WAINE
Automatic Generator of Web Based Applications

A. Delgado, A. Estepa and R. Estepa
Escuela Superior de Ingenieros, Universidad de Sevilla
C/ Camino de los Descubrimientos s/n, Seville, Spain

Keywords: User Interfaces, Software architecture, Automated Tools.

Abstract: This paper presents WAINE (*Web Application & INterface Engine*), a system for quick web application development based on a novel architecture which provide multiple benefits like: zero programming, integrated security, high re-usability and many degrees of independence. The architecture is well suited for development of multi-user applications and is based on an abstract model which captures all the elements of a typical application. The sample applications developed validate the advantages of the proposed architecture.

1 INTRODUCTION

Reducing the development cost is clearly one of the goals of software engineering. However, customers are more and more demanding respect to software, imposing higher requirements in the application quality and paying special attention to user interfaces. This justifies the fact that the user interface is growing in complexity, meeting currently requirements like: platform independence, remote access, usability, multi-user support, clean design, etc ... This also implies complexity in the development, burdening it with more technologies and code. Therefore, it is a well known fact since early 90's that the user interface development takes a significant part of the overall development cost (Myers and Rosson, 1992).

The trend in interfaces development is the use of web technologies. This represents a important step ahead in meeting user's requirements like platform independence and remote access. In addition, companies also save deployment costs, version update methods, etc... However, the use of web technologies have also exposed developers to new problems:

- the increasing number of web technologies (e.g. HTML, XHTML, CSS, Webservices, etc..), XML (e.g. dtd, xschema, xslt, etc ..) and programming languages involved the client side(e.g. Javascript, EMACScript, Java, etc..) or server side (e.g. java, PHP, ASP, etc ..) have all contributed to increase the complexity of interface development and maintenance.
- The long learning curves and the difficulty of developers integration in projects are increasing the development costs.
- The applications implementations and coding has become chaotic in some cases due to the fact that some programmers embedded the application logic in the interface, disfavoring the reutilization and maintenance of the application. In other cases, companies take advantage of open-source projects that have multiple contributors with different coding styles and manners, which difficulties its maintenance.

Although some web-specific implementations of the pattern Model-view-Controller like Struts(Davis, 2001) or Spring(Arthur and Azadegan, 2005) have notably helped to overcome the previous drawbacks, still the learning curve of these frameworks (which are difficult to use without the help of specific development tools like those included in jdeveloper or netbeans) makes them unaffordable for many small-sized software companies.

Some of the traditional practices to achieve cost savings in the software industry are code reutilization and code automatic generation. Regarding to the automatic generation of user interfaces, some authors have proposed since early 80's means to autom-

Delgado A., Estepa A. and Estepa R. (2007).
WAINE - Automatic Generator of Web Based Applications.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Web Interfaces and Applications*, pages 226-233
DOI: 10.5220/0001279002260233

atize the interface generation based in the close relationship between the data model and the interfaces that users need. Thus, the early approaches generated automatically forms that allow the user interaction with tables((Arturo Pizano and Iizawa, 1993)) or objects((Branko Milosavljevic and Milosavljevic, 2003)). However, most of these first attempts((Olsen, 1989), (Arturo Pizano and Iizawa, 1993), (Christian Janssen, 1993), (Vigna, 2002), (Branko Milosavljevic and Milosavljevic, 2003)) lacked of interfaces that handled important application elements like: user management, security policies, menus, reports, etc .. In addition, most of these first proposals were not platform independent but offered a strong dependence with elements like operating systems or database systems.

Most recent approaches ((eGen, 2005),(JSenna, 2005)) success in automatizing the major part of a complete web-interface application however they fail to support important aspects of interfaces like complex forms where several entities (i.e. several related tables) are involved in the same form. Some of these approaches also lack of the customization that customers frequently demand. In addition theses systems are attached to their own development toolkit that prevents third parties to contribute or improve the development tools.

We have developed a new approach that aims to overcome all the problems previously mentioned. The outcome is a system named WAINE, a system with the features of: efficiency, independence, reusability and simplicity necessary to potentially increase the productivity of software companies that develop medium-complexity-level web-based applications. The architecture of WAINE is based in a model that captures the relationship between the main aspects of a typical management application (e.g. users, menus, forms, security, etc..) and the application development is based in a simple descriptive language.

2 SYSTEM REQUIREMENTS

The challenges that our system pursues are the following:

- *Independence*. This key feature encompasses other requirements like: OS independence, datasource independence (i.e. relational databases, XML files, text files, directories, etc...), webserver, browser and application specification tool independence.
- *Security*. Another key feature that should be included is at least: Access Control List (ACL) and different authentication methods (i.e. password files, LDAP, IP address, etc...). New authentication methods should be easily added to the system.
- *Web-based interface*. This requirement is basic to grant universal access and suppress installation costs to applications. Additionally, the application should be able to represent results with different layouts and issue complex widgets which include charts, images or multimedia data. New widgets and form layouts should be easily added to the system.
- *Zero programming*. The application development should use the minimum coding task possible. Ideally, it should lack of any code except for, obviously, a declarative language used to describe the objects of the application model, but minimal pieces of software like actions or events could be potentially coded in some programming language.
- *Customizable*. The applications generated must be easy to configure. From different look&feels for menus, forms, etc... to the configuration of data sources, form templates, internationalization, etc ... Also should be customizable for single users with particular needs (for example for accessibility issues). The engine must be easy to extend for new authentication methods, datasource types, form layouts, form widgets, etc...
- *Independence between logic and GUI design*. The system should allow the application-logic and graphic design teams work separately as done in (Puerta, 1997). This is a key factor in web-based development.
- *Efficient*. The run-time engine must be light and small enough to run in low-performance hardware or embedded systems.

3 THE APPLICATION MODEL

This section proposes a simple model to capture all the components of an application. We first describe each component of the model and then we explain the relationship between them using an ERD.

3.1 Model Components

3.1.1 Users and Groups

Users represent different ways of interacting with the application and allow the access control mechanisms necessary to achieve a secure system.

In our model, users are classified in groups that represent roles, and the access to different application functions or components is based on creating different menus associated to user's id or user's group. Additionally, ACLs grant the access of users or roles to the application's forms or actions. A user is characterized by attributes like a username, some authentication info (password or others), a reference to a main menu, a profile to customize the system to the user preferences and some account information.

3.1.2 Menus and Options

Any application holds a main menu with a number of options that allow users to access all the application functions. The menu layouts in a web interface are diverse (e.g. toolbar, link list, javascript or java built-in menus, etc ..) and should not be restricted by the system engine.

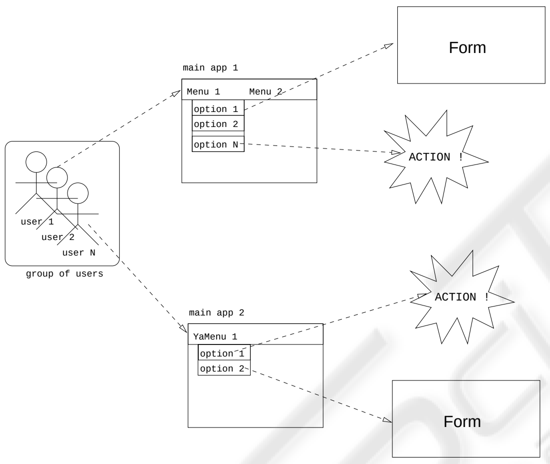


Figure 1: Relationship between users, menus and options.

As shown in figure 1, in our model, users have a main menu which is composed by a set of menu items. The menu items are composed by a set of options. Every option will launch a form, an action or just a URL link. Actions are used to perform application functions while forms are used to manage data.

3.1.3 Forms, Fields and Actions

Forms are used by applications to manage and represent data. Forms and the data model have a close relationship since forms are the application components that allow to visualize data (mainly by means of fields elements) and perform operations over the data (through action buttons or links). Usually one form represents data from one entity in a ERD diagram(Chen, 1976) (i.e. one table) but in our application model this should not be a constraint and the use of composite forms (i.e. a number of related single-forms) should be allowed to represent complex relations (Arturo Pizano and Iizawa, 1993).

The main properties of a form should include, at least, a data source, a set of fields, a set of actions, and a set of events. Additionally, different layouts could be used to place fields inside a form. There are some common layouts like combo, table, form, tabbed form, grid, etc ... but a programmer could define how widgets must be placed in a new kind form extending the system layouts. A simpler way to define how a particular form must be displayed is using form templates which, basically, indicate how fields are distributed for a specific form.

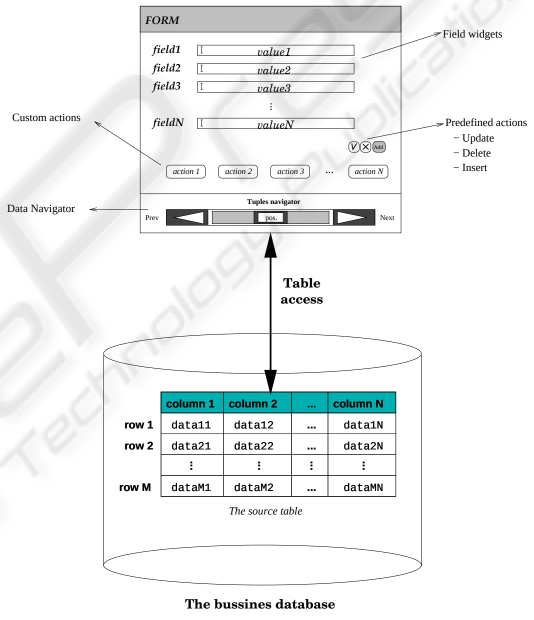


Figure 2: Forms.

Thus, one form can be related to a set of fields and to a set of actions. Next we address these elements in deeper detail:

- *Fields.* A field should be characterized at least by its type, its source, and the widget presenting it. The field source is the element of the data-source that the field represents, (e.g. one column of a given table in a relational database) while the field type represents the domain of the field value, i.e. the allowed values for the field. The source of some fields would be an expression with some other fields. They are called calculated fields. Finally, a widget defines how a field value must be presented and edited. Programmers should be able to write their own widgets to improve the interface usability modifying existing widget objects or extending the system widgets classes. Some field attributes like read/write or show/hide could be potentially customized resulting in different form views.

- *Actions.* Actions are user procedures triggered by either a button in a form or a menu option. An action can be also automatically be triggered before of after an event like the form creation, the form destruction, or data management actions. Regarding to the set of potential actions included in a form, at least it should include those stated in the so called CRUD model (Create, Retrieve, Update and Delete) but the set of potential actions should not restricted to these ones, but should allow user's defined actions and events.

3.1.4 Composite Forms

In many cases a set of forms are related between them and a single form is not sufficient to implement a useful interface. Composite forms (termed *structs* in our model) are forms which are themselves composed by a set of single related-forms in a structured way. Some other systems only allows the use of master-detail forms. This is a very useful form but not powerful and flexible enough for any situation.

3.2 Relationship Between Model Components

Finally, to summarize this section we could think of all the previous application components as a set of entities related between them. In fact, we could use a Entity Relationship Diagram (ERD) to represent our model. Figure 3 represents such a diagram.

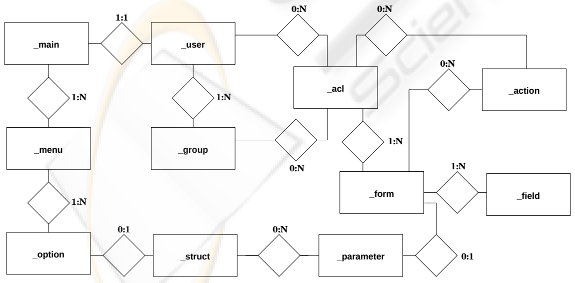


Figure 3: The Application's model.

According to the diagram of figure 3, we could potentially store all the application components in a database handled by a common engine to serve the application. This is the key idea behind the system architecture presented in next section.

4 SYSTEM ARCHITECTURE

The model described in previous section is implemented following an architecture that allows to meet all the characteristic expressed in the initial requirements. This section presents the architecture implemented, which is based on four basic elements as illustrated in figure 4:

- *The business database*(DB). It handles all the business information managed by the system. The system uses a extensible data abstraction layer that allows the connection to different data sources which allow datasource independence.
- *The application repository or Meta-Database* (MDB). Since the elements of the applications (i.e.:forms, structs, users, menus, forms, etc...) are a set, they can be represented in a relational database (meta-base) whose schema is presented in figure 3. Different database systems could be used to implement the Meta-Database , which would allow more flexibility and independence to the system.
- *The application engine*. The run-time engine accesses to the application Meta-Database to obtain the information for the application automatic generation (menus, forms, etc...) and access to the application database to perform the basic actions (create, update, and delete) and performs the user defined actions and events.
- *The configuration repository*. The system can be configured, customized (e.g. look&field) and extended in a easy way by means of setting a set of parameters in the configuration repository.

Next, we address in detail the main components of the system: the application engine and the configuration repository.

4.1 The Application Engine

The engine accesses the Meta-Database and automatically generates the menus and forms that will be send to the end-user by means of a web server. It also executes the actions selected by end-users like: insert, delete, update over the database or any user defined action or event and stored in the meta-base.

The engine is composed by a set of modules or sub-systems (e.g. menus, forms, reports, etc ...) which mainly use the services provided by two abstraction layers: data and render, in order to access to different data sources and to generate all the application objects and outputs. Additionally, the user and security modules provide access control and authentication services to the rest of the engine modules.

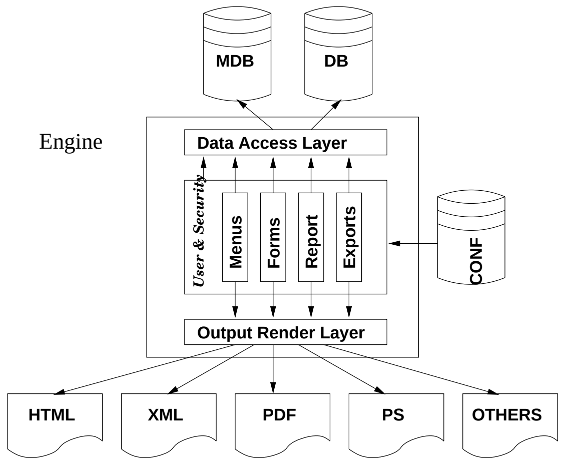


Figure 4: The system architecture.

The object access security is based in ACLs. The authentication methods used in the login process could potentially be extended to allow new ones. This is an important requirement for any system.

4.2 The Configuration Repository

The configuration repository allows developers to customize the application in many ways. The main aspects to be addressed are: database configuration, preferred authentication methods, application fonts and colors, form templates, etc ...

The users subsystem can be used to override values of some system variables declared in the configuration repository and customize the application for special users. This can be useful to customize locale, languages, color preferences, printers, etc...

Also some system functions can be redefined or extended parameterizing the system, extending system classes or creating new ones where possible. An example of this kind of extension are new widgets (extending the basic widgets classes or creating new widget objects), new form layouts (creating a new layout class), or new authentication methods. All this extensions are made in the configuration repository and allows the high system flexibility required in section 2.

5 IMPLEMENTATION ISSUES

This section addresses those aspect specific to the current implementation of WAINE. This allows to have a closer view of the insights of the application engine.

5.1 The Application Repository

As one of the main objectives in this system is the data source independence, a data abstraction layer is used to access to both: the application databases and to the system Meta-Database see figure 4. This allows that both databases are run in different database systems like csv text files, XML files or a number of relational database management systems, making our run-time more portable, flexible and independent.

5.2 The Application Engine

The application engine is the kernel of the system. It is designed using an Object-Oriented methodology and it has been coded using PHP. PHP is free and available for many platforms and webservers, it's our choice to build WAINE as an OS and webserver independent piece of software. PHP is also small enough to run in low-performance hardware as required in section 2.

The main implementation modules of the engine are:

- The data access layer is composed by two interfaces (i.e: *Data_source* and *Data_result*) which define an abstract driver for data access. Any system (or user defined) driver extends these interfaces and register itself as ready to be used. Current implementation includes native drivers for PostgreSQL, MySQL, SQLite, Firebird, cvs files, XML files, ldap directories, etc...
- The output render layer. Current implementation contains three rendering classes which represents three kinds of render: the plain render, the head-body-tail render (HBT), and the multiple render. A Render substitute widgets inside a form template which can be created automatically by a system-defined layout object like, table, combo, form, etc ... or can be directly specified for a single form by programmers. Form templates are used to allow graphic designers and programmers to work separately. Graphic designers can generate web pages for special forms (where graphic design if very important) while programmers can work using system predefined layouts to develop the application. When all the work is done programmers only tell the system to use the template (a web page with some special tags to locate fields inside it) made by graphic designers.

The run-time also performs system actions (predefined or user-defined). The predefined actions are those of the CRUD model, and of course, are data source driver dependant ones. Currently, the following types of user-defined actions are supported:

PHP code, native database code driver dependent (like SQL sentences ...), opening new forms, calls to external programs, or to external web pages, CGIs or web services.

5.3 The Configuration Repository

The configuration repository is a set of configuration files codified in PHP, defining system variables (for system configuration), functions or classes for system extension when necessary.

The user profile stored in the metabase is also a set of variables in a field-value type configuration. User profiles are useful to customize the application for a particular user or group.

6 POPULATING THE METABASE

The Meta-Database contains objects collection from the application model. Additionally, as it has been shown in the previous sections, several database formats can be selected for the Meta-Database , so a common way to define the application objects in every database type is needed. We have opted for defining the model objects in a XML based language named ASL (Application Specification Language). There are some advantages in the use of ASL:

- XML is a widely extended language and provides a set of facilities to parse the contents or to convert XML documents to a different formats like HTML, SQL, textfiles, etc... This will be useful to parse ASL or to translate a file written in ASL to the selected target database for the Meta-Database
- Developers are familiar with XML technologies. They haven't to learn new languages and new syntax rules, and they can use their preferred XML editors and tools to work with ASL.
- Using ASL to specify the application model allows developers to reutilize the code. Also they can use an ASL description file as a media to share design ideas and application concepts.
- The use of an ASL file to populate the Meta-Database , allows the system to be independent of the tools used to specify the application model. Currently a text editor (or XML editor) and a web-based RAD can be used to specify the application model and generate the ASL file. Also, any person interested in other methods to specify the application model can implement their own tools to do this work.

The syntax for the ASL language in dtd format is available at http://waine.us.es/DTD/WAINE/0.6/asl.dtd

7 DEVELOPMENT PROCESS

To illustrate the development process we have decided to create a full small system in this section. The system is a very reduced suggestion track system see figure 5. A secretary (joe) will receive suggestions by email, letter, telephone calls or sms and he'll fill a form in the application with the required information. The supervisor (sarah) can only fill the categories for people and suggestions type. In this reduced version of the application there are no reports or query forms.

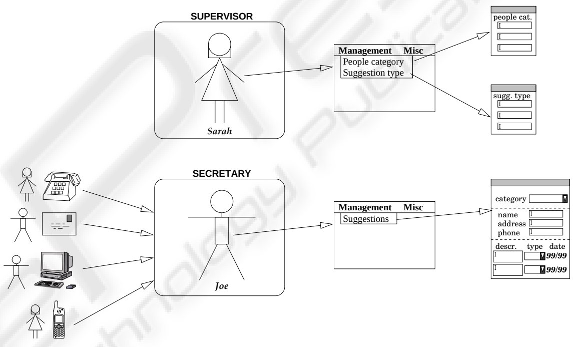
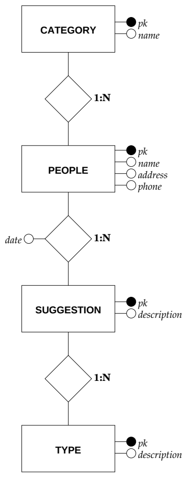
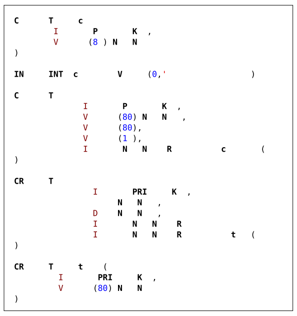


Figure 5: Suggestion system.

1. Create the application Database: The first step is to design and build the application database (DB). The conceptual design for the DB and the SQL creation script are presented in figure 6. The system must impose some restrictions.
   (a) The Default category (code 0) will be uneraseable.
   (b) A people category can't be deleted if some people belong to it.
   (c) When a person is deleted all his/her suggestions are deleted.
   (d) Suggestion types can't be deleted.

   We have selected sqlite 2.8.16 for both application DB and Meta-Database . Sqlite is a small, secure and fast SQL database engine which implements most of SQL92, but foreign key constraints are not enforced. We can create some database triggers to do this work, but we prefer to code some WAINE events to illustrate how WAINE can force this restrictions (a,b,c). The last restriction will be forced disabling the delete button in the form frm_type.

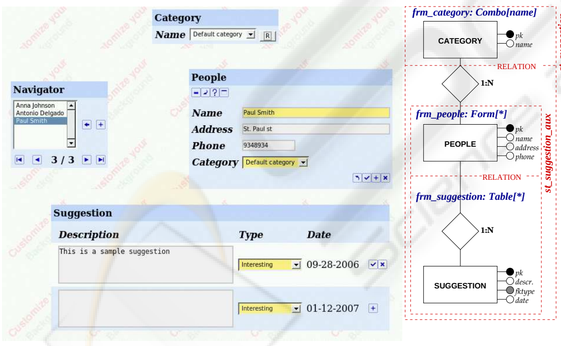




```
C      T     c
        I        P       K  ,
        V      (8 ) N   N
)

IN     INT  c        V      (0,'                )

C      T
              I        P       K  ,
              V      (80) N   N    ,
              V      (80),
              V      (1 ),
              I        N   N    R          c       (
)

CR     T
                I        PRI     K  ,
                     N   N    ,
                D    N   N    ,
                I        N   N    R
                I        N   N    R          t   (
)

CR     T     t    (
         I       PRI     K  ,
         V      (80) N   N
)
```

Figure 6: Suggestions Database.

2. Design the application model. Design groups, users and the different menus associated them. Of special interest is the design of forms and structs. In the figure 5, a small schema of this objects is represented. The system has only two groups (supervisor,secretary) with a single user each one (sarah and joe). The supervisor will have a main menu linked to two simple forms: category and type. The secretary will use a main menu linked to a composite form where he can select categories and manage persons and suggestions at the same time (see figure 7).



Name: Default category

Anna Johnson
Antonio Delgado
Paul Smith
3 / 3

Name: Paul Smith
Address: St. Paul st
Phone: 9348934
Category: Default category

Description	Type	Date
This is a sample suggestion	Interesting	09-28-2006
	Interesting	01-12-2007

Figure 7: The resulting interface for the secretary.

3. Write the application specification in ASL. A XML editor or a RAD like wbuilder (http://waine.us.es/demo/wbuilder) can be used to generate this file. In http://waine.us.es/demo/sample/code.asl the ASL for this application is presented.

4. Translate the ASL into the desired target Meta-Database format using the **asl2mdb** system tool. asl2mdb is a shellscript that uses the *xmlstarlet* command line XML toolkit.

5. Create a new application with the **mkapp** command utility. This will generate a new application instance linked to a run-time engine. The generated configuration repository contains default variables that would be customized.

6. Configure the application. Edit some config files from the configuration repository, at least (db.cfg and mdb.cfg). These files are used to configure the access to the application database and Meta-Database . Many other files can be modified if desired to parameterize the application (colors, titles, printers, extensions, etc ...). For example we can see here the configuration code for the default business database.

```
<?php
   $DBDRIVER='dssqlite.inc';
   $DBFILE='./DB';
?>
```

The sample application can be tested at http://waine.us.es/demo/sample.

8 MORE APPLICATIONS

We recruited a group of students who developed a number of small-medium size applications (see http://waine.us.es/demo) from which we stand out the following:

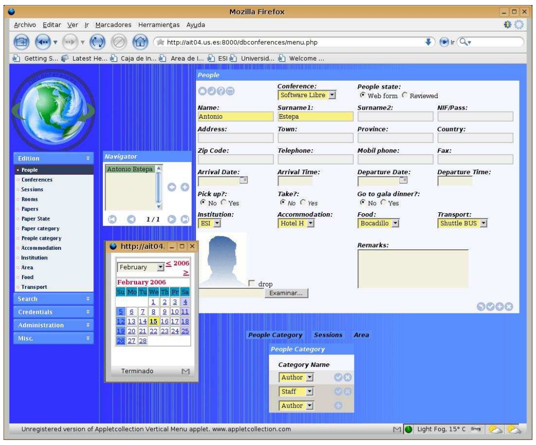
8.1 Conferences

Dbconferences (http://waine.us.es/demo/dbconferences) is a complete accreditation system specially designed to be used in large scientific events. The application manages all the logistic information related to attendants, reviewing process, papers, conference scheduling, accommodations, transport, categories, access areas, etc... It can also be used to print the accreditation cards of the people involved in the conference (lecturers, organizing committee, etc...).

The application considers three basic roles: administrator, user, and worker and every role has its associated menu.

8.2 Sysbook

System Book (http://waine.us.es/demo/sysbook) is a system to write documents in collaborative environments. The documents have a predefined structure with a title, an abstract, some section levels, etc...



People

Conference: Software Libre

People state: [✓ Web form] Reviewed

Name: Antonio

Surname1: Estepa

Surname2:

NIF/Pass:

Address:

Town:

Province:

Country:

Zip Code:

Telephone:

Mobil phone:

Fax:

Arrival Date:

Arrival Time:

Departure Date:

Departure Time:

Pick up?: [✓ No] Yes

Take?: [✓ No] Yes

Go to gala dinner?: [✓ No] Yes

Institution: ESI

Accommodation: Hotel H

Food: Bocadillo

Transport: Shuttle BUS

Remarks:

drop

People Category

Category Name
Author
Staff
Author

Figure 8: Conferences accreditation system.

which meet the basic requirements of technical documents. Conversely to a wiki, every part of the document has a user in charge. Currently the system supports documents in the following formats: LaTeX, DocBook, LinuxDoc, HTML, and XML.

Finally, we would like to remark that in our team, the learning period did not exceeded one month and the application development was done after 4 weeks after finishing the learning period. According to student's opinion, once they have already developed one application, they would be ready to develop a new one of the same level of difficulty in less than 2 weeks.

In figure 9 we present some indicators of the complexity of some applications developed by students.

Application name	Groups	Users	Main Apps	Menu options	Structs	Forms	Actions	DB Tables	Code lines
Sample aplication	2	2	2	9	4	4	2	4	163
dbBibTex	3	3/*	3	61	46	27	9	17	1152
Conferences	3	3/*	3	45	38	29	2	24	1126
Sysbook	3	3/*	3	20	22	9	3	8	692
Small Web Portal	3/*	3/*	8	170	82	35	5	21	2147

Figure 9: statistics for some applications.

9 CONCLUSIONS AND FURTHER WORK

In this paper we have presented WAINE, a system that serves web applications implemented from a single description xml file and a business database. The system presents automatically the user interfaces necessary to handle all aspect of the application: from user's access or menus to the web forms that interact with data. The architecture is based in a simple application model that addresses all the aspects of a typical management application. It presents independence at different levels and is flexible, being easily customizable in many ways. The experience acquired after several developed applications let us conclude that WAINE is a cost-effective alternative for a number of medium-complexity applications, achieving a short learning curve and development time and a high reusability of application specifications. We are currently working on the design of new tools to specify applications using visual interfaces or natural language.

REFERENCES

Arthur, J. and Azadegan, S. (2005). In *Spring framework for rapid open source J2EE Web application development: a case study*, volume ACM 1-58113-867-9/04/0500. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks.

Arturo Pizano, Y. S. and Iizawa, A. (1993). In *Automatic Generation of Graphical USer Interfaces for Interactive Database Applications*, pages 344–355. Proceedings of CIKM'93.

Branko Milosavljevic, Milan Vidakovic, S. K. and Milosavljevic, G. (2003). In *User Interface Code Generation For EJB-Based Data Models Using Intermediate Form Representations*, pages 259–262. IEEE International Conference on Automated Software Engineering.

Chen, P. (1976). In *The Entity-Relationship Model Toward a Unified View od Data*, volume 1, pages 9–36. ACM Transactions on Database Systems.

Christian Janssen, Anette Weisbecker, J. Z. (1993). In *Generating User Interfaces from Data Models and Dialogue Net Specifications*, volume ACM 0-89791-575-5/93/0004/0418. INTERCHI'93.

Davis, M. (2001). In *Struts, an open-source MVC implementation*, volume IBM developerWorks, February, 2001.

eGen (2005). e-Gen Group. http://www.egen.com.br.

JSenna (2005). JSenna project. http://www.jsenna.org.

Myers, B. A. and Rosson, M. B. (1992). In *Survey on user interface programming*. Proceedings of the ACM CHI'92 Conference on Human Factors in Computing Systems.

Olsen, D. R. (1989). In *A Programming Language Basis for User Interface Management*, pages 171–176. Proceedings of the CHI'89 Conference on Human Factors in Computing Systems.

Puerta, A. R. (1997). In *A Model-Based Interface Development Environment*, volume 14, pages 40–47. IEEE Software.

Vigna, S. (2002). In *ERW: Entities and relationships on the web*. Poster Proc. of Eleventh International World Wide Web Conference.