part of the implementation that uses a procedural programming style. Also the security issues are formulated in terms of functions and not grouped in a class. The last part of the procedural programming part, is the XSLT-parser, which will convert XML to HTML.

A class diagram of the Web-for-Web application is given in figure 1.15. This class diagram is not complete; some of the classes that are not very important to discuss here are omitted in this figure and also the operations of the classes are not shown, because this would make the diagram too complicated and would not give us a good overview. We can see that an object-oriented programming style has been used. Everything starts with the EventHandler-class that gets a request for a particular web page. There are two subclasses for the EventHandler: the MetaEventHandler and the CustomEventHandler. These two subclasses will be used according to the mode we are in: defining the structure of an Object Class or defining the contents of an Instance of an Object Class. When we are defining the structure of the Object Class, the MetaEventHandler will be used since in this case the operations should be performed on the Meta Database. In the other case, where we are defining the contents of an instance of an Object Class, we will use the CustomEventHandler. The EventHandlers will send a command/request to the DataHandler that will handle these. The DataHandler will perform the necessary operations on the appropriate database, or delegate operations to the more appropriate classes for this: Attribute, ObjClass, View

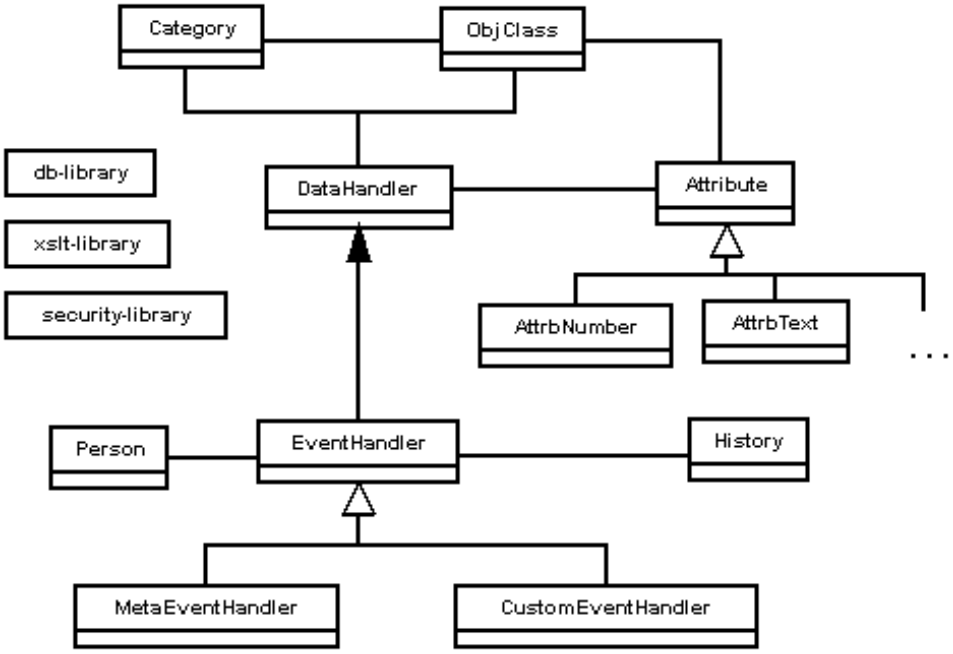



Figure 1.15: Web-for-Web Class Diagram

... For each type of Attribute a separate class is created, that inherits and can override operations from the parent class Attribute, so that each class can perform the correct operations according to that type of Attribute. The db-library, xslt-library and security-library are files that are included in the Web-for-Web application so that they are available to every class of the system.

### 1.3.2 XML

The Extensible Markup Language (XML) [8, 9] is not a language for marking up text, nor is it a replacement for HTML. XML is a standard syntax for in-line markup for use in text documents and it also includes facilities for defining a set of markup elements that are used together as an application. Really explaining XML, however, requires a little history.

#### History

The Standard Generalized Markup Language (SGML) [10] was created in an effort to standardize markup systems. In order to handle all of the features of the markup systems of the time, the SGML design was comprehensive, with many optional features and shortcuts. These features made it a very powerful system. But they also make SGML tools difficult to program.

SGML is actually not a markup language; it is a meta-markup language. SGML provides a language to support the definition of new markup languages

that are called SGML applications. All SGML applications have a similar structure. Unlike many proprietary systems, all SGML markups use normal printable characters. Markup is defined in terms of elements and text. Elements are made up of tags and attributes. All elements, tags, and attributes must follow a well-defined format. This simplifies validation and translation of the documents. Separate from the issue of format is the validity of the tags used in a particular document. SGML prescribes the use of a Document Type Definition (DTD) to specify which SGML application pertains to the document. A processing system could then use information from the DTD to validate a particular document.

Technically speaking, HTML can be called an application of SGML. It is possible to define a SGML DTD for HTML. HTML uses a set of fixed tags and we are not able to add our own tags. As it is the case for SGML, in the beginning, HTML was designed to only describe the structure of a document, not its presentation, but the first browsers soon defined a default presentation for many of the structural elements. Most people started focusing on the presentation aspect of HTML and less on the structural meaning of the tags. Most browsers like Netscape and Internet Explorer started to extend HTML to support more presentational control. In this way, a lot of tags were added to HTML, and in the most recent version of HTML, there are about 100 pre-defined tags, but still there is no support for adding new tags. It is also possible to make mistakes inside an HTML document since most browsers have the ability to interpret non-well-formed tags and give a decent presentation to them.

When handling data on the net, it may be clear that one could need more tags to deal with the information in a uniform way. Like when somebody would be selling cars on the internet, he would probably prefer to have tags like <HORSEPOWER>, <PRICE>, <CONSTRUCTIONYEAR> to present the cars he is selling in a uniform way. This is something that is not possible within an HTML document.

The need for a specialized markup language that is simpler than SGML and that has more possibilities than HTML grew. And as a result of these two markup languages, XML was born. It combines the best of the two markup languages and tries to overcome their worst disadvantages. We are able to define new markup languages, just like in SGML, but a much simpler feature set is used in XML, which makes it easier to parse the documents. Also a document must be well-formed, otherwise the XML-parsers should report an error. The way the XML documents have to be presented is described in a separate file, which also supports the separation of the contents from the presentation, as seen in the Three-Tier Architecture (see section 1.2).

#### XML Document Structure

Figure 1.16 [11] is an example of a well-formed XML document that shows the different document parts and the items that can be added to each part. An XML document consists of two main parts: the prolog and the document element, also known as the root element. In addition, following the document element, a well-formed XML document can contain comments, processing instructions and white spaces. In the prolog we state the declaration of the XML,

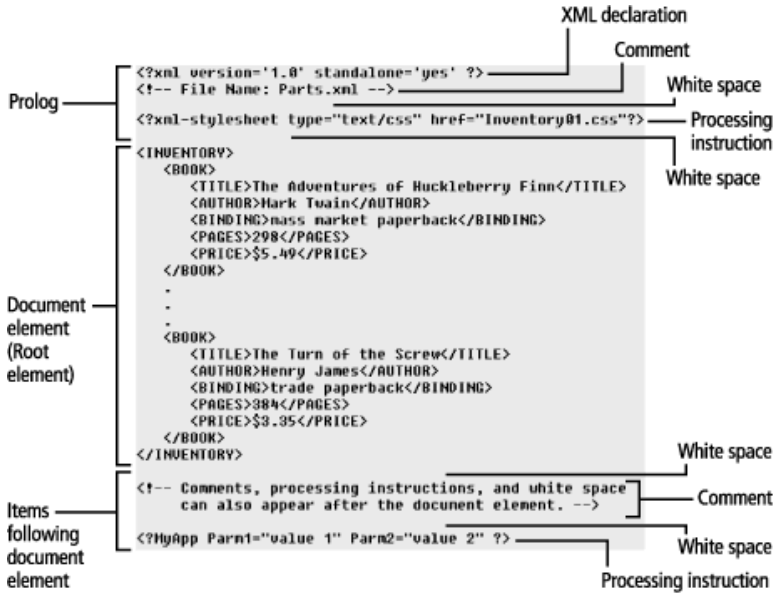



Figure 1.16: Example XML Document

and together with this declaration we specify the XML version number. The version number is optional but it is best to include this since there might come newer versions of XML.

The elements in the XML document contain the actual information for that document. They define what will be shown on that page. The name of an element should start with a letter or an underscore and may not start with 'xml'. Furthermore there are no restrictions on creating new names for elements. The elements are arranged in a tree-like hierarchy with elements nested within other elements; the document should have one top-level element, the root element, with all the other elements nested within it. Unlike in HTML, every element must have a start-tag and an end-tag, except for the empty element that has a special form. For our example in 1.16 the start-tag for the book is <BOOK> and the end-tag is </BOOK>. All the other tags that are in between these tags are part of the book-element. In this way it is possible to group important information together so that it can be easily displayed to the user. An important requirement of the nested elements is that the elements should be nested properly. If an element begins inside another element, it must also end within that element. Otherwise an error message will be returned. The empty element tag is not presented in the example, but for example when there is no price defined for a particular book, than we could present the price by the following empty tag: <PRICE />.

Every element can contain a number of attributes. An attribute can be included in the start-tag of an element or in the empty-element tag. An attribute specification is a name-value pair that is associated with the element. All attributes take the form of a name, followed by an equal sign (=) followed by a

value in either double or single quotes. An attribute has the restriction that it can only appear once in a given tag, but it gives the possibility to the writer of the document to include information in the element in an alternative way. The following example shows an example of a book containing the category-attribute:

<BOOK category="fiction">
<TITLE>The turn of the Screw</TITLE>
<AUTHOR>Henry James</AUTHOR>
</BOOK>

White spaces in a XML document, which consist of one or more space, tab, carriage return or line feed characters, will not be filtered out as in HTML. Since it is possible that the white spaces can be an integral part of the content of an element, but sometimes they can be there just to make the XML document easy to read. Since the XML parser cannot know the intention of the programmer, all the white spaces will be sent to the application.

In order to display the XML document as a web page, some form of style sheet is needed to explain the formatting to the browser. The two most common style sheet languages that are associated with XML are Cascading Style Sheets (CSS) [12] and Extended Stylesheet Language (XSL) [13]. CSS style sheets are currently used in many web browsers for HTML purposes and it is the most standard way of separating formatting information from HTML content. And it can also be used for XML documents to show them to the browser. XSL was developed exclusively for use with XML documents. Where XSL is an XML vocabulary for formatting semantics, XSLT will be used as a transformation language for XML documents. A more in detail explanation of XSLT will be given in the next section.

This is just a short introduction on XML, but it is sufficient as background information to get through the rest of this thesis. There are still some more specialized features of XML, the Document Type Declarations (DTD) document being one of them, but these features will not be used during this thesis and an explanation of them would be out of the scope of this thesis.

**Advantages**

The first benefit that comes up is the fact that XML documents are **extremely readable**. The tags are used to describe what the reader can expect from the contents. This is not always the case in HTML documents. For example when we take the above example, we know that we are dealing with a book, and the elements that belong to this book are: title, author ... The element tags describe the type of contents that will be represented. This is not the case for HTML-tags. When we would give the <LI> tag to a person who does not know anything about HTML, he would not know that this is the tag for an element of a list.

The second one is that the **presentation of the data can be separated from the data itself**. This gives us the benefit that we do not have to change the complete document when the data would change. This is the case in HTML

documents where we have to change the complete document every time the data has changed. In XML we define the presentation of the data in a separate style sheet file that will specify the layout of the web page.

As a last important advantage, it is easy to **search for information** in an XML document. Through the element-tags, one can easily find what he is looking for, since they already give a description of the contents of the element. In this way, we do not have to fight ourselves a way through the presented data as in other documents.

#### The use in Web-for-Web

As mentioned before, Web-for-Web uses an object-oriented approach to display its information. XML can be easily used to structure information in an object-oriented way, since all the information of a particular object can be grouped together into one XML element. But this is not the only reason why XML was chosen as an output format. Since it should be possible to use Web-for-Web for multiple projects without having to change its implementation, the presentation of the data should be separated from the data itself; otherwise we would need to rewrite the presentation part of the Web-for-Web application for every new project. This would be the case e.g. when Web-for-Web would directly give HTML output. Using XML, we can separate the presentation of the data from the data itself, since the XML documents are converted by using XSLT transformation scripts, which will be explained in further detail later on in this thesis.

Different elements are used to represent an appropriate XML document. The root element is the <page>-element, which states that all the elements that are in between these element tags will be displayed on the page. One of the tags that are obliged to be in the document is the <title>-element, which contains the title of the web page. Furthermore we have <lstObjClasses>- and <lstInstances>-elements which states that from that point on, a list of Object Classes or Instances of an Object Class will be shown to the user. Enclosed in these tags, a specification of the (Instances of) Object Classes is given by the <objClass>-elements, which on their turn have their own child elements that can be specified together with them, such as <id>-element, <name>-element, <attrb>-elements (which are different for each type of Attribute) With this kind of XML document we are able to define the structure of the information that needs to be shown on the web pages.

### 1.3.3 XSLT

The Extensible Stylesheet Language Transformations [14] is mainly used to transform one vocabulary into another. A transformation expressed in XSLT describes rules for transforming a source tree (a XML document) into a result tree (e.g. a HTML document). The main purpose of XSLT was to transform an XML document into an XSL document, but this is not necessarily the case, since the output format can be anything the designer wants, including HTML. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree.

The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added. A transformation expressed in XSLT is called a style sheet. This is because, in the case when XSLT is transforming into the XSL formatting vocabulary, the transformation functions as a style sheet. A style sheet contains a set of template rules, which will be matched against the elements from the XML document. These template rules allow us to use the style sheet for a variety of XML documents with similar source trees.

A template that matches with a particular element of the source tree, is called and instantiated for that particular element. The code that is associated with the template will be completely executed and it will return a part of the result tree. The operations can also access the child source elements and apply the appropriate templates for these children. In this way the complete result tree can be built using all the parts resulting from the application and instantiation of the templates. This process is started by applying the template to the root element and going down in the hierarchy.

**XSLT Document Structure**

An XSLT document should always begin with the <xsl:stylesheet version=-"..."> tag and end with closing this one with </xsl:stylesheet>. In between these two tags, the contents of the XSLT document can be specified. The order of further children of the stylesheet-tags is not important. These children are called elements of the xsl:stylesheet element and these possible elements are: xsl:import, xsl:include, xsl:param, xsl:template ... In addition to these elements, the xsl:stylesheet element may also contain other elements which are not in the XSLT namespace. Such elements can provide information about what to do with the result tree, information about where to get the source tree, metadata about the style sheet Now we will give an overview of the structure of an XSLT style sheet and discuss the possible operations.

The example in figure 1.17 could be a good XSLT template for the example of the book-inventory of figure 1.16. This example is far from complete, but it gives us a first impression of what we can expect from an XSLT style sheet. The Inventory-template would be matched with the INVENTORY-element of our XML document and the code inside the INVENTORY-template will be executed. In this example we can see that the Inventory will be displayed as a collection of tables, each table representing a book. This example gives us some good examples of some basic building blocks needed to make a good XSLT style sheet.

First of all we can see the <xsl:value-of select="TITLE">-element. This element will fetch the information from the <TITLE>-element of the XML document. It is only possible to fetch this information when this element is a direct child element of the <BOOK>-element. The title-information will be displayed inside a cell of the table. A second example is the <xsl:value-of select="$detailsAuthor">-element. At first sight, it looks the same as the previous example, but we can see the $, which means that detailsAuthor will be

```
<xsl:template match="INVENTORY">
    <xsl:for-each select="BOOK">
        <table class="tblRegular">
            <tr>
                <td class="tdTitle">
                    <font class="txtTitle">
                        <xsl:value-of select="TITLE"/>
                    </font>
                </td>
            </tr>
            <tr>
                <td class="tdAuthor">
                    <a class="lnkRegular">
                        <xsl:attribute name="href">
                            details_author.php?name=<xsl:value-of select="AUTHOR"/>
                        </xsl:attribute>
                        <xsl:value-of select="$detailsAuthor"/>
                    </a>
                </td>
            </tr>
        </table>
    </xsl:for-each>
</xsl:template>
```

Figure 1.17: Example of an XSLT Template

```
<xsl:template match="INVENTORY">
    <xsl:for-each select="BOOK">
        <xsl:if test="BINDING='trade paperback'">
            <table class="tblRegular">
            ...
            </table>
        </xsl:if>
    </xsl:for-each>
</xsl:template>
```

Figure 1.18: Example of Conditional Processing in XSLT

a variable that is defined inside the XSLT style sheet, by <xsl:param name-="detailsAuthor">Details Author</xsl:param>. In this way, it is possible to define variables that will be used more than once throughout the document. A third example is the <xsl:attribute name="href">-element. The xsl-attribute-element will set the value of the attribute of a previous defined element to the value that is defined within its tags. In this case, the href-attribute of the link will be set to point to the 'Details Author'-page. The result of this operation will be a well-formed HTML-link. The last element that we can see in this example is the <xsl:for-each select="BOOK">-element. This element will perform all the operations that are defined inside its tags for each BOOK-element that is defined in the XML document between the INVENTORY-tags, since we are in the INVENTORY-template. But also conditional processing is supported. This is shown in figure 1.18 where an if-test is performed to test whether the binding of the book is trade paperback. When the test is true, the book will be displayed in a table, otherwise no operation will be performed. The result of this example will be a list with only the books with a binding that equals 'trade paperback'.

These examples give a small view of what XSLT has to offer, but of course

this is not the complete specification, since we are also able to include other XSLT style sheets and use the contents of these files by using the xsl:include statement, the when-element and a lot more fancy stuff which is out of the scope of this thesis. For more information about XSLT and a detailed description of all the elements and functionality that is supported by this language, I recommend the W3C recommendation on XSL Transformations [15].

**The use in Web-for-Web**

The transformation from an XML document into a HTML document is done completely using XSLT documents. A variety of XSLT documents are defined and they are able to transform the XML documents, coming from the DataHandler, into HTML documents that can be returned to a browser. For each of the different types of pages that can be shown by Web-for-Web, an XSLT document is created. These types of pages are: view, edit, print and search. This is a good example to illustrate that it is possible to use different style sheets for the same source tree (XML document). The difference between the separate style sheets is that the view and print style sheets will display the contents of the database with plain text inside tables, while the edit and search style sheets will give the contents of the database with HTML form fields. This is necessary to be able to fill in the form fields to be able to interact with the Web-for-Web application. The correct style sheet will be chosen by the application according to the type of page that has to be shown.

The different style sheets have one thing in common: they all have some files which they share: shared.xsl, dutch.xsl and english.xsl. These three files are quite important for the good functioning of the XSLT style sheets. The dutch.xsl and the english.xsl style sheets are used to be able to provide versions of the tool in different languages. The designer will have to choose the file he wants to include and according to this choice, the standard help texts, text on the buttons and text for the standard links, will be provided in the appropriate language. The user has the ability to create more language files with the same parameters as defined in the original files. In this way, Web-for-Web can be used for other languages as well.

The shared.xsl file is an important file since it defines the layout of the page. It takes the form of a HTML-template and invokes the application of the XSLT templates in the style sheets. The benefit of an HTML template is that one can define a standard layout for every page, which is likely to be changed. Web-for-Web has no possibilities to define a fancy layout, since it is possible to use advanced tools for this like Macromedia Dreamweaver or Microsoft Frontpage. But due to the lack of this, we have this shared.xsl file. In this file, we can put our layout that we defined using such an advanced tool. In this layout, we should have left some space where we can put the output coming from the Web-for-Web application. We indicate this by putting the xsl:apply-templates element on the spot where this contents may come. With this technique, we incorporated the power of using an advanced layout together with the application, without having to implement this feature inside the tool.

The transformation of XML to HTML is invoked by initiating the built-in

XSLT-parser from PHP. The XML document is given to this parser, together with the appropriate XSLT style sheet. The parser will check both documents to see whether there are errors in one of those documents. When this is so, an error will be reported, since both have to be well-formed. When no errors are reported, the HTML document will be returned as a result of the transformation. And afterwards, the HTML page will be shown to the visitor of the web site through the browser.

# 2 Web Site Design Method (WSDM)

In this chapter, I will explain WSDM (Web Site Design Method) [16, 17, 1, 18, 19], which is a design method that uses an audience-driven approach, which is in contrast to most other web design methods that take a data-driven approach. WDSM will be used as a guideline to come to a good design for the Audience-Driven Web-for-Web application. First of all I will give an overview of WSDM itself and the relation of WSDM to Web-for-Web.

## 2.1 Overview

Figure 2.1 gives an overview of the different phases in the design process of WSDM. In this section I will give a brief explanation for the role of each phase and what is done in the different phases. We will use a website of a Primary School as an illustration of the different phases of WSDM:

*"Provide a website where parents and students can find some general information about the school,the teachers, the activities of the school and the different classes. Furthermore we want to provide a place where teachers can exchange some private information."*

### 2.1.1 Mission Statement Specification

The first step in WSDM is defining the Mission Statement. With the Mission Statement we want to express the purpose and subject of the website and declare the target audience(s). For our primary school example, this gives:

- *Purpose*
  - Offer some information about the school
  - Allow teachers to exchange private information
- *Subject*
  - Information about the school, its teachers, its classes and its activities
- *Target Audience*
  - Teachers
  - Students
  - Parents

### 2.1.2 Audience Modeling

With the Mission Statement as input, we come to the second step, the *Audience Modeling* phase. This phase is divided into two different steps: the first step is the *Audience Classification* step and the second one is the A*udience Class Characterization* step.

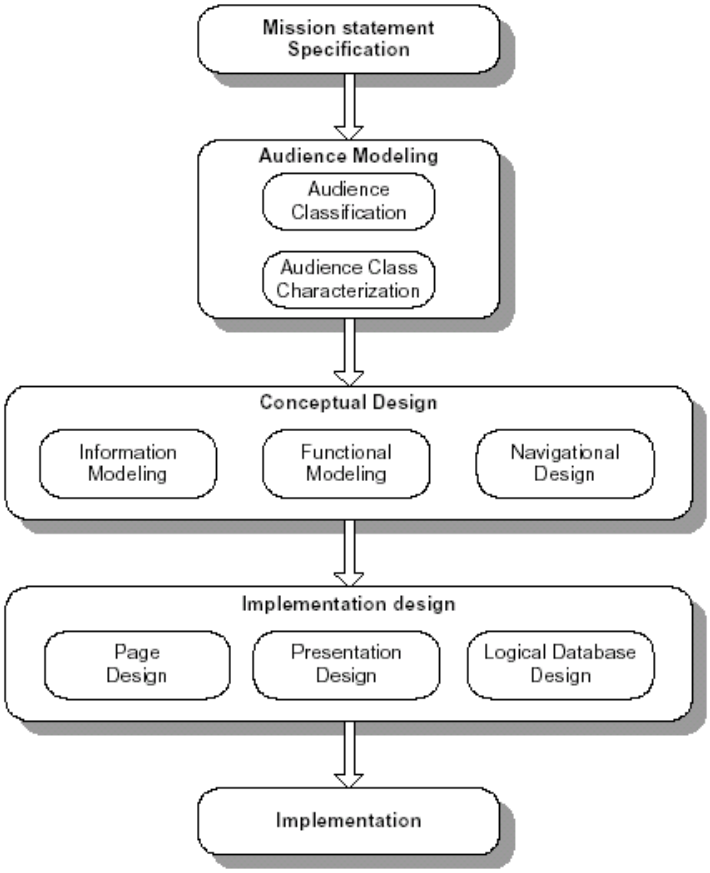



Figure 2.1: WSDM Overview

**Audience Classification**

During the Audience Classification step, we will identify and classify the different types of users into Audience Classes. Members of the same Audience Class will have the same information and functional requirements. The Audience Classes need not to be disjointed; a user may belong to one or more Audience Classes. When some Audience Classes have requirements in common, we may choose to use subclasses. A member of a subclass will contain all the requirements of its superclass, plus some additional requirements. When all the Audience Classes are defined, we can create an Audience Class hierarchy in which the Audience Classes are shown in terms of sub- and superclasses. One of the conventions in WSDM is that the top class of the hierarchy is always called the Visitor-class. All the other Audience Classes will be (direct or indirect) subclasses of this Audience Class. For our example this gives us:

**Audience Class Teacher**
*Information Requirements:*

- *Private information about classes*
- *Important private documents*

*Functional Requirements*:

- *Add some information to a class*
- *Add a document*
- *Delete information*
- *Delete document*

**Audience Class Student**
*Information Requirements:*

- *Public information about the classes*
- *Pictures/reports of the organized activities*

*Functional Requirements:*

- *None, since a Student can only browse through the information*

**Audience Class Parent**
*Information Requirements:*

- *General information about the school*
- *Information about the activities*

*Functional Requirements:*

- *None, since a Parent can only browse through the information*

The Audience Class Hierarchy Diagram for these Audience Classes is given in figure 2.2.

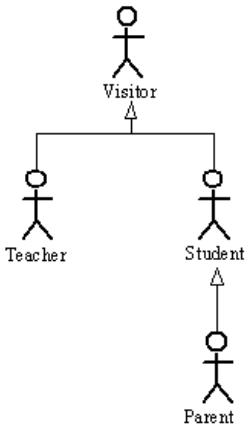



Figure 2.2: Audience Class Hierarchy Diagram

### Audience Characterization

The second step in the Audience Modeling phase, the *Audience Class Characterization* step, will define all the characteristics of the different Audience Classes. We will specify, for example, the average age for a particular Audience Class, or the experience level of the members of that Audience Class, or  For our example we have:

**Audience Class Teacher**
*Characteristics:*

- *adults*
- *mostly good experience with WWW*
- *speaks English*

**Audience Class Student**
*Characteristics:*

- *between 3 and 12 years old*
- *mostly not yet much experience with WWW*
- *speaks English*

**Audience Class Parent**
*Characteristics:*

- *adults*
- *experience with WWW varies*
- *speaks English*

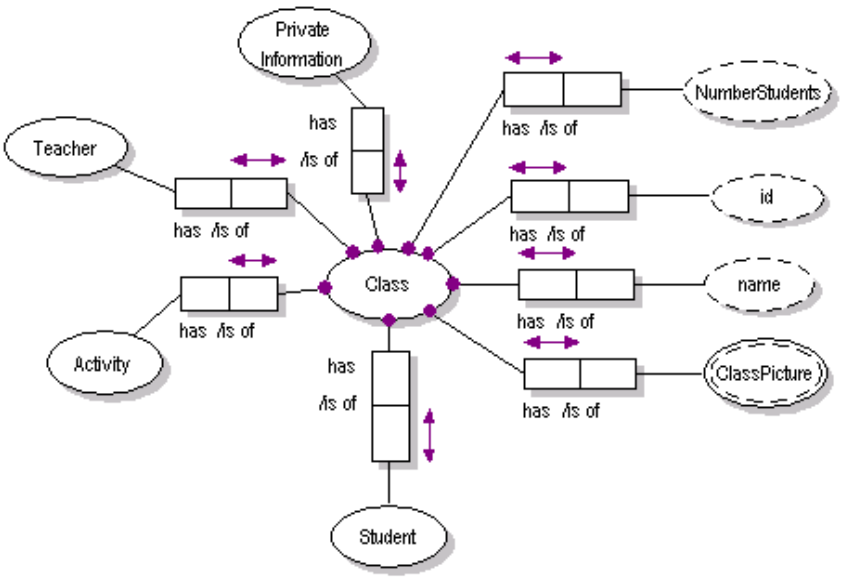



Figure 2.3: Object Chunk for "Class" of the Primary School example

### 2.1.3 Conceptual Design

With the information from the Audience Modeling phase as input, we can come to the third phase, the *Conceptual Design* phase. This phase is divided into three different steps: *Information Modeling*, *Functional Modeling* and *Navigation Design*.

#### Information Modeling

During the *Information Modeling* step, we will model all the information requirements from the Audience Classes using Object Role Modeling (ORM) [20]. The information requirements are decomposed into several elementary information requirements for which we will make an Object Chunk. An Object Chunk will be modelled using ORM, but any other technique like Entity Relationships (ER) or Unified Modeling Language (UML) [21] or etc can be used too. The Object Chunk itself corresponds with an Entity Object Type in ORM and consists of a number of Value Object Types or has relations with a number of other Entity Object Types. These relations may represent relations to other Object Chunks in the Information Modeling step. Figure 2.3 shows the Object Chunk for the Information Requirement 'Public information about the classes' of the Student Audience Class.

#### Functional Modeling

In the *Functional Modeling* step, we will make use of functional chunks in order to be able to describe the functionality for the different Audience Classes. For each functional requirement, as well as each information requirement, elementary requirements are elaborated and for each elementary requirement a

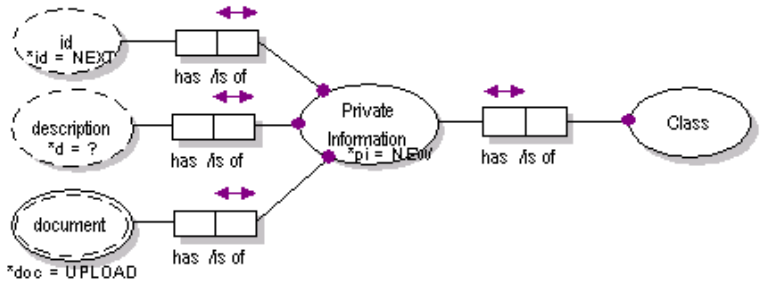



Figure 2.4: Functional Requirement for Primary School example

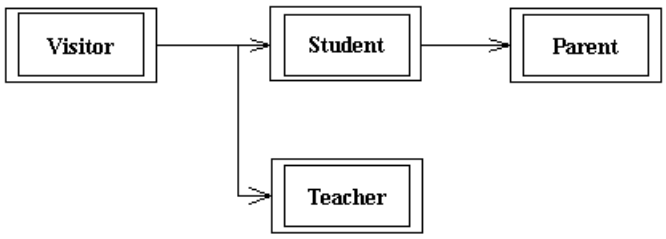



Figure 2.5: Different Navigation Tracks for the Primary School example

functional chunk is created. These functional chunks are created using ORM with some extensions in order to be able to model the functionality. Figure 2.4 gives an overview of the Functional Requirement 'Add some information' of the Teacher Audience Class.

### Navigation Design

The *Navigation Design* will model how the members of the different Audience Classes will be able to navigate through the web application. For each Audience Class, we will define a Navigation Track. For each task that is required for the Audience Class, a Task Model is defined. These Task Models will be linked within the Navigation Track through the concept of Structural Links, which may also be conditional links. A Task Model itself will describe the different steps in the task and the sequence in which these steps can be performed. These Task Models will be composed out of components, which represent a step in the task and process logic links, which will indicate the sequence between the steps. Since a component corresponds with an elementary task and there is an Object Chunk created for each elementary task, we will have to connect the components with its corresponding Object Chunks. When we have defined each of these components within the Task Models, we will have the complete Conceptual Structural Model that we were planning to define with the Navigation Design. Figure 2.5 shows us the different Navigation Tracks in the website and figure 2.6 gives an overview of the Student Navigation Track.

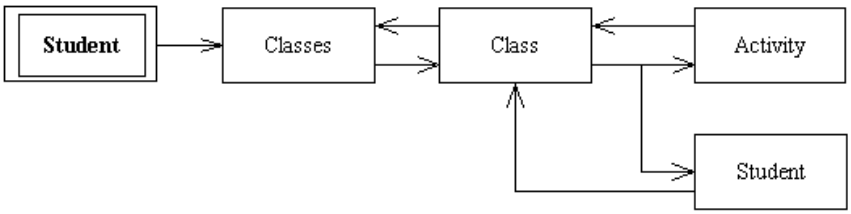



Figure 2.6: Navigation Design of Student Navigation Track

### 2.1.4 Implementation Design

The fourth phase of WSDM is the Implementation Design phase, which consists of three steps: the *Page Design*, the *Presentation Design* and the *Logical Database Design* step.

#### Page Design

During the *Page Design* step we will group the components and links that we get from the previous phase into pages. In this step, the designer will have the freedom to group several components connected by links and represent its information on one single page, or he may choose to spread the information on different pages.

#### Presentation Design

When he has defined the content of the different pages, he will come to the second step, the *Presentation Design*. During this step, the designer will specify the look and feel of the web application. And also the layout of the different pages will be defined. This step is not yet completely specified in WSDM, so we do not have a lot of information about this step yet.

#### Logical Database Design

The last step in the Implementation Design phase is the *Logical Database Design*. This step will only be performed when the web application needs a database as back-end. The conceptual Information Model can be used to generate a logical database schema.

### 2.1.5 Implementation

The fifth and last phase is the *Implementation* phase. In this phase, we will actually build the web application that we have designed in the previous four phases. We will choose our implementation environment: HTML, XML and we will start implementing the web application according to the WSDM design document.

## 2.2 Relation to Web-for-Web

WSDM is a method for designing web application using an audience-driven approach. Web-for-Web allows developing data-intensive web applications in an

object-oriented way. At first sight, the two do not seem to match. However, in some (restricted) way, Web-for-Web supports the Information Modeling step of the Conceptual Design phase.

For the Information Modeling step, we should be able to create a number of Object Chunks that models for an Object Type a number of simple Attributes, which represent a piece of text, a number, a URL ... or/and attributes that refer to other Object Chunks and/or relations with other Object types. Partially, Web-for-Web allows doing this. We are able to define an Object Class that consists of a number of Categories. We attach a number of Attributes to these Categories to specify its content. These Attributes can represent a number, a piece of text, a URL, an email-address or even represent relationships to other Object Classes: the One-to-Many, Many-to-Many and Many-to-One Attributes. In a sense, Web-for-Web allows to define object-oriented Object Chunks.

To make an Audience-driven Web-for-Web, we need to be able to build general Object Chunks, which may contain information about several Object Types instead of one single Object Type. To realize this, we will build a layer around WfW, which will reuse the Object Class concept of WfW and which will allow to capture the Object Classes inside a Component. Next, components can be associated to a Page. In this way, it is possible to group information from different Object Types into one page. These Pages will be put inside a Navigation Track where we will link those using Structural Links. The Navigation Track represents the information needed by an Audience Class. In fact, the Audience-Driven Web-for-Web will not follow or support the different steps of WSDM, but will allow specifying a web application using the results of applying the WSDM method.

# 3 Analysis of the current Web-for-Web

Before I will give a design for the implementation of the Audience-Driven Web-for-Web, I will first give a complete description of the problems that I have encountered while analyzing the current implementation of Web-for-Web. Each of these problems need special attention and an appropriate solution, which I will formulate in this section, in order to come to an optimized version of Web-for-Web, which can then be used for the Audience-Driven version.

## 3.1 Object Classes

The first problem we are dealing with is the concept of an Object Class. This concept is inappropriate for our purpose, because the structure that it defines is not a class on itself, but it defines a view on an Object. So the name 'Object Class' is confusing. In the rest of this thesis, we will not use the term Object Class anymore, but instead we will call it an Object View. An Object View defines a view on an Object in terms of Categories and Attributes grouped inside these Categories, and the designer can specify the contents of the different Categories and Attributes and the order in which they should be displayed. We define an Object View as a view on an Object in terms of categories and the respective Attributes.

But the concept of an Object View is insufficient when we want to come to an Audience-Driven application. It is too limited to serve our purpose, because in an Audience-Driven application information on different objects may be needed on a single page. In the current version of Web-for-Web, there is a one-to-one relation between an Object View and a page where each page contains exactly one Object View. So we will need to be able to group the relevant Object Views for a particular Audience Class. This will be handled by the Audience Class itself, which will act as a container, so that the Object Views can be used to specify the content of the Pages of the Audience Classes. In the design of the application, I will give an overview of the way in which the content of the Object Views can be assigned to a particular Page and the way this will be handled by the Audience-Driven Web-for-Web application.

## 3.2 Objects and Attributes

In the previous section, we mentioned the term Object, but we did not exactly specify what this is. An Object corresponds to an Object Type (OT) in ORM, which has relationships with value types and with other Object Types. In Web-for-Web, these relationships are defined by means of Attributes. So an Object is built up by means of Attributes. There are two types of Attributes: the ones that represent the value types of ORM and the ones that represent the relationships with other Object Types.

We take an example of the Conference Review System (CRS) [18] to demonstrate the Object Types. In figure 3.1, the Object Type Paper is shown with three Attributes: PaperId, Abstract and PaperTitle. These three Attributes are all value types for the Paper Object Type. The PaperId will not be modeled as an explicit Attribute in Web-for-Web, since each Object already has its own

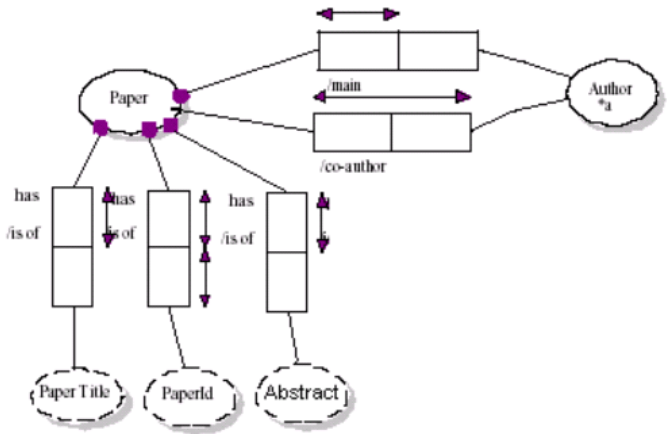



Figure 3.1: ORM Example of CRS

| Paper | |
|---|---|
| Paper Title | Web-for-Web: A Tool for Evolving Data-Driven Web Applications |
| 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.... |
| Main Author | De Greef J. |

| Co-Authors | |
|---|---|
| Co-Author | De Troyer O.<br>Stuer P. |

Figure 3.2: Example Category in Web-for-Web

unique ID. The Object View of this Object will contain two Attributes that represent a value type and two Attributes that present a relationship with another Object Type. An instance of Paper with values for its attributes using Web-for-Web is given in figure 3.2.

There are multiple possibilities to represent a value type. The user has the possibility to choose to add a text-, email-, URL-, date-, image-, file- or number-Attribute to an Object. But this is only a difference in representation; the content of the Attributes is always stored as plain text in the database. A Web-for-Web Object is mapped onto a table of a relational database and it contains an attribute for every value type Attribute that belongs to this Object. An Attribute knows which attribute of which table it represents and so it knows where it has to find its contents. If needed by the application, the contents of the appropriate attribute will be fetched and the XML-code for that Attribute will be generated. Every Attribute has its own XML-code and in this way, it will have its own HTML-representation after converting the XML-code to HTML-code, through the XSLT-files. In these XSLT-files, each Attribute has its own unique representation format. This is because Web-for-Web has opted for an Object-Oriented way of representing the pages to the end-user. Every Object has the same structure: a set of categories consisting of a set of Attributes. For

this purpose, it was easy to define the layout of each part of a page in advance, since it would never change.

In Web-for-Web, it was chosen to display a category as a table with a various number of rows, as can be seen in figure 3.2 where there are two categories: Paper and Co-Authors. The first row of the category indicates the name of the category, followed by the rows that will show the Attributes of this category. The Attributes are presented in two columns: the first column shows the name of the Attribute and the second column shows the value of that Attribute.

As already pointed out, the Object-Oriented structure of the application is inappropriate for our purpose. In addition, (1) we do not want to give every page the same structure and (2) we don't want that every Attribute is displayed inside a table. During page design, it should be possible to choose an Attribute out of the collection of Attributes (as specified for the Object Views) for that page, and put that Attribute somewhere on the page. While doing this, we don't want the Attribute to be shown as two columns with a fixed structure, but we want to specify, in a simple way, how this Attribute should be displayed on the page. We will keep the specification simple because we don't want to be fading out to a tool that is capable of designing a page completely like highly sophisticated WYSIWYG-tools, such as Macromedia Dreamweaver. This is way out of the scope of what we are planning to do and it is unlikely that somebody would be using it when you have the possibility to use other, more sophisticated tools to do the job.

The second type of Attributes, which represent the relations between other Object Types and thus between other Objects of the Web-for-Web application, have the same problem as mentioned above. These Attributes can represent three different relationships: One-to-Many, Many-to-Many and Many-to-One relationship. An example of this type of Attributes can be seen in the Category Co-Authors of figure 3.2. The Co-Author Attribute is a Many-to-One relationship, which links the Paper Object Type to the Author Object Type. When these Attributes are used within an Object, the user only sees a list of the names of the Instances of the Objects that are used by the relationship. In this case only the name of the author is displayed. The user gets a link for each of these Object instances on which he can click and will bring him to a page where he finds the information that is relevant for that Object instance. But this is not what we need in an audience-driven approach. What we want is that all the information needed for those Object instances can be displayed on the same page, without the user having to make another click to get to this information. In this way, we can come to a more audience-driven application, and we can leave the Object-Oriented approach of Web-for-Web behind us.

## 3.3 List of Instances

The next part of the Web-for-Web application that gives problems to fulfill our objective is the list of instances, which is available for every Object as a starting point to browse. In fact, the user has no other possibility than using this list of instances, because otherwise he will not be able to browse through the instances of an Object. The problem with these lists is that this is again too much Object-

Oriented. In the current version of Web-for-Web, there is no possibility at all to change the structure of these lists so that they are more focused towards the audience of the website. The designer can specify a set of Naming Attributes that are used to refer to the instances of the Object. For each instance, these Attributes will be shown, together with the links to view, edit or delete every instance. In the last version of Web-for-Web, there has been added a possibility for the user to sort the lists on the Naming Attributes that are most important for him. This gives the user a little bit more freedom in the ordering of the instances so that the most important instance, from his point of view, will be on top of the page, but still does not fulfill our needs for an Audience-driven application.

What we need is a possibility for the designer to specify which instances he wants from an Object to be displayed on a page. This means that a list of instances should become part of the page and not a stand-alone page, as it is now. The designer should have more control over the lists of instances, which means that, apart from the number of instances of an Object that are displayed on one page, he should also be able to specify in more detail, which instances are to be displayed. For example, the first five instances of a specific Object or the last ten instances of that Object.

# 4 Design of Audience Driven Web-for-Web

In this chapter an overview of the design for the Audience-Driven Web-for-Web application will be given. With this application, we will mainly focus on the Audience Modeling and the Conceptual Design phase of WSDM. We will be able to capture the results of these modeling phases with this application and as a result it will be able to generate a fully functional Audience-Driven web application.

The Audience-Driven Web-for-Web application (AD-WfW) will be implemented on top of the existing Web-for-Web application (WfW). The functionality of AD-WfW will be added as a layer around WfW in which all the functionality of WfW can be used in order to come to an Audience-Driven web application.

The rest of this chapter will deal with the different new concepts that we will introduce in AD-WfW. We will give a complete specification for each of these concepts, so that the reader of this thesis knows what their functionality will be. Furthermore, for each concept, an implementation design will be given, which will consist of two parts: the Database Design and the Class Design. AD-WfW will extensively make use of a relational database, which is an important part of its functionality. Therefore we will give a complete overview of the tables that we will use for the different concepts and the relations between them. With this database design as input, we will be able to define the classes used in the AD-WfW application. AD-WfW will be implemented with an Object-Oriented design in mind, so we will specify all the different classes, and their operations, needed for the correct functioning of the different concepts. We will only give an explanation for the operations that might not be clear through the name it has. Also, operations that were already explained in previous classes and that do not have another meaning, will not be explained. Even so for the attributes of some tables of the database. The concepts that we are about to discuss in this chapter are 'Audience Class', 'Component', 'Page' and 'Navigation Track'.

## 4.1 Audience Classes

In this section, we will give a full specification of the concept Audience Class, the way it will be used in AD-WfW and an overview of the implementation of the Audience Classes. We have already mentioned the concept of an Audience Class in the overview of WSDM, but in this section we will give a more detailed explanation of this concept and how it will be incorporated in AD-WfW.

### 4.1.1 Specification

We will have two kinds of Audience Classes in the AD-WfW application: the regular Audience Class and the Audience Subclass. The difference between them is merely situated in the fact that the Audience Subclass inherits all the properties of its Audience Super Class. As we have seen in the WSDM overview, this includes the information requirements, and the functional requirements, but this is not the only thing since also the functionality of the superclass will be inherited. This means that the subclass has access to all the resources of the

superclass: Components, Pages ...

We are not building a tool that is able to completely support the modeling phases of WSDM, but a tool that is able to generate a web application that corresponds to the web application as modeled by WSDM. Therefore, we will not force the user to enter all the requirements for an Audience Class (according to WSDM) in AD-WfW. Instead, we will give the designer the opportunity to add some of the most important requirements, so that he can inspect these requirements when he is building the web application. We will use the WfW One-to-Many Attributes in order to relate the Audience Classes with its requirements. A requirement is represented by a WfW Object View that contains only one text-Attribute. Two Object Views are needed to model these requirements: an 'Information Requirement' and a 'Functional Requirement'. All the requirements from a superclass will be inherited by the subclass. Note that they do not reflect any functionality of the AD-WfW application. They are only included for documentation and to enhance the usability of the application. They are also a first example of the interaction between WfW and AD-WfW.

An Audience Class itself will act as a kind of container in which all the information about an Audience Class will be stored. This container can later on be used to generate the complete web application: page layout, navigation through the pages ... During the Navigation Design phase, a Navigation Track for each Audience Class is made. In AD-WfW we will be able to describe such a Navigation Track and it will be stored inside the container of the Audience Class. How we can describe a Navigation Track will be discussed later. The input to describe these Navigation Tracks will be the Object Views that are associated to the Audience Class. These Object Views will define the content of the different Pages, which will be responsible for the presentation of the contents of the Audience Class.

### 4.1.2 Implementation Design

First of all, we need some space in a database to save all the information that belongs to the Audience Classes. For this purpose we will use the existing Meta Database of WfW. The new tables will be an extension to the already existing tables in the Meta Database. An overview of the relational database tables we will use for the Audience Classes is given in figure 4.1. After discussing these tables we will give an overview of how the Audience Classes are actually implemented in AD-WfW.

#### Database Design

Figure 4.1 gives us an overview of the tables in the Meta Database that we will use for the implementation of the Audience Classes in AD-WfW.

The ***AudienceClasses*** table will contain a row for each Audience Class that is defined for the web application. The attributes that are defined for this table are:

- *tmp:* This is an attribute solely used to ease the implementation. This attribute is used to indicate that not all information about the Audience

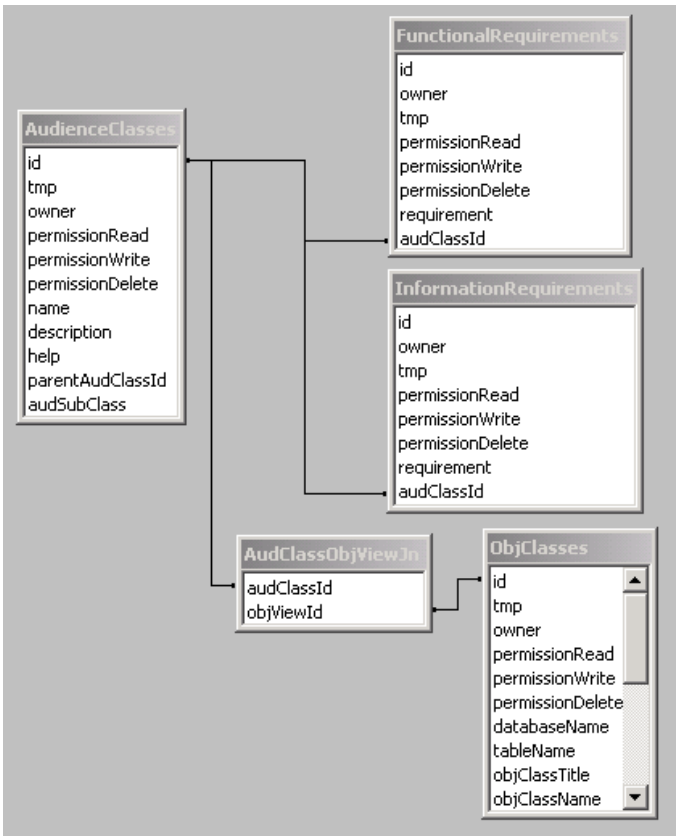



Figure 4.1: "Audience Class" database Diagram

Class instance has already been given. If the value is "1", this indicates that the instance is "temporary". AD-WfW will only show the instances that are not temporary. This is an attribute that is used in all following tables, such as *NavigationTracks*, *Pages*, *StructuralLinks* ... so we will not repeat the meaning of this attribute anymore;

- *owner:* this attribute will contain the id of the owner of the Audience Class. The owner of an instance has the ability to set the permissions for that particular instance. This means that he has full access to the instance. The owner-attribute itself is a foreign key that is related with the id of the persons table in the Database. Again, its functionality is the same for all the tables that contain an owner-attribute;

- *permissionRead:* this attribute will contain the IDs of the users or the groups of user that have permission to read the content of an instance of the Audience Class. It will again be used across the AD-WfW application to ensure the security of the application, even so for the permissionWrite and permissionDelete attributes;

- *permissionWrite:* this attribute will contain the IDs of the users or the groups of users that have permission to write and update the content of one of the instances;

- *permissionDelete:* this attribute will contain the IDs of the users or the groups of users that have permission to delete one of the instances;

- *name:* each Audience Class has a name, according to the different types of visitor the AD-WfW web application will be able to handle;

- *description:* a short summary of the purpose of the Audience Class will be stored in this attribute;

- *help:* the designer may display some help text at the bottom of the screen in order to give some guidance to anyone who may work with AD-WfW;

- *audSubClass:* this will be used as a kind of indicator-attribute with which we express whether the Audience Class is a subclass or not, denoted by an integer value which will represent its type: one for subclass and zero when no subclass;

- *parentAudClassId:* when an Audience Class is a subclass, this attribute will be important. It represents a foreign key to the id of the *AudienceClasses* table. AD-WfW will need this id to be able to fetch the properties and functionality of the superclass in order to generate the correct content of the Audience Subclass.

The Audience Classes have a number of related Object Views that are defined by the WfW application. They are stored in the existing ***ObjClasses*** table. Since we have to make a connection between the Audience Classes and the Object Views, we have introduced a new join-table in the Meta Database, called the ***AudClassObjViewJn*** table, which consists of two attributes:

- *audClassId:* a foreign key that is related to the id of the *AudienceClasses* table;

- *objViewId:* a foreign key that is related to the id of the *ObjClasses* table.

For each association between an Object View and an Audience Class, a row will be stored in this table with the id of the Object View and the id of the Audience Class.

Additionally, we have added two new tables ***InformationRequirements*** and ***FunctionalRequirements***. They have two common attributes that will be used to store its information:

- *requirement:* this attribute will be used to store the text of the requirements that will be specified for the particular Audience Class. For each requirement, a new row will be added to this table;
- *audClassId:* a foreign key related to the id of the *AudienceClasses* table. It will store the id of the Audience Class to which the requirement is defined.

In WfW these tables should have been put in the Database and not in the Meta Database, since they are defined as an Object View. In AD-WfW this is not the case since the requirements are part of the Meta data of the web application. They are used for modeling the user interface of the web application and are not part of its content.

Now that we have a structure for the Meta Database, we can give an overview of the implementation of these Audience Classes.

### Class Design

Figure 4.2 gives an overview of the classes that will be used to implement the Audience Classes in AD-WfW. The attributes of the classes are not shown in the figure since these would make the drawings too complicated. Important attributes that need some special attention will be mentioned in the specifications of the classes.

The ***AudienceClass*** class will be used as a kind of container from which we can fetch the Attributes, Categories and Object Views that will be needed to generate the corresponding Pages for that particular Audience Class. Its functionality includes:

- *AudienceClass(iAudId:int):* this is the constructor. The id of the Audience Class is given as input and according to this id, the properties of the appropriate Audience Class are fetched from the database. These include some general parameters as the name of the Audience Class, the help text and the description. Furthermore, the values of the audSubClass and the parentAudClassId attributes will be fetched;
- *isSubClass():* This function will check the value of the audSubClass attribute and return TRUE when the Audience Class is a subclass;
- *metaToXML():* this function will generate the correct XML code for the edit Page of the Audience Classes in the Meta part of AD-WfW. The XML code will consist of the name, the description and the help-text for

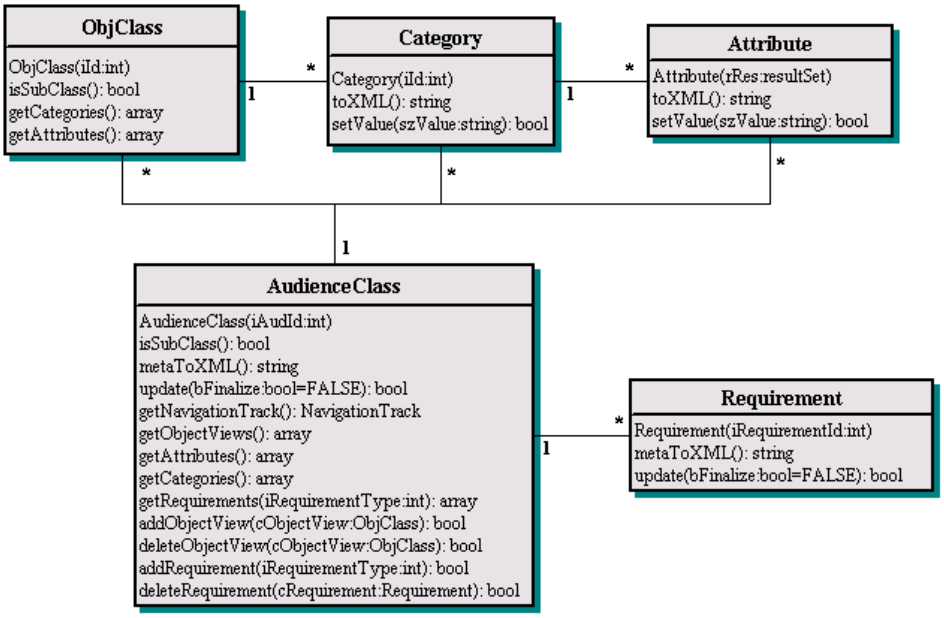



Figure 4.2: "Audience Class" Class Diagram

the Audience Class, a list of the Information and Functional Requirements, a link to its Navigation Track, a list of all the associated Object Views and a list of all the Object Views that can be associated. The XML document will be converted to HTML code through the ADMetaAudienceClass.xsl file in which a template will be defined for the XML tags used in the XML document. For example for the <audienceClass>-tag (indicating the content of an Audience Class), the <name>-tag (indicating the name of the Audience Class), the <lstObjectViews>-tag (indicating a list of all the associated Object Views) and many more. There is no presentation needed for the Data part, since this part will be handled according to the specification of the Navigation Track;

- *update(bFinalize:bool=FALSE):* This function will be called at the Meta-level and will write the content of the Page of the Audience Class to the Meta Database. When bFinalize is set to TRUE, the tmp-value will be set to zero, which indicates that the specification for the Audience Class is finished. This function will be used in the following classes and will have the same functionality;

- *getNavigationTrack():* each Audience Class should have one Navigation Track which will be used to generate the navigational path through the Pages for the Audience Class. This function will fetch that Navigation Track from the database;

- *getObjectViews():* this function will return an array of all the Object Views that are associated to the Audience Class, including those of the superclass;

- *getAttributes():* this function will be called from another object, for example the Component, in order to find all the Attributes that are available for the Audience Class. A request is sent to all the Object Views to return the associated Attributes;
- *getRequirements(iRequirementType:int):* an array of all the Requirements of a given type (iRequirementType) will be returned;
- *addObjectView(cObjectView:ObjClass):* and similar the *deleteObjView()* function will respectively add and delete an association between an Audience Class and an Object View. The *AudClassObjViewJn* table is updated with a new row with the id of the Audience Class and the id of the Object View;
- *addRequirement(iRequirementType:int):* a new Requirement of the type iRequirementType will be added to the Audience Class. The table of the Requirement will be updated with a new row. Together with this function we can also mention the *deleteRequirement()* function which will delete a certain Requirement from the Audience Class.

We also have added a ***Requirement*** class that takes the responsibility over a certain Requirement of the Audience Class. It has the following functionality:

- *Requirement(iRequirementId:int):* the constructor will fetch the requirement from the database and store it in a local variable, for further use in the generation of the XML document;
- *metaToXML():* an XML document will be generated for the specification of the Requirement. This will only contain a form field for the text of the Requirement.

## 4.2 Components

Components are used during *Navigation Design* in WSDM. A component will be one of the most important building blocks in AD-WfW. It will also be the connection between AD-WfW and the old WfW application. There are three types of Components that will be incorporated in AD-WfW: *Information Component*, *Mixed Component* and *External Component*. We will now give a specification for each of them.

### 4.2.1 Information Component

The first Component we are about to discuss is the Information Component. WSDM states that the content of this Component can be described by a simple image-, text-, video-chunk, called Simple Chunks, or by a composed Object Chunk which may be a collection of one or more Simple Chunks. When we bring this in relation to WfW, we can see that the concepts of Object Chunk and Simple Chunk are already available: a Simple Chunk corresponds with an Attribute (containing a small piece of information like an image, a piece of text ...) and a composed Object Chunk corresponds with a Category (a collection of Attributes). With AD-WfW we want to provide a way to fetch the contents of the Attributes and Categories and present it to the visitor of the web application

in a suitable presentation. The easiest way is to use the functionality of WfW to fetch the content of the Attributes and Categories. Since we do not want to touch the implementation of WfW, we will have to provide a layer around them, which will use the functionality of WfW to fetch the contents and convert the content to a new presentation according to the presentation specifications, which will be discussed in section 4.2.4.

The Information Components can be associated to an Attribute or a Category, which means that we will have two types of Information Components: the *Simple Information Component* that will use an Attribute to generate its content and the *Combined Information Component* that will use a Category to generate its content. The Information Components will be defined for a particular Audience Class to which a number of Object Views are associated. The Object Views will provide the Attribute and the Categories for the Simple and Combined Information Components. The user will be able to select one of them of which the content will be displayed on the Pages. They will perform the database operations

There is still one problem with the Information Components. Each Object View corresponds to a particular table in the Database of the web application and each Attribute associated with that Object View will be able to fetch the content of a particular attribute of that table. But since multiple instances of one Object View can be defined, we need to be able to fetch the content of the correct instance. This problem will be handled by the Pages on which the Components will be placed. Each Page will know the instances of the Object Views and according to these, the correct content for the Attributes can be fetched and displayed.

### 4.2.2 Mixed Component

The Mixed Component consists of a number of Information Components and a number of other Mixed Components. Where an Information Component is only able to take one Attribute or Category, the Mixed Components will be able to group a number of Attributes and Categories through the concept of the Information Components.

The designer will have to choose a particular presentation for the Mixed Components, as will be specified in section 4.2.4. The concept of the Mixed Component can in fact be seen as a kind of container in which all its elements, Components in this case, will have the same presentation specifications.

Again we could have the problem of fetching the content that is specified for this Component. But since the Mixed Component will take the form of a container that, in the end, will consist of a number of Information Components (since each Mixed Component will have to contain at least one Information Component, otherwise its specification is invalid), we just have to send a request to the Information Components which will return its content according to the presentation specifications set by the top Mixed Component.

### 4.2.3 External Component

In WSDM, an External Component is defined as a Component that represents an external application. Since it is impossible to include the complete external application into our web application, the External Component will only provide a link towards the external application. In fact, this is almost the same as a URL-Attribute in WfW, but there is a small difference. A URL-Attribute should belong to a particular Object View and this is not the case for an External Component. We want the External Component to be a standalone concept of AD-WfW, independent of changes to the Object Views.

Therefore, the designer will have to specify an URL that points to the external application and a name for this URL. The name is optional; it just gives the designer the opportunity to specify a name that says more to the visitor than only the plain URL. When no name is given, the URL itself will be shown to the visitor.

### 4.2.4 Presentation Specification

We have already referred to the presentation specifications in the sections about the different types of Components. Here, we will actually discuss how the Components can be presented in AD-WfW.

The designer will have the possibility to specify two different formats for the Components. The first format deals with the number of columns that will be used to display the information: one or two columns. When the information should be displayed in one column, only the content of the Component will be shown. When the two columns option is chosen, the first column will display the name of the Component, and the second column will display the content that is associated with that particular Component.

The second format deals with a header that is used for the Components. This means that the designer has the possibility to specify whether a header will be used (which will contain the name of the Component) or not. We can compare a header to the name of the Category that is shown in the old WfW application. It is just a simple field with a different color that will be used to specify a name for the grouped information. This will be used to give a better overview on the information on the pages, so that the visitor will find more easily what he is looking for.

The presentation part is very limited. This is because we are designing a tool in which the focus is on the Audience-Driven organization of the information and not on the fancy representation of this information. For more advanced representations, we would recommend more specialized design tools like Macromedia Dreamweaver (www.macromedia.com) or other advanced HTML tools. The limited presentation specifications that the developer is able to make in AD-WfW should be enough to provide a decent presentation of the information, in an Audience-Driven way.

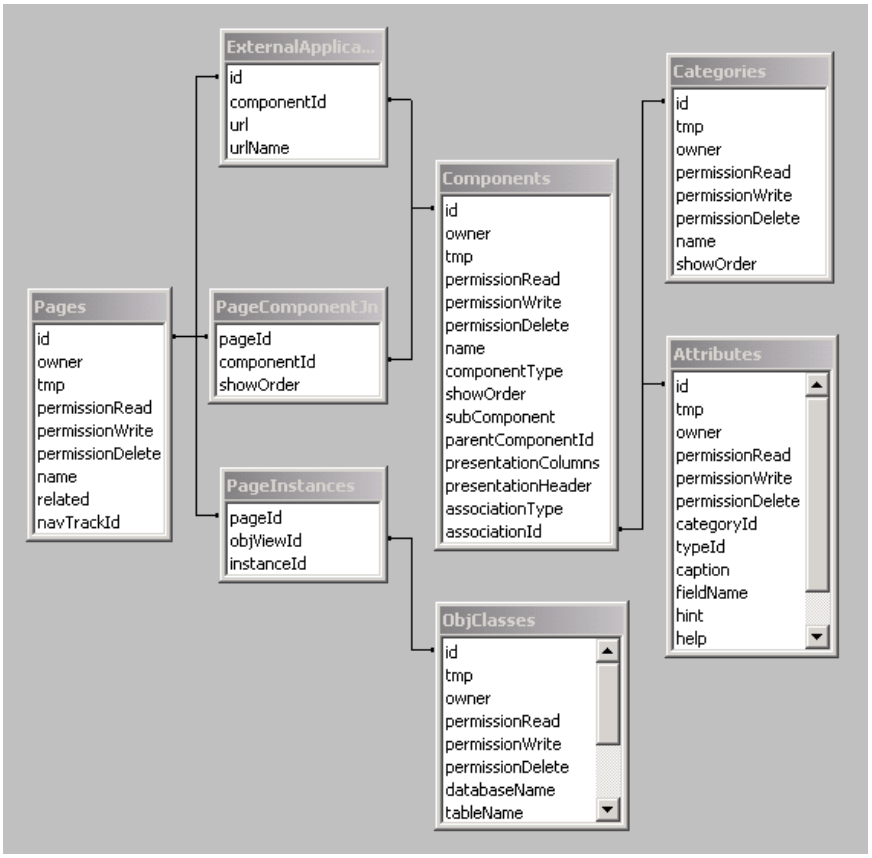



Figure 4.3: "Component" database Diagram

### 4.2.5 Implementation Design

#### Database Design

Figure 4.3 gives an overview of the extensions that will be made to the Meta Database to include Components. We will discuss the role of the new Components table in the database.

The database diagram for the Components is given in 4.3. First of all we can see that there are next to the *Components* table two other new tables introduced, the *Pages* and the *PageComponentJn* tables, which will be discussed in detail in the following section; they are introduced here to show the connection between the Components tables and the Pages tables.

When we look to the ***Components*** table, we can see the following attributes:

- *componentType:* this attribute will contain an integer value that indicates which type of Component is stored. We can have an Information-, an External- or a Mixed Component;

- *showOrder:* a Mixed Component will be created as a group of Components. We should have the possibility to state the order in which the sub Components are shown within the Mixed Component. The showOrder attribute will contain an integer that indicates the position of the subcomponent;

- *subComponent:* this will be an indicator-attribute that states whether the Component is a subcomponent of a Mixed Component or not. Its value will be set to TRUE when this is the case. This will be very important for the presentation specifications, since according to the value of this attribute the Component will use the presentation of its supercomponent or not;

- *parentComponentId:* according to the value of the subComponent attribute, there will be an id in the parentComponentId attribute. This is a foreign key to the id of the Components table. It is needed to get up or down into the hierarchy and fetch all the Components that are related to the top Mixed Component;

- *presentationColumns:* contains the number of columns that will be used for the presentation of a particular Component. This is an integer value that can be one or two (stating the number of columns that will be used), according to the preferences set by the designer;

- *presentationHeader:* this will be a kind of indicator-attribute that contains a boolean value that states whether a header should be shown for the presentation of the Component or not;

- *associationType:* when we have an Information Component, this attribute will store the type of Information Component: Simple or Combined;

- *associationId:* for an Information Component, this attribute will be a foreign key to the id of the *Attributes* or *Categories* table, according to the type of Information Component.

Furthermore we have the ***ExternalApplications*** table. This table is not part of the Meta Database, but it will be stored in the Database. The reason for this is that it will contain the URL's for the External Components, as they will be shown on the Pages. The content of the Pages is stored inside the Database, so this table will also be part of the Database. Its attributes are:

- *componentId:* this is a foreign key to the id of the *Components* table. Next to the pageId, we also need the componentId to be able to uniquely identify the URL, since it could be possible that we include multiple External Components on one Page;

- *url:* this field will contain the URL to the external application;

- *urlName:* this field will contain an optional value for the name the designer can give to the URL to mask the actual URL to the visitors.

As a last addition, we have the ***PageInstances*** table. We already mentioned that the page is responsible for providing the correct instances from

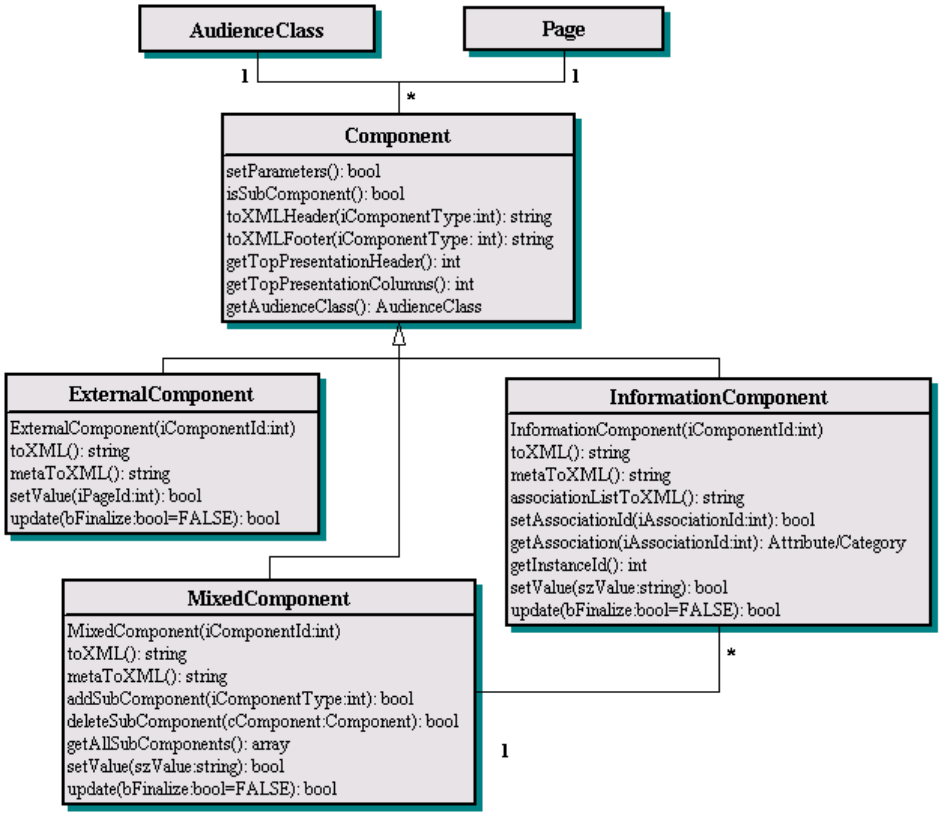



Figure 4.4: "Component" Class Diagram

which the Information Components will have to fetch their content. We mention this table in this section because the Components are the structures that will mostly use this table. We have an Audience Class to which a number of Object Views are associated. Each of these Object Views is related to a particular table in the Database. When we add a new Page to the Navigation Track of an Audience Class, we will create a new instance in the table to which the Object View is related. The id of the new instance will be stored in the *PageInstances* table, together with the id of the new Page and the id of the Object View. The content of an Information Component consists of either a Category or an Attribute, both being part of a particular Object View. Having the id of the Object View (*objViewId*) and the id of the Page (*pageId*) to which the Component belongs, the id of the instance (*instanceId*) of the table in the Database can be found. According to that id, we can fetch the contents of the Information Component through the functionality of WfW.

### Class Design

Each of the component classes is a subclass of the top ***Component*** class that contains some functions that are available to each of the subclasses:

- *setParameters():* this function will fetch the presentation specifications

for the Component which will be stored in a public variable inside the Component class, so that they are available for each of its subclasses. This function will be called from within the constructor of each of the subclasses;

- *toXMLHeader(szComponentType:string):* The header of each type of Component is the same, except for the begin tag. This means that we will have a <externalComponent>, a <informationComponent> and a <mixedComponent> tag. This function will return the correct XML code for the header, including the presentation specifications. For the presentation, we will use the <presentationHeader> and <presenationColumns> tags;

- *toXMLFooter(szComponentType:string):* the header should also be closed, this can be done by this function, which will return the correct XML code for it;

- *getTopPresentationHeader():* this function will be called by the Information Components and the Mixed Components to generate the XML code for the correct presentation. It will be used by the Components that are defined as subcomponents of a Mixed Component. The *getTopPresentationColumns()* function has similar functionality;

- *getAudienceClass():* this function will fetch the Audience Class for which the Component is defined. This Audience Class is important since it contains functions that can return a list of all possible Attributes and Categories to which the Components can be associated.

The ***ExternalComponent*** class is the one with the least functionality of the three. It consists of:

- *toXML():* this function will generate the correct XML code for the External Component, according to its presentation specifications. We will need a new XML tag for it, the <externalComponent> tag. This tag consists of the <url> tag and the <urlName> tag to be able to generate HTML code for the link. The link will be displayed according to the presentation specifications, returned by the *toXMLHeader()* function;

- *metaToXML():* the specification of the structure of a Component is done at the Meta part of the AD-WfW application. The designer will only be able to specify the number of columns and the specification for the header. The URL and the name for the URL are specified at the Data part of the web application;

- *setValue(iPageId:int):* This function will be called from the Data part of the web application. The user has edited the content of the Page on which the External Component was placed. The *setValue()* function will fetch its value from the form fields on the Page and will update the *ExternalApplications* table of the Database.

As we can see, the ***InformationComponent*** class does also have the *toXML()* function. One could argue that it would be better to put this function in the top Component class, but this would be inappropriate since each of the subclasses will have its own representation. The *InformationComponent* class has the following functionality:

- *toXML():* this function will have the responsibility to fetch the content of the Component through the Attributes and Categories and convert the XML code to an XML document that is suitable for the specification of the Information Components. Since each Attribute of WfW has its own XML tags, we will also need these XML tags for the Information Components, like the <infoCompText>, the <infoCompNumber> tag  Each of these Information Components will have its own presentation which we will specify in the XSLT documents. When we are about to edit the content of a particular Page, we will use another XSLT document in which we will have templates provided for each of the Information Components, but instead of simply displaying the content of the Component, a form field will be generated so that the user can update its content;

- *metaToXML():* the XML code that will be returned by this function will include a form in which the user has the possibility to change the Attribute or Category to which the Information Component will be related. The designer will also have the possibility to change the number of columns and the specification of the presentation of the header;

- *associationListToXML():* this function will return the XML code for a list with all the names of all the possible Attributes or Categories, according to the type of Information Component (Simple or Combined). For the list a complete new XML tag will be included: the <associationList> tag, which consists of a number of <association> tags containing the names and the IDs of the associations;

- *setAssocationId(iAssociationId:int):* the designer will choose one of the possible associations (as given by the *assocationListToXML()* function) and then the *setAssociationId()* function is called, which will update the assocationId attribute of the *Components* table;

- *getAssociation(iAssociationId:int):* this function will return an object of the Attribute or the Category to which the Information Component is associated. This object will be used for both updating and displaying the content of the Information Component;

- *getInstanceId():* this function will return the id of the instance of the table in the Database from which we will fetch the content of the Information Component;

- *setValue(szValue:string):* in the Data part of the web application, the user will have the possibility to update the content of a particular Page. That Page will mainly consist of Information Components of which the content has been fetched through the Attributes of WfW. So the Information Component will have to make use of the Attributes in order to be able to update its content. The Page will fetch the content of the form field for a particular Information Component and call the *setValue()* function of the Information Component with the corresponding form field value. According to this value, the database will be updated.

The ***MixedComponent*** class is the last of the different types of Components. Its functionality includes:

- *toXML():* this function will send a request to all the Components that are defined as one of its subcomponents. Each of these subcomponents will return its own XML code, which will be combined into one large document that is returned;
- *metaToXML():* the designer will see a list of all the sub components that are defined for the Mixed Component. He will get the possibility to delete or add a Component to this list. The functionality of this function is the same as for the Information and External Components;
- *setValue():* this function will loop over all the subcomponents and send the *setValue()* command to each of them, so that the content of each of the subcomponents is updated with the value of the corresponding form field.

## 4.3 Pages

The Page concept in the AD-WfW application will be used to group Components onto separate physical pages. Each Page will consist of a number of Components. In this section we will give the specification of these Pages and the way they can be defined.

### 4.3.1 Specification

The purpose of Pages in AD-WfW is to group components. In WSDM, this is done by first creating Task Models and Navigation Tracks. We will not consider Task Models in AD-WfW, since this would complicate the tool too much and it was never the intention to make a tool that supports the design steps of WSDM completely. Rather, AD-WfW should support to build a website based on the output of WSDM. Therefore, the Navigation Tracks will be directly expressed in term of Pages.

As the presentation design is not yet supported by WSDM, the layout of the Pages will be more or less fixed. In AD-WfW, a Page will consist of two major parts: the Navigation Part and the Components Part. The Navigation Part will be a small part of the page in which the navigation to other pages will be displayed (e.g. the structural links in the Navigation Track). In order to give the designer of the web application some freedom about this part of the Page, he will be able to modify the XSLT documents in order to define another template for the Pages. This means that, for some web applications, the Navigation Part can be placed on top of the Page and in other web applications the Navigation Part will be placed in a menu on the left side of the page. In this way, the designers will have enough freedom to give their Audience-Driven web application a personal touch.

The Components Part will contain the actual content of the different Pages. This will be the place where the Components that are defined for a particular Audience Class will be grouped. Each page will consist of at least one Component. However, there is still one problem with the Pages. In section 3.3 we stated that it should be possible to include a list of instances on a particular Page. In AD-WfW we do not have anything yet that is able to support this.

What we want to create is a type of Page of which we can create multiple instances, which means that we would have multiple Pages with the same layout, but with another content.

For this, we use a new concept: the *Page-List*. A Page-List corresponds in roughly to the List of Instances known in WfW. The only difference with these lists is that the designer will have more control over them. The Page-List concept can be placed on a page in the same way as the Components are placed on the pages. But the Page-List will need more parameters where the most important one is the related Page of the Page-List. This means that the related page will only be accessible through the links that are provided by the Page-List structure and that we are able to define multiple instances for it. As in WfW, we will also have to specify the Naming Components (Naming Attributes) that need to be displayed in the list and we will also be able to choose one of the chunks of the related page as a parameter for the default ordering of the instances. Next to this, the number of instances that we want to show in the list should be specified. Since it could be possible that there are hundreds of instances, it would be very inappropriate to display them all at once on a single page. Therefore, we give the opportunity to the designer to limit number of instances and AD-WfW will automatically provide links to reach the rest of the list of instances.

As in WfW, editing information is mixed with viewing the information. We realize that this is a restriction and not completely according to the Audience-driven approach of WSDM, but with the current architecture of WfW it was not possible to change this. When the user is editing a particular Page, a form will be displayed for that page in which he can enter the information for the different Components that were placed on the Page. For each of these Components, a form field will be shown in which the user can specify the correct content. This could be a piece of text, a number, an email-address ... according to the type of information that is about to be stored. A Page can also contain a Page-List structure for which the designer will be able to add some new Page Instances. The display of the forms will be handled by the XSLT document. The XML code that will be returned by AD-WfW will be the same as for the pages that display the information. But the XSLT parser will receive a different XSLT document as input, so that the XML-code will be parsed and displayed in a completely different way. This is a perfect example of the way in which we are taking advantage of the Three-Tier Architecture. The presentation of the information is separated from the generation of the information so that we are able to generate completely different presentations of the Pages with the same XML documents.

### 4.3.2 Implementation Design

#### Database Design

The diagram of the Meta Database for the Page structure is shown in figure 4.5. We see many tables that are also present in the database diagram for the Components.

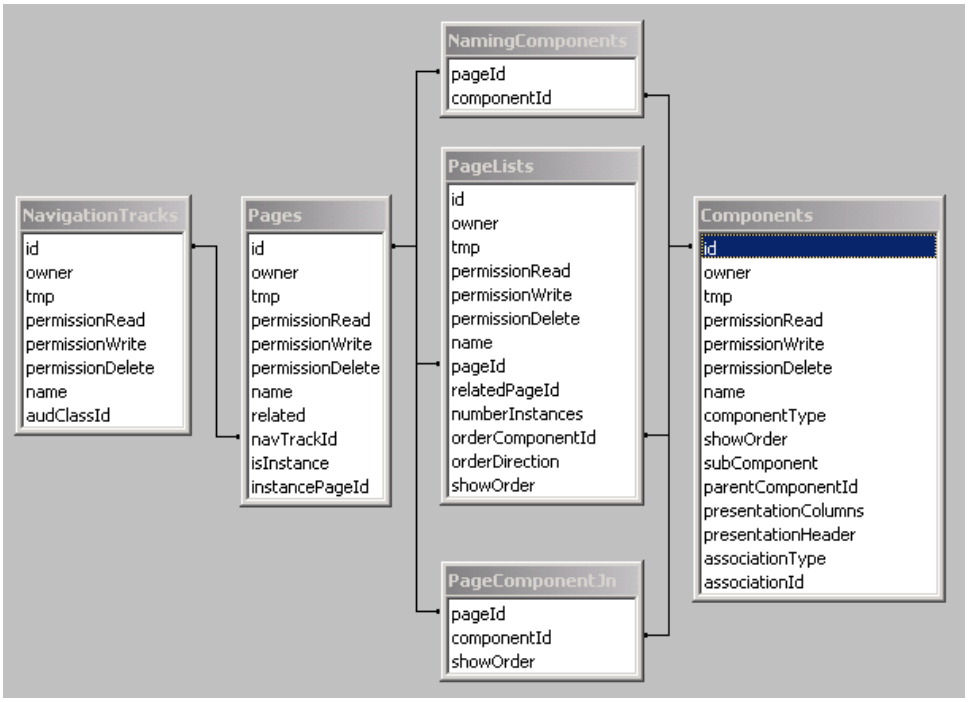



Figure 4.5: "Page" database Diagram

The ***PageComponentJn*** table will be used to provide a link between the Pages and the Components. For each Component that belongs to a particular Page, we will add a row to this table, containing a value for the attributes:

- *pageId:* this is a foreign key related to the id of the *Pages* table. With this attribute, we are able to fetch all the Components that are placed on a particular Page;
- *componentId:* this is a foreign key related to the id of one of the *Components* tables. The exact table will be determined in the implementation of the Pages, according to the value in the componentType attribute;
- *showOrder:* this attribute will contain an integer value by which the Components will have to be ordered on the Page. The Component with the highest integer value for a particular Page, will appear on top of the Page.

The ***Pages*** table also contains a number of attributes that might need some explanation:

- *related:* the value that will be stored in this attribute has a lot to do with the concept of a Page-List. As we have mentioned in the specification, a Page-List will be related to a particular Page, and according to this relation, the related Page will not be accessible through the navigation specified by the Navigation Track, since it will be accessible through the concept of the Page-List. The related attribute will contain a value TRUE when the Page is related to a Page-List. By specifying this attribute, we

are able to easily fetch all the Pages that can be used within a Navigation Track;

- *navTrackId:* this attribute is a foreign key to the id of the *Navigation-Tracks* table. Since each Page will be related to only one Navigation Track, the id of this Navigation Track can be stored inside the Pages table;

- *isInstance:* some of the Pages will be related to a Page-List structure. We will be able to define a number of instances for these Pages, where the instances will all have the same layout;

- *instancePageId:* this is a foreign key to the id of the *Pages* table. It contains the id of the Page of which it is an instance. The layout of that Page will be fetched to specify its layout.

We have already mentioned that the related attribute of the *Pages* table has something to do with the Page-Lists, so now it is time to get deeper into the ***PageLists*** table itself :

- *pageId:* this attribute is a foreign key that is related to the id of the *Pages* table. This attribute will indicate to which Page the Page-List belongs;

- *relatedPageId:* this attribute is also a foreign key to the id of the *Pages* table. But instead of indicating to which Page the Page-List belongs, this foreign key will contain the id of the related Page. The related Page will be the Page of which the instances will be shown in the Page-List;

- *numberInstances:* this attribute will contain an integer that indicates how many instances of the related Page may be shown in the list. When the number of instances of the Page is bigger than the number of instances to be shown, AD-WfW will automatically generate the appropriate links that will lead to the rest of the instances;

- *orderComponentId:* this attribute is a foreign key to the id of the *Components* table. Each Page will have a couple of Naming Components that are stored in the *NamingComponents* table. The user will have the possibility to select one of these Naming Components to be the default Component on which the list of instances will be ordered;

- *orderDirection:* this is the default direction on which the list of instances will be ordered, according to the default Naming Component, saved in the orderComponentId attribute. This direction can be ascending or descending, based on the preferences of the designer;

- *showOrder:* as with the different Components for a particular Page, we will also have to specify the place that the Page-List will take on that Page. This attribute will contain an integer value that will indicate its position.

As a last addition, the ***NamingComponents*** table will store the Naming Components that we are able to define for a particular Page. We have something similar in WfW, namely the Naming Attributes (Attributes that have been chosen to be able to uniquely identify the instances of a particular Object View).

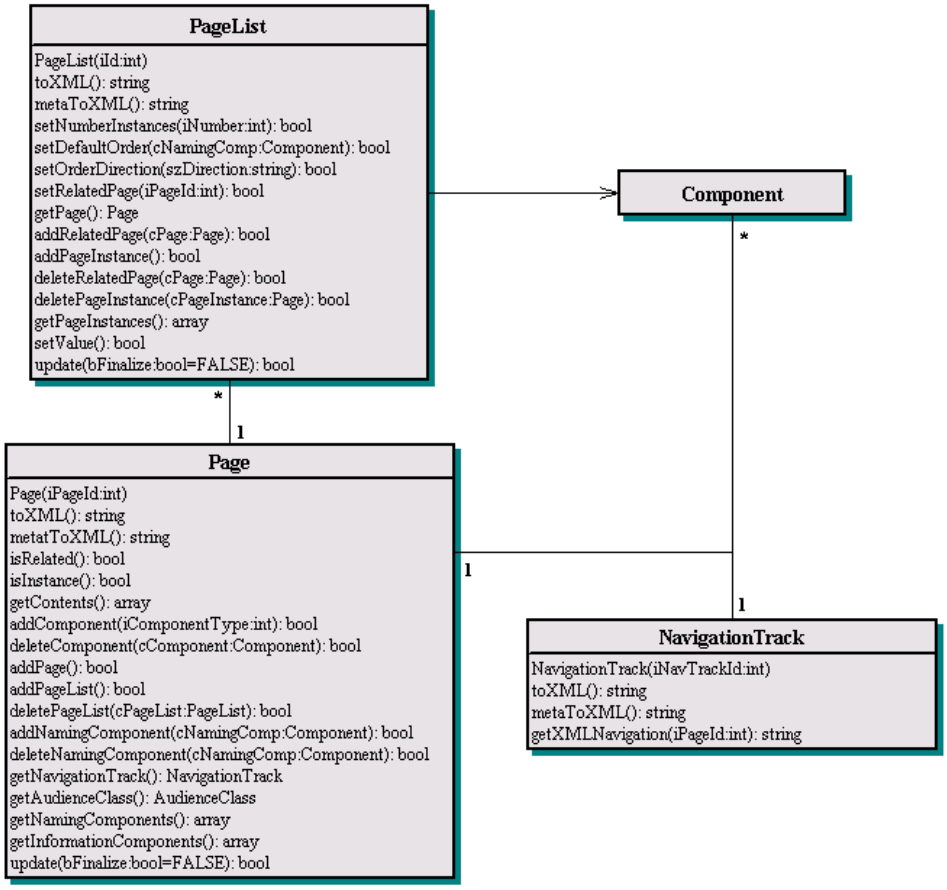



Figure 4.6: "Page" Class Diagram

The idea of the Naming Components in AD-WfW is the same. For this purpose, only the Information Components can be used since they will be the only Components that will display a small amount of information. When we would allow using a Mixed Component as a Naming Component, we would give the opportunity to display too much information in the list of instances and the list would no longer be clearly structured. The attributes in the *NamingComponents* table are:

- *pageId:* this attribute will be a foreign key to the id of the *Pages* table. According to this id, we will be able to fetch all the Naming Components that are defined for a particular Page.
- *componentId:* this attribute will be a foreign key to the id of the *Components* table. This id will store the id of the Information Component that is set to be a Naming Component.

### Class Design

The class diagram of the Page structure is given in figure 4.6. Two new classes will be needed for the implementation of the Pages. We will start with the explanation of the ***Page*** class:

- *toXML():* the XML code generated by this function will be used for the presentation of viewing the pages, as well as for editing the pages. On the edit Page, the user gets a page with a number of form-fields in which he may specify the contents of the Page. The difference between the two presentations will be handled by the XSLT document. The XML document that is returned will be the same for both, but the XSLT parser will get another XSLT document as input, in order to be able to generate a completely different output. We will certainly need some new tags to be able to display the pages correctly. The first new tag that we will introduce is the <page> tag that will indicate the start and the end of a Page. Within this tag we have the <pageNavigation> and <pageContent> tags. The <pageNavigation> tag will contain the content of the navigation part of the Page. To fetch the XML code that corresponds with the navigation part, we will need the Navigation Track. We will fetch the correct Navigation Track through the *getNavigationTrack()* function and send a request to it with the *getXMLNavigation()* function. The result is a series of <link> tags with the name and the id of the Pages the visitor is able to reach, starting from that point;

- *metaToXML():* where the *toXML()* function handles the presentation of the Page in the Data part of the web application, the *metaToXML()* function will take care of the presentation of a particular Page in the Meta part of the web application. A list of all the Components and Page-Lists for that Page will be shown to the designer, together with their order on the Page;

- *getContents():* this function will be called by *toXML()* in order to fetch the XML contents for the <pageContents> tag. The result value of this function will be an array with all Components and PageList structures, ordered according to their position on the Page. Each of these objects has its own implementation for the *toXML()* function so that the *toXML()* function of the Page can loop over all the objects in the array and combine the resulting XML code into one complete document;

- *addPage():* this is one of the most important functions of the Page class. It will update the database according to the information that the user has provided on the forms. Since each of the Pages will have another structure, we do not know in advance how many form fields will be provided. The technique we will use is one in which we number all the form fields, according to the number of Components that we have placed on that particular Page. Since the structure of a particular Page will not change when we are updating its content, we can fetch the same structure again when we are about to update its content. For the update, the *addPage()* function will fetch all the Components that are placed on the Page and put them in an ordered array, according to their positions on the Page.

The place of the Components in the array will correspond with the number of the form field on the Page, so we can update the content of the database for the Components, according to the value that was provided in the corresponding form field;

- *getNavigationTrack():* this function will return an object of the Navigation Track to which the Page belongs;
- *getAudienceClass():* this function will return an object of the Audience Class to which the Page belongs;
- *getNamingComponents():* this function will return an array with objects of all the Naming Components that are specified for that particular Page. This function will be used by the Page-List structure in order to be able to generate the correct list of instances;
- *getInformationComponents():* this function will return an array with objects of all the Information Components that are defined for a particular Page. These are needed to generate a selection list of all the possible Naming Components for that Page.

Next to the Page class we have the ***PageList*** class that is responsible for the functioning of the Page-List concept. The following functionality is supported:

- *toXML():* this function will return the XML code that is needed to display the Page-List structure. We will need a new XML tag for it, namely the <pageList> tag, and within this tag we will use a number of other tags like the <namingComponent> tags, which will all hold a piece of information that is stored inside the Naming Component, so that we are able to build up the correct list of instances. For each instance of the related Page, a link will be provided so that we can reach the instance the appropriate XML tags will be included to reach the rest of the list of instances when the number of instances was too small to show them all. The <next> and <previous> tags are included to support;
- *metaToXML():* the return value of this function is the XML code that is needed to display the information that is needed to be able to define a Page-List in the Meta part of AD-WfW. The related Page will be shown and the designer will be able to specify the number of instances, the default Component on which the list will be ordered and the direction of the ordering of the Page-List;
- *setNumberInstances(iNumber:int):* this function will be called when the designer specifies the number of instances that may be displayed in the list of instances. The numberInstances attribute of the *PageLists* table will be updated with the integer value;
- *setDefaultOrder(cNamingComp:Component):* the designer is able to choose one of the Naming Components of the Page to be the Naming Component on which the list will be ordered by default. The database will be updated with the id of this Naming Component;

- *setOrderDirection(szDirection:string):* this will update the direction in which the list of instances will be ordered, according to the default Naming Component that has been chosen. This direction can be ascending or descending, according to the preferences of the designer;
- *setRelatedPage(iPageId:int):* we will be able to select one Page from a list of all the Pages that are specified within a Navigation Track and that are not yet related to another Page-List. Or the designer may choose to add a completely new Page, which will automatically be related to the Page-List. For this new Page, the designer will be obliged to specify at least one Naming Component so that we may identify the instances of this Page;
- *getPage():* this function will return an object of the Page on which the Page-List is placed. The Page object is needed to fetch the Naming Components that are defined for that particular Page;
- *addRelatedPage(cPage:Page):* this function will update the *PageLists* table and set the id of the related Page. Together with this function, we have the *deleteRelatedPage()* function which will set the value of the relatedPageId attribute in the PageLists table to zero, indicating that there is not yet a Page set;
- *getPageInstances():* all the instances of a related Page will be found in the *Pages* table and an array consisting of all these instances will be returned, so that the correct XML code can be generated by the *toXML()* function.

With all the classes explained, we have given a complete specification of the Pages and the way they are used within AD-WfW. Now we can come to the specification of the Navigation Tracks, which is another important structures of AD-WfW since it will handle the navigational part of the web application.

## 4.4 Navigation Track

We will follow WSDM in the fact that a Navigation Track is needed for each Audience Class. This Navigation Track needs to fulfill all the information, functional and navigational requirements that are specified for the Audience Class. The Navigation Tracks will be one of the most important features towards the Audience-Driven aspect of the tool. Using these tracks, the visitors of the web application will receive the information they are looking for according to the Audience Class they belong to.

### 4.4.1 Specification

As explained earlier, the Navigation Track consists of a collection of Pages that are connected to each other through Structural Links. First of all, we will have to specify all the different Pages that can be associated to a particular Audience Class, and which will all have to be included in the Navigation Track. We will see a list of all these Pages and we will have to connect them to each other through a new concept called S*tructural Links*.

The Navigation Track is a concept the visitor of the web application will not see explicitly. It is used by the system in order to generate the correct navigational links for the Pages, according to the Structural Links the designer has specified. As stated in the section about the Pages, each Page has a navigational part in which these links are provided to the Pages he can visit from that point. The XML code that will be used to generate that navigational part is generated by the Navigation Tracks.

### 4.4.2 Structural Links

A Structural Link [22, 23] connects two or more Pages to each other. The possible types of Structural Links are:

- One-to-One uni-directional
- One-to-One bi-directional
- One-to-Many uni-directional
- One-to-Many bi-directional
- Many-to-One uni-directional
- Many-to-One bi-directional
- Many-to-Many uni-directional
- Many-to-Many bi-directional

The user will have to select the type of link that he will specify between the Pages. Each of these require their own specifications. For the One-to-One links, the designer will have to choose two Pages. One to indicate on which Page the link starts and one to indicate to which Page the link goes. For the One-to-Many links, we have to specify one Page from where the link starts and multiple Pages to which the link is going. For the other types of links, this will also be the case but the number of Pages in both ends of the link will differ. With a bi-directional link, it does not matter which Pages were selected first, since both the Pages will be accessible through each other.

The Structural Links will also be used to generate the links between the different Navigation Tracks. Since a Navigation Track will be defined for a particular Audience Class, the user who enters the system will only see the Pages that are accessible for the Audience Class he belongs to. But sometimes the user can belong to multiple Audience Classes, which means that he should be able to navigate from the Navigation Track of one Audience Class to the Navigation Track of the other. This could be the case when a particular Audience Class is a subclass. Then the user can navigate from the superclass to the subclass or the other way around. This also means that there should be Structural Links provided on the Pages to go to the other Navigation Track. However, these links can be generated automatically by AD-WfW, because they can be derived from the hierarchy of the Audience Classes.

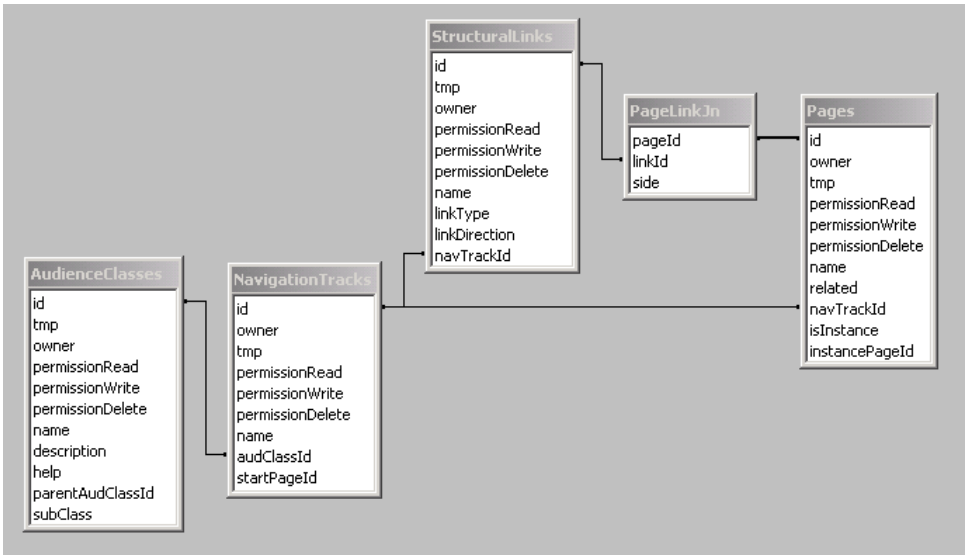



Figure 4.7: "Navigation Track" database Diagram

### 4.4.3 Implementation Design

We will give an overview of the structure of the Meta Database and the implementation for the Navigation Tracks, as well as for the Structural Links that will be used in AD-WfW. As in the previous sections, we will first give an overview of the structure of the Meta Database.

#### Database Design

Figure 4.7 gives an overview of the tables that are added to the Meta Database to support the implementation of the Navigation Tracks and the Structural Links. The new table ***NavigationTracks*** does not need much explanation. It contains only three attributes:

- *audClassId:* this attribute is a foreign key to the id of the *AudienceClasses* table. It contains the id of the Audience Class to which the Navigation Track belongs;
- *startPageId:* Each of the Navigation Tracks need a start Page that the visitor of the Navigation Track will see first. The id of this Page will be stored in this attribute.

The attributes of the ***StructuralLinks*** table may need some more explanation:

- *linkType:* as stated in the specification of the Structural Links, we have four types of links: One-to-One, One-to-Many, Many-to-One and Many-to-Many. Each of these types corresponds with an integer value that will be stored in this attribute;

- *linkDirection:* this attribute will contain a value concerning the direction of the link. This may be uni-directional or bi-directional, both will correspond with an integer value;
- *navTrackId:* this is a foreign key to the id of the *NavigationTracks* table. It will contain the id of the Navigation Track in which the Structural Link is defined.

The ***PageLinkJn*** table will handle the connection of the Structural Links between the different Pages. For each association between a Structural Link and a Page, a row will be stored in this table with the following attributes:

- *pageId:* this is a foreign key to the id of the *Pages* table. It will contain the id of the Page that is linked to the Structural Link;
- *linkId:* this is a foreign key to the id of the *StructuralLinks* table. It will contain the id of the Structural Link to which the Pages are linked. According to this attribute, we will be able to fetch all the Pages for a particular Structural Link;
- *side:* this attribute will contain a value that indicates whether the Page is on the left-hand side or on the right-hand side of the link. The left-hand side indicates the Page where the link will start. The right-hand side indicates the Page where the link ends.

Using the Structural Links we will be able to create the Navigational part of the web application.

### Class Design

Figure 4.8 gives an overview of all the classes needed for the implementation of the Navigation Tracks. The *NavigationTrack* class will play an important role in the navigation part of the AD-WfW web application. The *StructuralLink* class provides the implementation for the links between the different Pages. We will start with the explanation of the ***NavigationTrack*** class:

- *metaToXML():* as we already know this function will generate the correct XML code for the Navigation Track. We also explained that the visitor of the web application will not notice the Navigation Track. This means that the XML code will only be needed for the Meta Part of the web application. The XML code will provide a list of all the Pages that are defined for the Audience Class, using the <lstPages> tag. The designer will have to specify all the Structural Links between the Pages of the Audience Class so that the concept of the Navigation Track can work correctly in the background of the AD-WfW application. Next to the list of all the Pages, the list of all the Structural Links will be provided to the designer. For each of these links we get to see the name of the link and the Pages for which the link is defined, so that the designer can keep a good overview on the structure of the Navigation Track;
- *getXMLNavigation():* this function will return the XML code that is needed to generate the correct navigation menu for the Pages. This function will use the *getNavigationTrack()* function in order to be able to return the correct navigational structure;

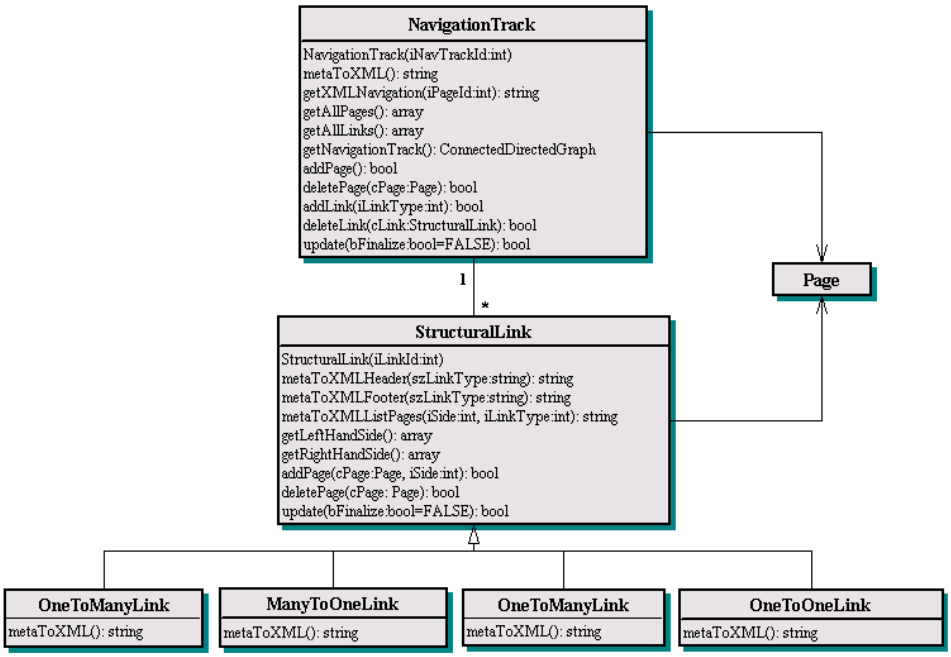



Figure 4.8: "Navigation Track" Class Diagram

- *getAllPages():* an array of all the Pages that are related to a particular Navigation Track will be returned. This function will be used by the *metaToXML()* function, which will traverse the array and generate the correct XML code for the list of all Pages that are available to the Navigation Track;

- *getAllLinks():* an array of all the Structural Links that are defined between the different Pages will be returned. Again, this function will be used by the *metaToXML()* function to generate the XML code for the list of all the Structural Links that are defined within the Navigation Track;

- *getNavigationTrack():* this is the most important function in the *NavigationTrack* class. The result of this function is a Connected Directed Graph of all the Pages in the Navigation Track. This function will loop over all the Structural Links of the Navigation Track in order to generate a correct Connected Directed Graph for the complete navigation structure of the Navigation Track.

Next to the *NavigationTrack* class, we have the ***StructuralLink*** class of which the specifications are:

- *metaToXML():* the StructuralLink class has four subclasses with each their own implementation for the *metaToXML()* function: ***OneToManyLink***, ***ManyToOneLink***, ***OneToOneLink*** and ***ManyToManyLink***. For each of these classes, a different page will be displayed in which the designer can specify the content for that type of link. That is why each

type of link has its own implementation for its presentation. The functionality however can be shared across the different types. The presentation of the links consists of the Page(s) that is (are) specified for the left-hand side and the right-hand side of the link;

- *metaToXMLHeader(szLinkType:string):* each type of Structural Link has the same header, except for the type of link, they will differ;
- *metaToXMLFooter(szLinkType:string):* the end of the XML code for the Structural Links will be the same for each of them;
- *metaToXMLListPages(iSide:int, iLinkId:int):* the XML code for a list of all the Pages that are still available to associate them with a particular side of a particular Structural Link (*iLinkId*), will be returned by this function;
- *getLeftHandSide():* an array of all the Pages that are specified for the left-hand side of the link will be returned. This function will be used by the Navigation Tracks in order to be able to generate a Graph of its structure and by the different types of links in order to be able to generate the correct presentation for the specification of the link;
- *getRightHandSide():* this function has the same functionality as the *getLeftHandSide()* function, but in this case, the pages at the right-hand side of the link will be returned;
- *addPage(page:Page, side:int):* this function will update the *PageLinkJn* table with a new association between a Page and a Link. The side parameter of this function indicates whether the link is added to the left-hand or the right-hand side of the link;
- *deletePage(page:Page):* an association between a Page and a link will be deleted from the database.

## 4.5 Overall Application Design

In the previous sections we have given an overview of the most important new concepts used in AD-WfW, but we still need to explain the overall architecture, which will be responsible for capturing the requests from the visitors. These requests will have to be interpreted and the correct actions will have to be taken. For this purpose, we will use the concept of an Event Handler. This concept was already used in WfW, but we will not reuse this concept in AD-WfW. We will build a complete new *Audience-Driven Event Handler* (AD-EventHandler). In an AD-WfW web application, a request for a particular page will first be passed to the AD-EventHandler, which will process the request and according to the type of request, a couple of actions will be undertaken before the actual page will be displayed to the visitor. One of these actions is that the content of the form fields is written to the database.

One of the main actions that the AD-EventHandler will have to perform is selecting the appropriated XSLT document. Each request to the AD-EventHandler will end in the display of some information, whether it is an 'update successful'-message or the presentation of a particular Page. For each of these displays, the

AD-EventHandler will call the toXML() function for the concepts we discussed in the previous sections when we are in the Data part of the web application. We will also use an AD-MetaEventHandler for the Meta part of the web application. This EventHandler will have the responsibility of performing the correct operations concerning the specifications of the Audience Classes, the Navigation Tracks, the Pages ... The metaToXML() function will be called to fetch the correct XML code. The EventHandlers have the responsibility over the task of selecting the correct XSLT file for converting the XML code, since they know exactly the type of information that will be displayed.

In addition to the EventHandlers, a mechanism is needed that will be able to specify the Audience Class of a visitor since the concept of an Audience-Driven web application is built on this property. For the division of the visitors into the different Audience Classes, we will make use of the security system that has been implemented for WfW. For the moment, the security system can add a new user-login and specify to which group this user-login belongs.

When the user enters the web application, a Page will be displayed to him. On this page, he is given the choice to login to the web application, or he may choose one of the Audience Classes to which he may belong. He will choose one of these Audience Classes and AD-WfW will bring him to the corresponding Navigation Track. From that moment on, he will be able to browse through the Pages of the Navigation Track, following the Structural Links. Selecting the correct Audience Class for the visitor will be handled by the Overall-AD-WfW structure. This will generate the starting Page and set the necessary cookies. These cookies will be needed to know the choice of Audience Class the visitor has made. But the selection process is not the only thing that will be handled by this structure. Also adding and deleting Audience Classes are part of its functionality. It will generate the XML code for a list of all the Audience Classes that are available in the system, which is part of the Meta part of the web application.

### 4.5.1 Implementation Design

Since the overall AD-WfW structure does not need any additional tables in the database, no Database Design is needed for this section. We will immediately go to the Class Design.

#### Class Design

The class diagram of the Overall-AD-WfW structure is given in figure 4.9. The functionality of the ***OverallADWfW*** class includes:

- *startToXML():* this function will return the XML code for the start Page of the web application, where the visitor will make the selection of the Audience Class to which he belongs. The XML code will contain a list of all the Audience Classes;
- *startMetaToXML():* this function will use the *startToXML()* function to generate the XML document, but another XSLT document will be used to convert the XML document into a different presentation, in which the

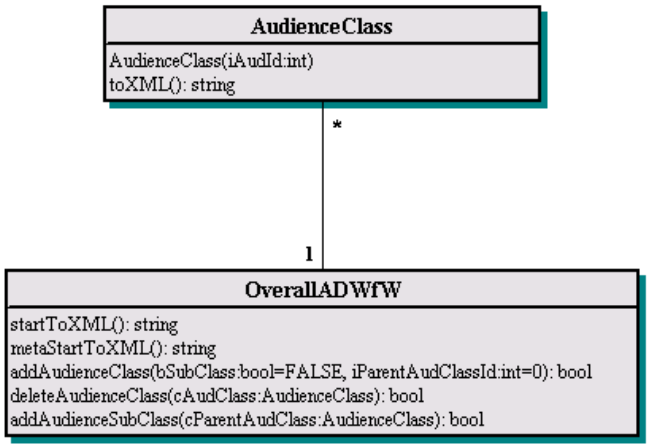



Figure 4.9: "Overall AD-WfW" Class Diagram

complete list of Audience Classes will be shown and in which the user has the possibility to update or delete them;

- *addAudienceClass(bSubClass:boo=FALSE, iParentAudClassId:int=0):* this function will update the content of the *AudienceClasses* table and insert a new row for a new Audience Class. The two parameters are used when this function is called from within the *addAudienceSubClass()* function. Both functions have the same functionality except for the fact that it is possible to specify an Audience Subclass or not.

### 4.5.2 Overall Functionality

In this part we will give an overview of all the possible paths the user can follow through AD-WfW, using State Diagrams [21]. We will split this explanation into two parts: the Meta Part and the Data Part. Each of them has another state diagram and will also have a completely different path that the user can follow. We will start with the Meta Part of which the State Diagram is given in figure 4.10.

#### Meta Part

When the user enters the Meta Part, he will see a list of all the Audience Classes that are available. The rest of the path the user is able to follow is quite straightforward. When he is in a particular state, he has the choice to follow one of the paths, as indicated on the diagram with an arrow. To keep the State Diagram simple, we have omitted the conditions needed to be able to reach the next state. The *Edit Component* state however has some conditions that may need some attention. The *Add/Edit Subcomponent* action will only be available when the Component type we are defining at that time is a Mixed Component, since this is the only Component that can have some Subcomponents. The *Set Association Type* action will only be available when we are defining an Information Component. The type of this Component can be Simple or Combined.

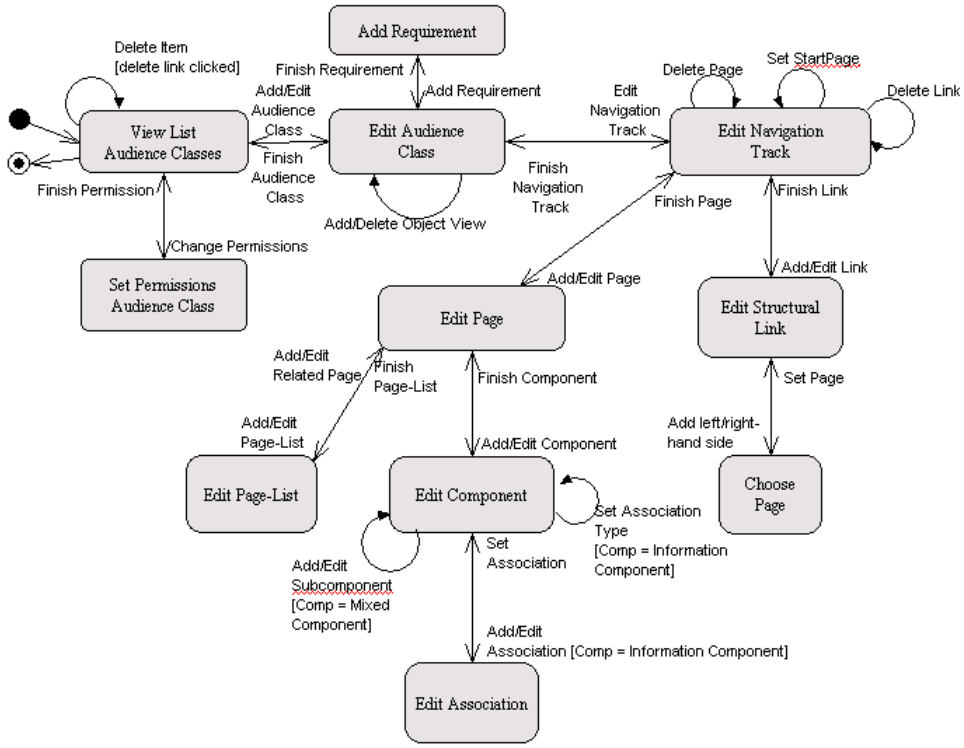



Figure 4.10: State Diagram for the Meta Part

Furthermore we have the action *Add/Edit Association* that will also be available when we have an Information Component. With this action, we can add a new Attribute or a new Category by selecting them from a list. The last state we will discuss is the *Edit Page-List*. We can see two actions going back to the *Edit Page* state: *Finish Page-List* and *Add/Edit Related Page*. The first action will be used when the designer finishes the Page-List and adds it to a particular Page. The Page-List however should be associated to another related Page. Therefore, the designer is able to add and edit that related Page, and that is why we are using the second action.

### Data Part

The Data Part of AD-WfW has a completely different State Diagram and it is shown in figure 4.11. Again, no conditions are shown in this Diagram, since the only condition that applies to all the actions concerns the security of AD-WfW. An action can only be taken when the user has permission to access a particular Audience Class or a Page. For each page to which the user has read-permission a link will be provided so that this Page can be reached. When the user enters a Page, and he has write-permission to that particular Page, he will see an edit-button so that he can change the contents of that Page. The links he receives to get access to the different Pages in the web application, are generated by the Navigation Track of the Audience Class, according to the security settings applied to them.

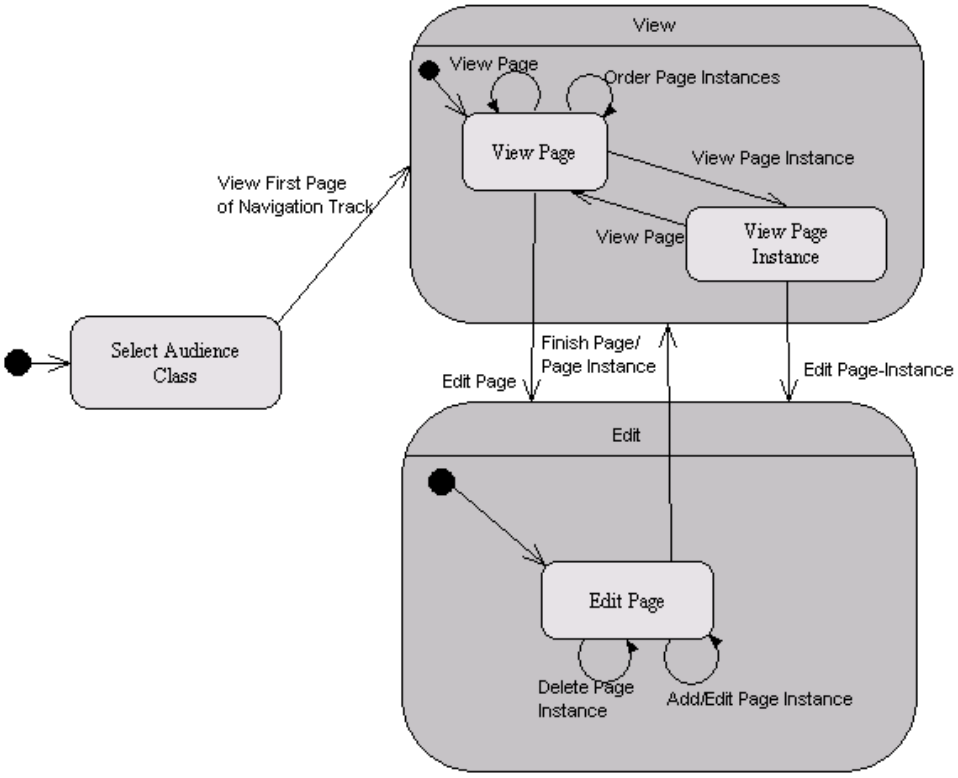



Figure 4.11: State Diagram for the Data Part

# 5 Case Study

For our Case Study, we will use the example of the Primary School again, the way we already discussed it in section 2 about WSDM. In the previous sections we have explained the underlying implementation of AD-WfW, but what the reader will be most interested in, is how this tool works: How the pages look like and which functionality there is available. Together with some screenshots, we will give a detailed explanation of the tasks the designer has to perform to come to a complete Audience-Driven web application in which students, parents and teachers will each have their own section, focused on their needs. I will give the explanation for the 'Parent' Audience Class and we may assume that the 'Visitor' and 'Student' Audience Classes, of which the 'Parent' is a subclass, are already defined. For the 'Visitor' Audience Class we have one Page, namely for the General Information and for the 'Student' we have also one Page, namely the Classes Page, in which a list of all the Classes of the school is shown. We will start with the explanation of the ***Meta Part*** of AD-WfW. In figure 5.1, the Page for editing an Audience Class is given.

The most important fields of this Page are the Associated Object Views, in which we get a list of all the Object Views from which we will be able to select the possible associations for the Information Components. The Possible Object Views field displays a collection of all the Object Views that are defined in WfW. The 'Add new Object View'-button will take the designer to WfW where he can specify the content of a new Object View (for the specification of an Object View, go to section 1.1). For each Audience Class we have to specify a Navigation Track, the Meta Page for the specification of the content of a Navigation Track is given in figure 5.2.

For our example, we see that there are already two Pages defined, those that belong to the parent Audience Classes 'Visitor' and 'Student'. We can use these Pages when specifying the Structural Links for the Navigation Track. They are an example of the fact that a visitor of the web application may belong to one or more Audience Classes and that he has the ability to move from one to another. Now we will add a new Page to the Navigation Track, namely the 'Teachers' Page. This Page will contain a list of all the teachers of the school, together with their details. The Meta Page for specification of the content of this Page is given in figure 5.3.

We have specified two items for the contents of the Page: 'Page Description' and 'Teachers'. The Page Description is a Simple Information Component, related to a text-Attribute. It will contain a small description of the 'Teachers' Page, describing what the user may expect. The Meta Page for the specification of this Component is given in figure 5.4.

We can see that we can specify the presentation of this type of Component, as specified in section 4.2.4 by setting the header and the number of presentation columns. Furthermore we have the possibility to change the type from a Simple to a Combined Information Component or vice versa. According to the type of Component, the Association-field will have another content: an association to an existing Attribute of one of the Object Views or an association

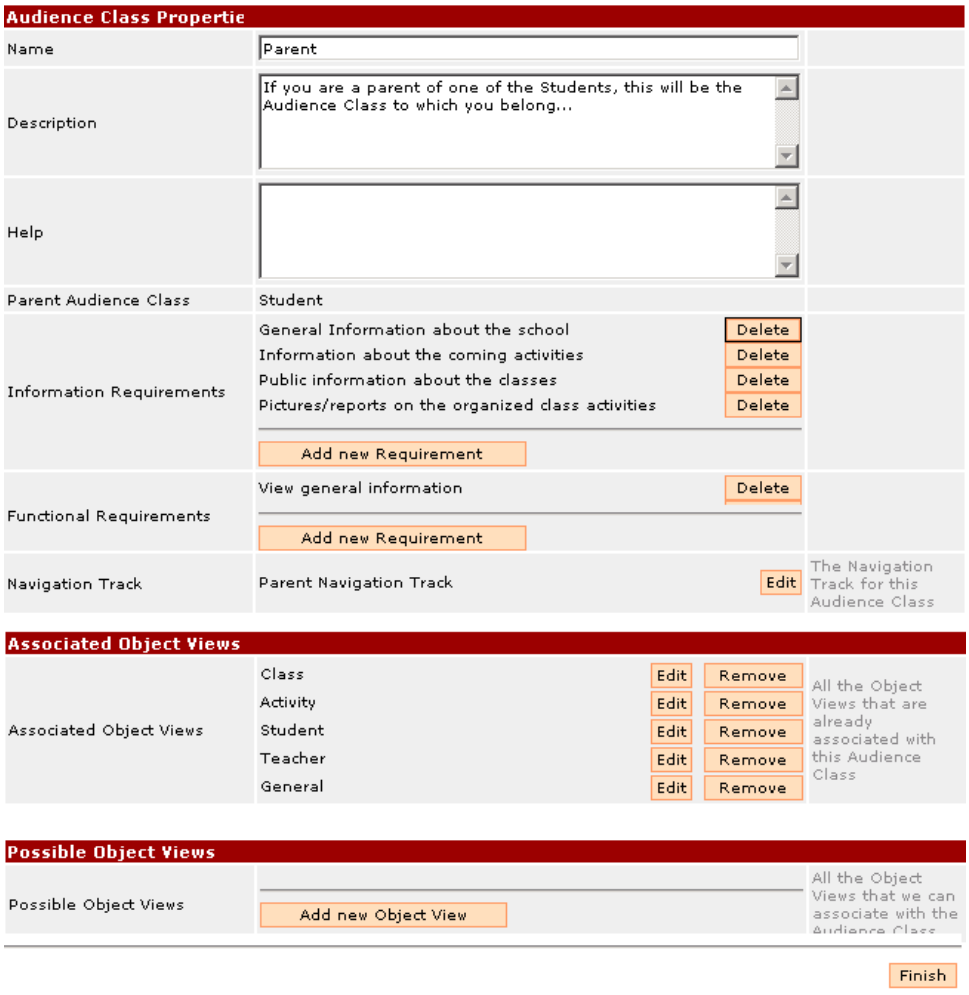



Figure 5.1: Edit "Parent" Audience Class

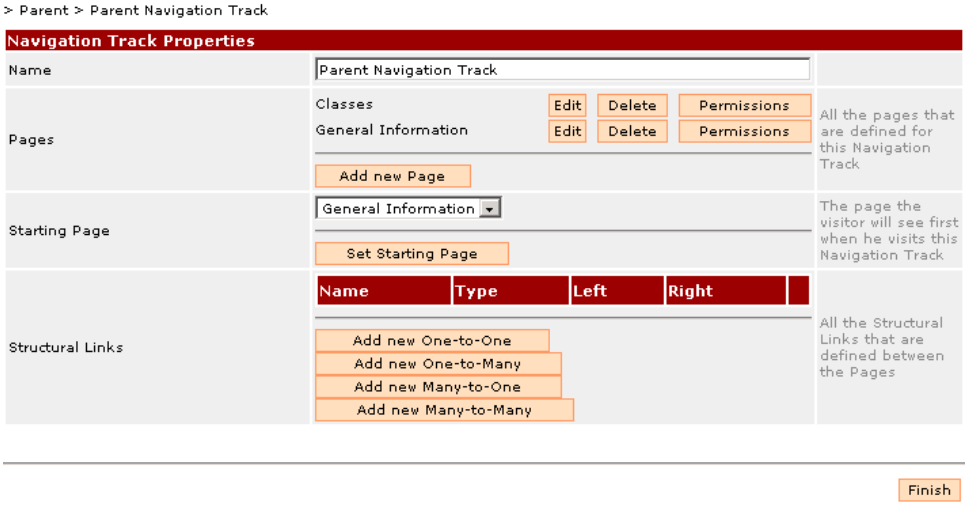



Figure 5.2: Edit Navigation Track Page

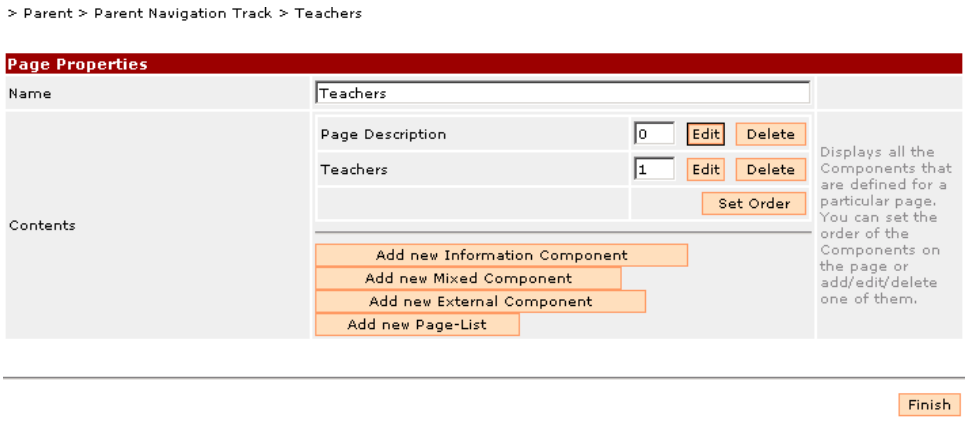



Figure 5.3: Edit "Teachers" Page

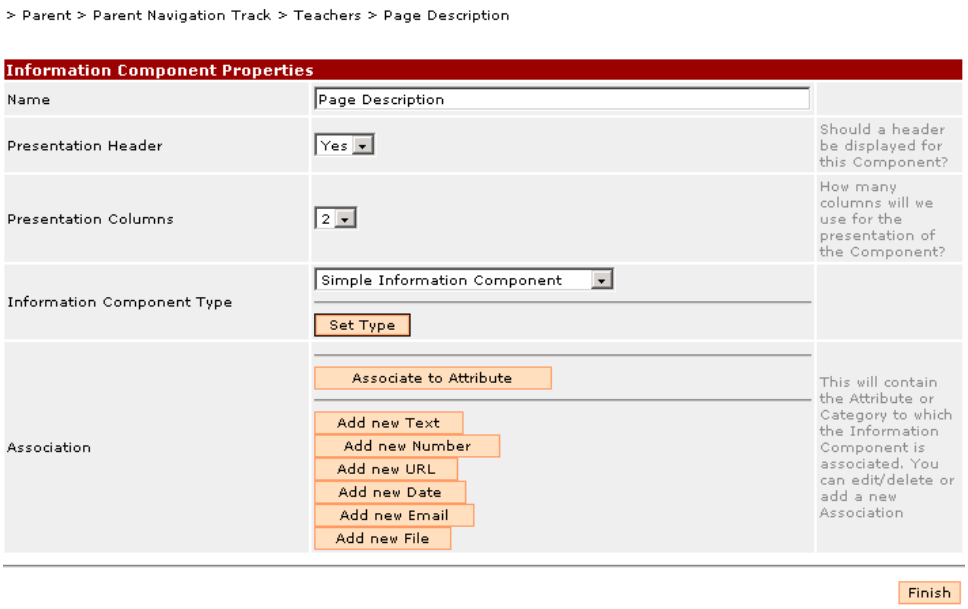



Figure 5.4: Edit Simple Information Component

to a particular piece of text, a number, an URL, which will not be related to an Object View. For the Combined Information Component, we will be able to associate the Component to a particular Category of one of the Object Views.

For the Page-List 'Teachers', we are able to define the number of Instances that should be shown in the list and the Page to which the Page-List is related. This related Page will be used as a 'template' for the all the instances. For our example, we have related the 'Teacher' Page to the Page-List. This Page will not be visible in the Navigation Track since it is only accessible through the instances of the Page-List. The specification of the content of that Page is given in figure 5.5.

For the first time, we use the Naming Components field. When we define a related Page, we will use the Naming Components to generate the name for the instances of the Page. The Page-name will consist of the value for the Naming Components, as stored in the Database. Furthermore, the specification of a related Page is exactly the same as that of a regular Page.

For the 'Parent' Navigation Track, we will need also an 'Activities' Page on which we specify all the activities that are organized by the school. The specification of this Page is similar to that of the 'Teachers' Page, where we will include a Page-List structure that has a relation to an 'Activity' Page which will hold the details of the different activities. We will not repeat the specification process anymore, since this is already explained in the preceding parts.

But what we do need for AD-WfW to work properly is to define the Structural Links between the different Pages. This has to be done in the Navigation

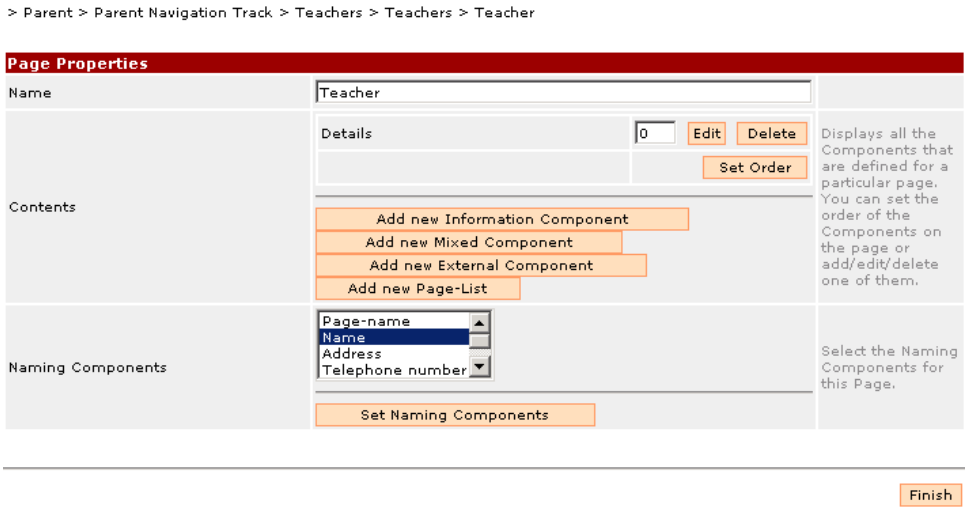



Figure 5.5: Edit Related Page "Teacher"

Track and the resulting links are shown in figure 5.6. For each of the links, we will have to specify the Pages on the left-hand side and the Pages on the right-hand side. The AD-WfW system will generate the necessary links of the navigation part of a Page with them.

Now that we have defined the complete structure for each section of the different Audience Classes, we can go to the ***Data Part*** of the web application. Where the Meta Part uses mainly the red and orange colors for the headers and the buttons, the Data Part will use a blue color. This feature is used to make it clear to the designer in which part he is.

When a visitor comes to the website, he will first have to choose the Audience Class to which he belongs. Therefore, he is given a Page as shown in figure 5.7. Once he has chosen the Audience Class, he is taken to the Starting Page of the Navigation Track for that particular Audience Class. For the 'Parent' Audience Class, this will be the 'General Information' Page. This Page is shown in figure 5.8. On top of the Page we can see the links that are available from this Page, according to the Structural Links in the Navigation Track. When we go to the 'Classes' Page, we get a Page as shown in figure 5.9.

As we can see, the Page only displays five Classes. We can find the rest of the Classes by clicking on the 'Next' link at the bottom of the list. When we login as a designer (Administrator) of the web application, we will see an edit-button on the bottom of the Page. When we click on that button, we are taken to the Edit Page for the Classes, in which we will be able to change the information of that Page. For the 'Classes' Page, we will get the Page as shown in figure 5.10. One could think that we are mixing up the Meta and the Data Part, but this is not the case. Since we are modifying the data of the web application and not the structure of that data, we are indeed in the Data Part with this Page. The Meta Part will deal with the specification of the structure of the Pages, such as

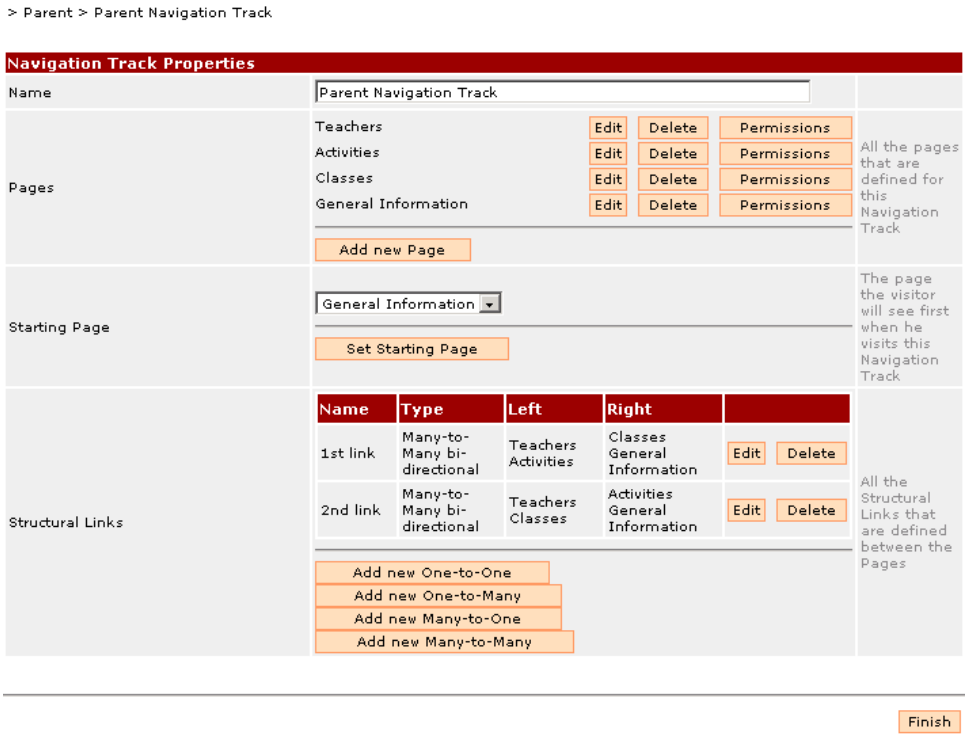



Figure 5.6: Edit Navigation Track with Structural Links

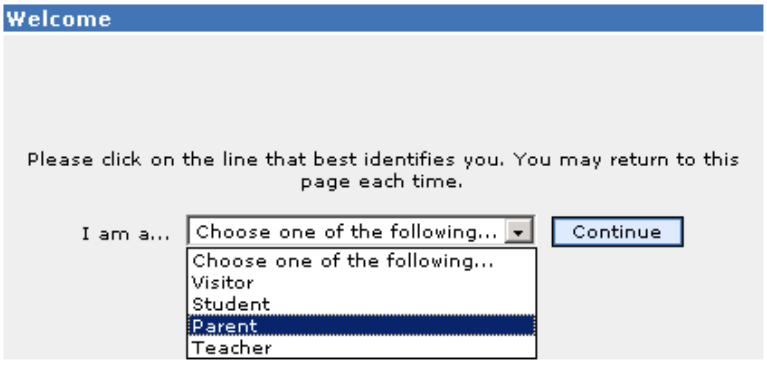



Figure 5.7: Choose Audience Class

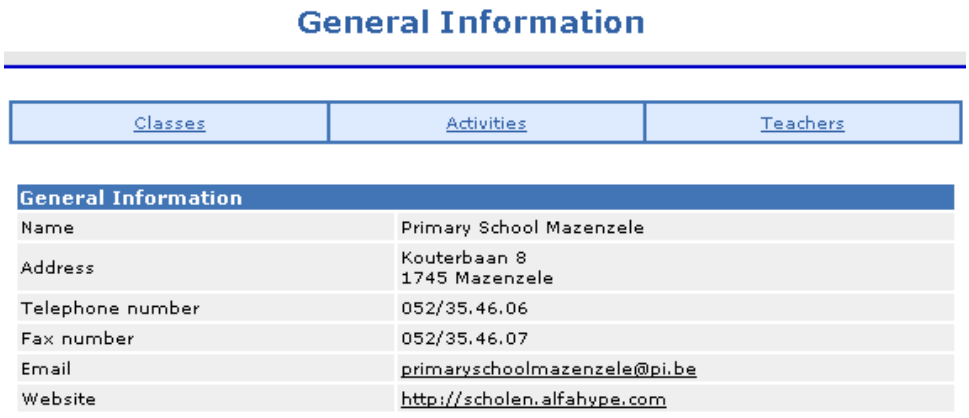



Figure 5.8: "General Information" Page

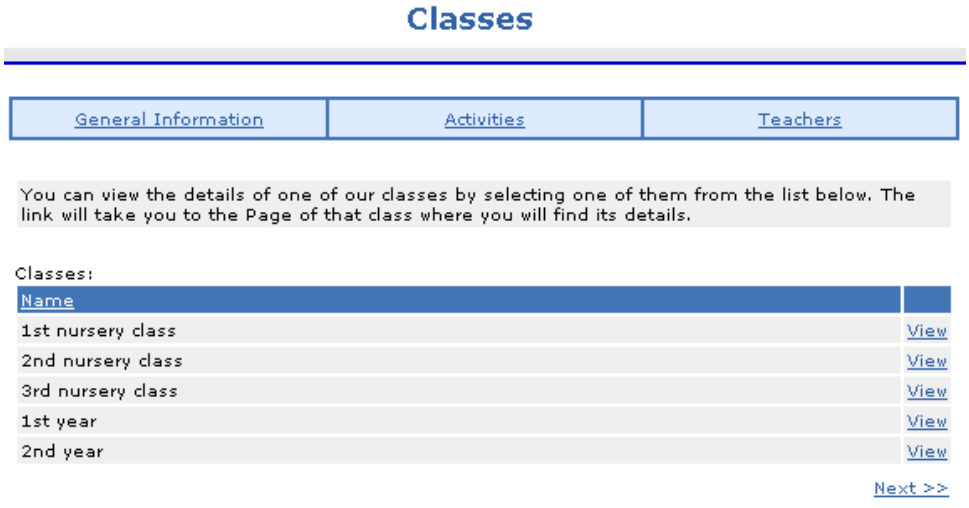



Figure 5.9: "Classes" Page

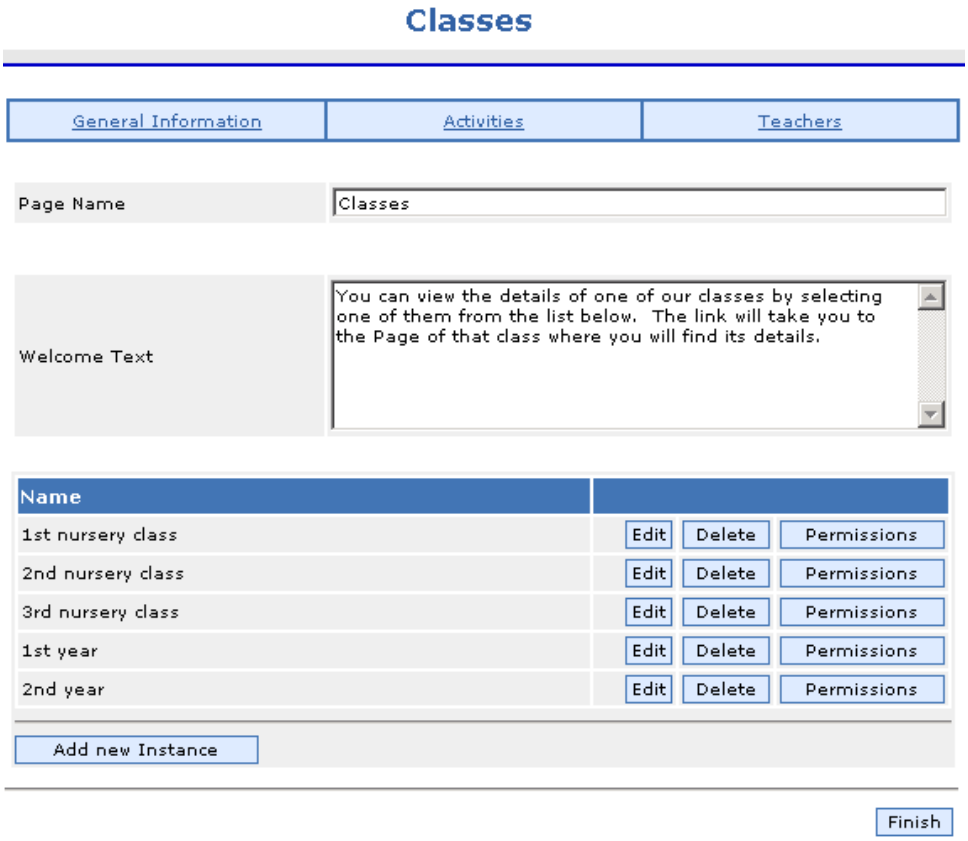



Figure 5.10: Edit "Classes" Page

the different Components that should be placed on a particular Page.

On the Edit Page, we get the form fields for each of the Components and a complete list of all the instances of the Classes. We can add a new instance to this list or edit/delete one of the existing instances. When we edit the '1st nursery class' instance, we get the edit Page as displayed in figure 5.11. For each instance of a Class, we will have the possibility to specify its content, according to the template of the related 'Class' Page, as it will be specified in the Meta Part: the 'Class Name', the 'Number of Students', the name of the 'Teacher' and a 'Class Picture'. Furthermore we have two Page-Lists on this Page: one for the Students and one for the Activities for the Class. A last feature I would like to discuss on this Page is the top field: 'Associate to existing instance of Object View'. This field is only available when we are defining a Page Instance. It will display all the Object Views that are used to be able to display the contents of that particular Page. For this example, we have only used the Object View 'Class'. We have the possibility to associate an existing instance of that Object View to the Page. When, for example, we would decide that this Page would have to contain the content of '2nd nursery class', we would be able to select that instance from a list and the content of '2nd nursery class' would be shown instead of the content of '1st nursery class'. This option will mostly be used

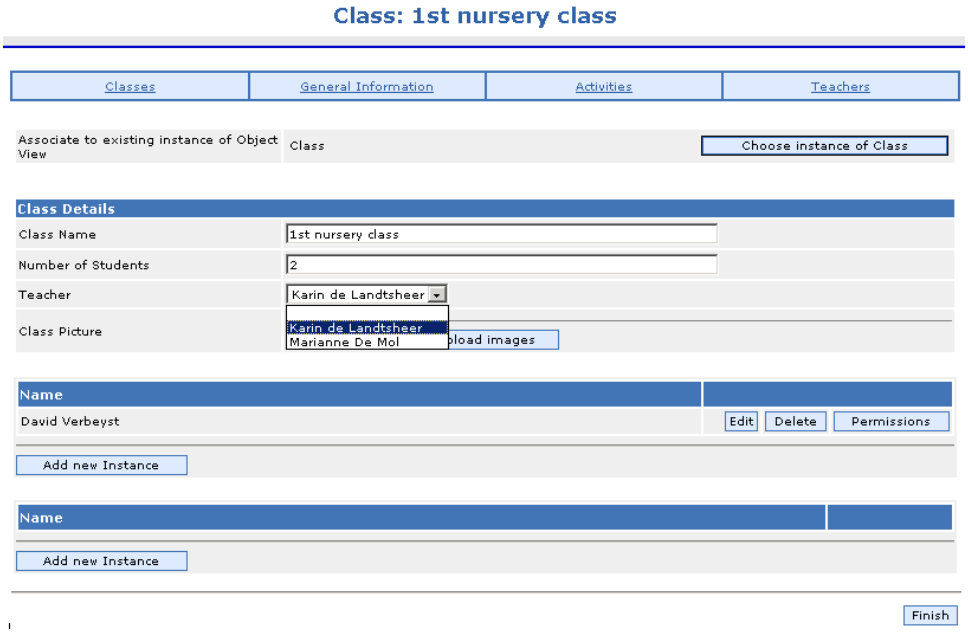



Figure 5.11: Edit Page for "Class: 1st Nursery Class"

when we use an instance of an Object View more than once in the web application. An example can be found with the Activities. When the visitor clicks on the 'Activities' link on top of the Page, he will see all the Activities organized within the school. These Activities however will also be available within the list of Activities in each Class. We are pointing to the same instance on different Pages, so when we change the information on one Page, the information on the other Page will also change.

With these screenshots and their explanations, I have tried to give an illustration of the different features of the tool and how the designer and visitors can work with them. The reader of this thesis should now have a good overview of the complete AD-WfW tool with a detailed description of the inside structure (the implementation with the database and Class Diagrams) and the outside presentation (the user interface) of the tool.

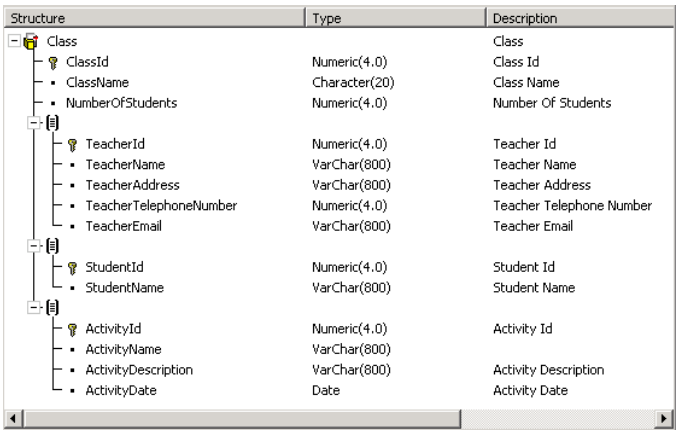



Figure 6.1: Example of the "Class" Business Component

# 6 Related Work

In this chapter, we discuss tools that are related (in some respect) to the tools (WfW and AD-WfW) we have discussed in this thesis. The first tool I am going to discuss is an extension to Visual Studio .NET of Microsoft [24], called DeKlarit [25]. The second tool is actually a family of tools, called the Content Management Systems. I will make a selection of some of these tools and will give a short description of them.

## 6.1 DeKlarit

The aim of Web-for-Web was to create an interface in which the users could easily create and update new tables to a database and create a web interface for accessing the database, without having detailed knowledge about a DBMS, nor about internet technology. Therefore, the user interface has been developed as a web interface in which designers can build a complete database model by means of Object Classes and their Attributes and specify the user interface.

DeKlarit allows the designer to build a complete normalized relational database schema, without the designer having knowledge about the database schema itself. What the designer has to do is specifying some Business Components, which represent some real life entities, such as Classes, Students, Activities ... These will be used to add/update/delete data to the database. The designer will specify the content of the Business Components as these would be represented in real life, without worrying how they are stored in the database. An example of how a class will be represented in the database is given in figure 6.1. A Class itself has some properties as the Class Name and the Number of Students, but also we can specify the relationships to the Students, the Teacher and the Activities. When we save this Business Component, Deklarit will automatically create the necessary tables in the database. This means: a table for the Class and tables for the relationships.

Furthermore, we can create a Business Component for each of the relation-

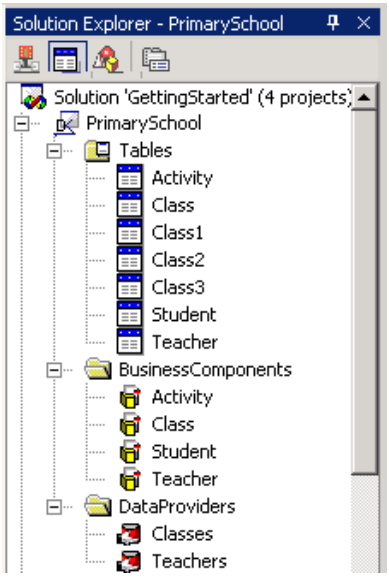



Figure 6.2: Primary School example: Tables, Business Components and Data Providers

ships: one for a Student, one for an Activity and one for the Teachers. For each additional Business Component, a new table will be created in the database and the complete database schema will be normalized again. We can also add some Data Providers to the project. These are hierarchical views that are used to retrieve some information from the database. They cannot be used to edit or delete the data of the database, for that purpose the Business Components will be used. The specification for each of these Data Providers is the same as for the Business Components. The designer can also specify the attributes that have to be displayed, but each of these attributes will have to match with one of the attributes of one of the Business Components. The Business Components, Data Providers and Database Tables that we are using for our test program (the same as for our case study) is given in figure 6.2. As we can see, the Deklarit tool has created seven different tables. The Activity, Class, Student and Teacher tables are quite straightforward, the Class1, Class2 and Class3 tables are used as join-tables between the Class and the Students, Activities and Teachers.

With the complete specification of all the Business Components and the Data Providers, we can get to what we were looking for: generating a complete Website with it. This is one of the options DeKlarit provides: generating an ASP.NET web application. This application will contain links to all the specified Business Components and Data Providers and for the Business Components, the necessary functionality will be provided to add/edit/delete the corresponding data of the database. For this purpose, a number of different ASP-files have been created, which make extensive use of the DataGrid control of ASP .NET [26]. This control is used to iterate over a collection of information (the content of a database) automatically and render a HTML table with some HTML code for each item of the collection. Each of the automatically generated ASP-files

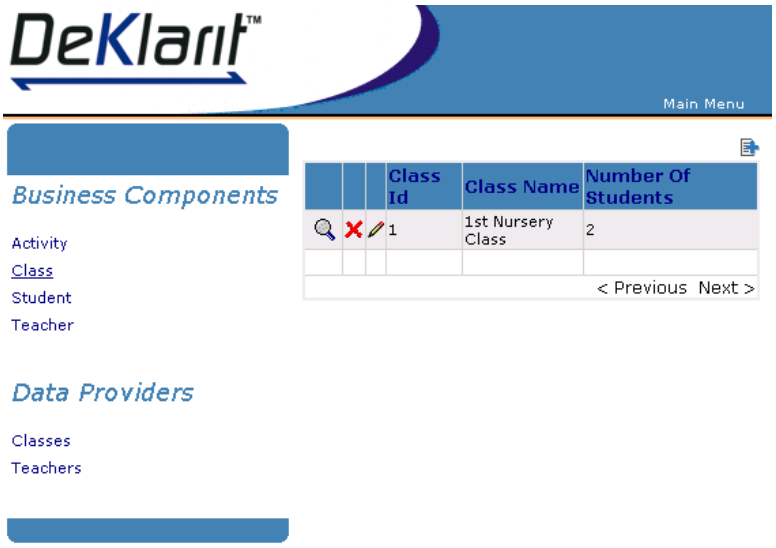



Figure 6.3: View Class Instances example Page

will make use of the DataGrid control and will use it to display its information by means of HTML tables, with or without some editing capabilities, according to the user's intensions. Examples of the generated web application are given in figure 6.3 and 6.4, where we can see the possibilities to simply view the instances of a particular Business Component and edit one of these instances.

### Conclusion

My experience with DeKlarit is that it could be a good alternative to Web-for-Web for editing the content of the database. The tool is completely focused on the database and the ability to define the structure of the database with simpler concepts such as the Business Components. The resulting databases, whether they come from WfW or from DeKlarit, have almost the same structure for the same sort of data. DeKlarit however has some limitations. I have been searching in the tool to find some possibilities to include some pictures on the pages, but I have not found any. Also the possibility to provide a link to an external URL or an email-address is not provided. Possibly this is because the tool is focused so much on the database that the presentation of the data was less important, as long as we are able to edit the data in an easy way. DeKlarit is also not limited to the generation of ASP-files that can be used for the web, but we are also able to generate standalone Windows applications and other formats, which support the editing of the data in the database in the same way as the ASP web application does. Maybe that is the reason why DeKlarit has not spent much attention to the presentation of the information in the ASP .NET web application, which is extremely data-driven. So we do not need to mention that this tool is certainly not suitable for creating an Audience-Driven web application. There is no basic support at all for the generation of different Audience Classes, but maybe this could be an addition to this tool, where the

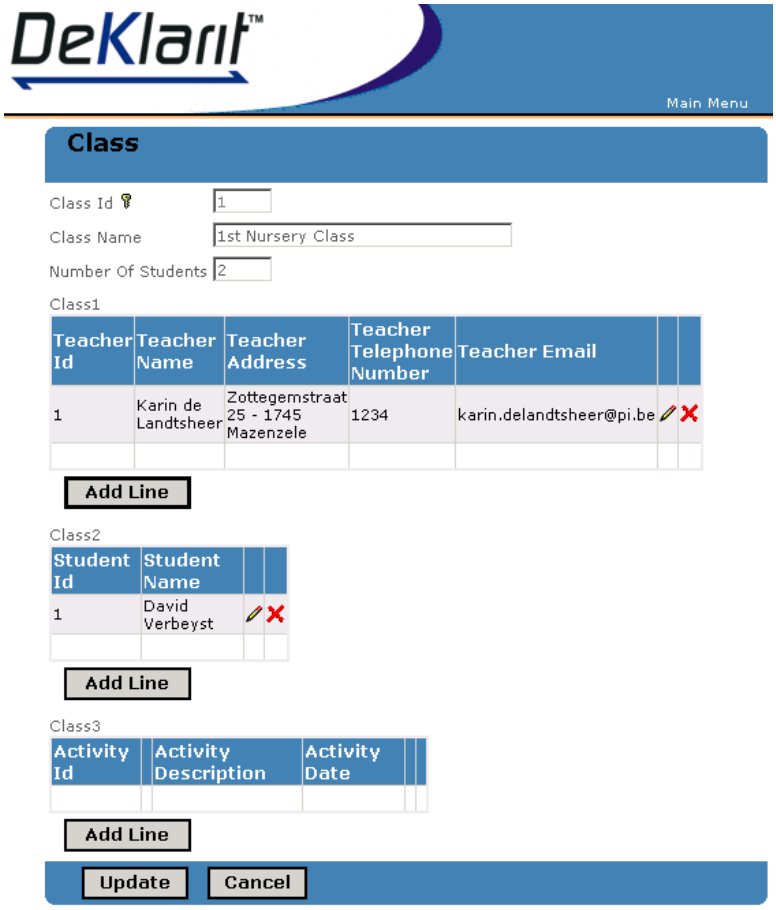



Figure 6.4: Edit Class Instance example Page

same principle as for AD-WfW could be used: building AD-DeKlarit as a layer around DeKlarit and use all its functionality to generate an Audience-Driven web application.

My conclusion for this tool is that it resembles a lot with Web-for-Web, that is why I discussed it in this section. By using DeKlarit, it is only possible to modify the structure of the relational database through an application that only runs using Visual Studio .NET. This means that people who do not possess VS .NET, will not be able to use this application. For WfW, this is not the case, since we are able to update the database through a simple browser, and WfW does not depend on other applications (except for the server). This tool has a lot to do with the data part of AD-WfW, but misses the functionality to generate an Audience-Driven web application.

## 6.2 Content Management Systems

Where the previous tool, DeKlarit, misses the features to easily define a possible navigation structure for the web application, this is largely compensated in the Content Management Systems. A Content Management System (CMS) is a system that runs on the web server, which we can access through a simple web browser enabling us to manage the complete content of a website through the web. With this description we come close to AD-WfW: a web interface from which we can create an Audience-Driven web application. There are a lot of CMSs around and each of them has its own functionality and its own features. Each of these CMSs uses its own standards (XML, XSLT, ...), uses its own type of Database System (MsSql, MySql, Oracle, ...) and is programmed in its own programming language (PHP, Java, ...). Since they all have the same purpose (creating a complete web application through a web interface) [27], I will only list a couple of Open Source CMSs with their most important features [28] to show the differences between them.

### Bitflux

**Description:** *Bitflux [29] is an XML based CMS with a WYSIWYG XML editor. It allows the user to reuse the content of the web application in different ways. It uses XSLT as a template engine to convert the XML documents to HTML. Furthermore it uses "Popoon" as a backend, which is loosely based on Apache Cocoon, so that it is customizable to the designer's needs.*
**Computer Platform:** *GNU/Linux, Macintosh OS X*
**Programming Language:** *PHP*
**Database:** *MySQL*
**Standards:** *XML, XSLT*

### Callisto

**Description:** *Callisto CMS [30] is an XML/XSL Web-based Content Management System built using Perl and AxKit. Designed for maintaining XSL-based websites, it can edit various types of XML files, and providing a WYSIWYG*

*interface by using the site's very own XSL stylesheets. Semantic data is edited on a per-region basis, in an "offline" domain (e.g. admin.domain.com). Sites can be deployed to production webservers when changes are made. The sites are deployed transactionally, and can be deployed to multiple target webservers, again, transactionally. Multiple independant sites can be managed in one Callisto installation, and one user login can be used for several sites, if given access.*
**Computer Platform:** *GNU/Linux, Windows*
**Programming Language:** *Perl*
**Database:** *none*
**Standards:** *XML, XSL*

#### Apache Lenya

**Description:** *Apache Lenya (formerly Wyona CMS) [31] is an Open-Source Content Management and Publishing System written in 100% pure Java. It is based on open standards such as XML and XSLT. One of its core components is Cocoon from the Apache Software Foundation.*
**Computer Platform:** *GNU/Linux, Windows, Macintosh OS X, Unix*
**Programming Language:** *Java*
**Database:** *MySQL, Oracle, PostgreSQL*
**Standards:** *XML, XSLT*

#### Typo3

**Description:** *TYPO3 [32] is a free Open Source content management system for enterprise purposes on the web and in intranets. It offers full flexibility and extendability while featuring an accomplished set of ready-made interfaces, functions and modules.*
**Computer Platform:** *GNU/Linux*
**Programming Language:** *PHP*
**Database:** *MySQL*
**Standards:** *HTML*

With the Typo3 CMS, we have tried to build the Audience-Driven web application for our Primary School example used for AD-WfW. The result for the Classes Page is shown in figure 6.5. At first sight, we could say that this looks quite good. But when we go into further detail, we have to say that the way we have specified the content for this Page is not exactly what we had in mind. The Typo3 CMS gives us a lot of possibilities to specify the content of a particular Page, but it has no features at all that lets us specify a table for which we have the possibility to add some instances which are automatically added to the table (e.g. a Page-List). For this Classes Page, the first thing we had to do is add five new Pages to the web application: 1st Nursery Class ... And for each of these Pages, we had to specify its content, but the content of each of them can be different, which is against one of the principles of a uniform layout for the Class-instances. Furthermore, we had to add simple HTML-links on the Page to be able to visit the different Classes, which makes it hard to update the list, since we have to do it all by hand.

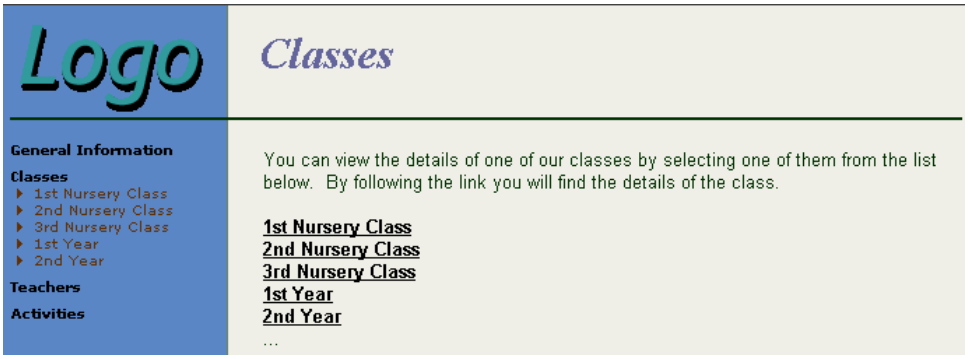



Figure 6.5: Typo3 example of the Classes Page

Furthermore, we are only able to simulate the Audience-Driven aspect of the web application through the use of the security system of Typo3. For each of the Pages, we are able to set security constraints about the groups of people that have access to the Page. With this security system, we are able to omit links to some Pages, which gives the impression that we have an Audience-Driven web application. But the problem is that the user has to login before the Audience-Driven aspect can be applied. This is not the case in AD-WfW, where we also have the security system for displaying or omitting links to particular Pages, but the user does not have to login for each type of Audience Class. When he chooses the Parent Audience Class, he can simply visit that part of the web application without logging in. This would not be the case in Typo3.

### Conclusion

The Content Management Systems are a good way to build some general web applications which need to be very flexible in the way we can update them and they have the capabilities of simulating an Audience-Driven web application, however, they all miss some important features which is the reason why we have built AD-WfW.

The most important reason is the fact that each of the Content Management Systems uses only one database. This database is used for the description of the data (Meta Data in AD-WfW) and the data itself. Our purpose to build AD-WfW was to be able to display the information of a database in an Audience-Driven way. That is why AD-WfW uses two databases: one for the data (Database) and one for its description (Meta Database). In this way, we are still able to reuse the data from the Database in other applications since it is stored in such a way that other users will still understand the meaning of the data. For the CMSs this is not the case. The data is stored in such a way that it is suitable for the CMS itself. When somebody would look at the database, he would not understand its structure and he would have some big problems in finding the data he needs.

The next reason we built AD-WfW is that I did not find a CMS in which I

could define a sort of template for a particular Page, which I could use over and over again to specify the same layout for a number of Pages (called Instances in AD-WfW). This is useful when we think of displaying the information of a particular table in the Database.

The last reason is the fact that we are only able to simulate the Audience-Driven aspect of a web application through the security system. We should also be able to support different types of users without having to create a security-account for each of them. When this would be supported, together with the separation of the presentation of the data and the data itself, the CMSs could be a good alternative for AD-WfW.

# 7 Conclusions

The goal of this thesis was to design a tool that was able to support the generation of an Audience-Driven web application, built on top of the existing tool Web-for-Web that handles the operations on the Database. Until now, the existing tools to build a web application had no explicit support for the Audience-Driven approach.

We designed a tool that gives the user complete control over the Audience-Driven web application. This tool is capable of:

- managing the structure of the Database and the data in it.
- specifying the different Audience Classes for the web application together with their Information and Functional Requirements. They are needed to structure the web application.
- specifying the content of the different Pages for each type of Audience Class by means of Components, which will use the data of the Database, using the implementation of Web-for-Web.
- specifying the navigational track for each type of Audience Class by providing Structural Links between the different Pages. This navigational track will be the menu that the visitor will see, based on the security settings of the Pages, while browsing through the web application.

When the complete design of the tool was finished, the tool was implemented as proof of concept. To demonstrate that the implementation was complete and satisfied the requirements, the tool was used for a case study about an Audience-Driven website for a Primary School.

My conclusion for this thesis is that I have met the goal I had set . I was able to turn Web-for-Web into an audience-driven version with a minimal effort. The complete Audience-Driven Web-for-Web tool was designed and built as a layer around Web-for-Web. The tool is functionally complete and can be used for websites that have a large amount of data that should be presented in a way that is suitable to its visitors.

## References


[1] O. De Troyer (2001), *"Audience-driven web design"*, In Information modelling in the new millennium, Eds. Matt Rossi and Keng Siau, IDEA Group-Publishing, ISBN 1-878289-77-2.

[2] O. De Troyer, J. De Greef, J. and S. Stuer (2002), *"Web-for-Web: A Tool for Evolving Data-Driven Web Applications"*, in Proceedings of the WWW2002 Workshop on Real World RDF and Semantic Web Applications, Hawaii.

[3] "Custom apps", http://www.devigus.com/serv_custom_apps.asp.

[4] "PHP: Hypertext Preprocessor", http://www.php.net.

[5] L. Algerich, C. Lea, K. Egervari, M. Anton, C. Hubbard, J. Fuller and C. Killian (2002), *"Professional PHP4 XML"*, Wrox Press Inc, ISBN 1-861007-21-3.

[6] M. Zandstra (2002), *"Sam's Teach Yourself PHP in 24 Hours (2nd Edition)"*, Sams, ISBN 0-672-32311-7.

[7] R. Lerdorf and K. Tatroe (2002), *"Programming PHP"*, O'Reilly and Associates, ISBN 1-56592-610-2.

[8] A. Ceponkus, F. Hoodbhoy (1999), *"Applied XML: A Toolkit for Programmers"*, John Wiley and Sons, ISBN 0-471-34402-8.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler (2000), "Extensible Markup Language (XML) 1.0 (Second Edition)", `http://www.w3.org/TR/REC-xml`.

[10] C. F. GoldFarb (1996), *"The Roots of SGML - A Personal Recollection"*, http://www.sgmlsource.com/history/roots.htm.

[11] M. J. Young (2001), *"XML Step by Step, Second Edition"*, Microsoft Press, ISBN 0-7356-1465-2.

[12] H. W. Lie, B. Bos (1999), "Cascading Style Sheets", http://www.w3.org/TR/REC-CSS1.

[13] "Extensible Stylesheet Language (XSL)", http://www.w3.org/TR/xsl/.

[14] K. Y. Fung (2000), *"XSLT: Working with XML and HTML"*, Addison-Wesley Pub Co, ISBN 0-201-71103-6.

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

[16] O. De Troyer, C. Leune (1998), *"WSDM: A User-Centered Design Method for Web Sites"*, In Computer Networks and ISDN Systems, Proceedings of the 7th International World Wide Web Conference, Elsevier, pp. 85 - 94.

[17] W. Godefroy, R. Meersman, O. De Troyer (1998), *"UR-WSDM: Adding User Requirement Granularity to Model Web Based Information Systems"*, In Proceedings of 1st Workshop on Hypermedia Development, Pittsburgh, USA.

[18] O. De Troyer, S. Casteleyn (2001), *"The Conference Review System with WSDM"*, In Proceedings of DASWIS 2001 workshop (attached to the ER 2001 conference), Yokohama, Japan.

[19] O. De Troyer, P. Plessers, S. Casteleyn (2003), *"Conceptual View Integration for Audience Driven Web Design"*, In CD-ROM Proceedings of the WWW2003 Conference, Budapest, Hongary.

[20] T. Halpin (2001), *"Information Modeling and Relational Databases"*, Morgan Kaufmann Publishers, ISBN 1-55860-672-6.

[21] M. Fowler and K. Scott (2000), *"UML Distilled, Second Edition"*, Addison-Wesley, ISBN 0-201-65783-X.

[22] S. Casteleyn, O. De Troyer (2002), *"Exploiting Link Types during the Web Site Design Process to Enhance Usability of Web Sites"*, In Proceedings of the IWWOST 2002 workshop (attached to the ECOOP conference), Malaga, Spain.

[23] O. De Troyer, S. Casteleyn (2003), *"Exploiting Link Types during the Conceptual Design of Web Sites"*, In International Journal of Web Engineering Technology, Vol 1, No. 1.

[24] "Visual Studio Home", http://msdn.microsoft.com/vstudio/.

[25] "DeKlarit turns Visual Studio .NET into a RAD tool for building database applications", http://www.deklarit.com.

[26] S. Mitchell (2003), *"ASP.NET Data Web Controls Kick Start"*, Sams, ISBN 0-672-32501-2.

[27] "OSCOM - Zurich", http://www.oscom.org/conferences/Zurich/.

[28] "CMS Review - Features, Resources, Demos and Downloads", http://www.cmsreview.com/.

[29] "Bitflux GmbH - Home - Bitflux", http://www.bitflux.ch.

[30] "Callisto CMS - Home", http://www.callistocms.com.

[31] "Apache Lenya", http://cocoon.apache.org/lenya/.

[32] "Content Management - TYPO3", http://typo3.com.

[33] N. Guel, D. Schwabe and P. Villain (2000), *"Modeling Interactions and Navigation in Web Applications"*, Proceedings of the World Wild Web and Conceptual Modeling'00 Workshop, ER'00 Conference, Springer, Salt Lake City.

[34] C. Kerer and E. Kirda (2000), *"Layout, Content and Logic Separation in Web Engineering"*, 3rd Workshop on Web Engineering, World Wide Web Conference (WWW9), Amsterdam, The Netherlands.

[35] G. Rossi, D. Schwabe and R. Guimarães (2001), *"Designing Personalized Web Applications"*, Proceedings of the WWW10, Hong Kong.