Upgrading an Academic IS From a Prototype to a Product

Filipe Silva*, António Cunha*, Manuel Machado*, Lígia Ribeiro‡ and Gabriel David¥

*FEUP, Portugal
{fsilva,cunha,mmachado}@fe.up.pt

‡IRICUP/FEUP, Portugal
lmr@iric.up.pt

¥FEUP/Inesc-Porto, Portugal
gtd@fe.up.pt

Abstract

Scaling-up a web-based academic IS developed in-house from one school to all the schools in the university implies upgrading the software from a prototype to a product. In the present case this has been done by revising the software in order to make it configurable in three directions: the definition of which functionalities are active in each school; the specification of local user access policies; and the design of a visual identity in the website. Keeping the set of local configurations updated through releases requires a specific migration tool.

Keywords: academic information system.

1 Introduction

The Engineering Faculty of the University of Porto (FEUP) is operating since 1996 a Web-based Information System (SiFEUP) developed in-house [1]. It has progressively covered many aspects of the academic information needs. All the steps in the pedagogic process are automated, from the creation of new programmes, curricula establishment, and teaching service distribution to the preparation of timetables, student registration, syllabus definition, summaries recording, and several course and programme statistics and reports. The research and development activity is also supported, with information on the funded projects and the record of all the publications produced. Several tools related to communication are available like a dynamic mail list management system, a web discussion forum, and a distributed news system with private groups. The system integrates information coming from other sources like for instance the human resources, the accounting and the library applications.

The system continues to evolve but the results already obtained have been very positively appreciated both internally to the Faculty and also by independent institutions. It has received the 1998 Prize for Administrative Modernization granted by the Portuguese Informatics Institute of the Finance Ministry and also the 2000 EUNIS Award of Excellence. As a consequence of this recognition, several Faculties asked for a license of the system and the Rector of the University of Porto, which comprehends 16 Organic Units (14 are Faculties), decided to promote its installation in the interested ones. At the same time the conditions for making the system available to other universities were created.

This decision represented a great challenge that can be summarized, from the technical viewpoint, in a simple assertion: the system should be upgraded from a prototype to a product. This paper presents the main developments which were required to accomplish this goal.

2 Requirements

To set up a new information system (IS) in an organization requires a lot more than installing hardware and software. Even for schools of the same university, as in the current case study, where the main rules are similar, there are different priorities and cultures that must be respected. Dimensions like understanding the user needs, assuring the strong implication of the management, defining adequate priorities for the activation of the several functionalities, and obtaining the cooperation of a significant part of the academic community are crucial for the success of the project. This approach, with institutions used to a significant degree of autonomy and willing to preserve their identity, very easily produces diverging requirements that, in the limit, would lead to a different system in each one. That could mean a best fit for each institution but such a solution has two main disadvantages: it is very expensive to maintain as that would mean a different prototype in each school, and it raises several difficulties to the possibility of coordinating information at the university level.

The alternative is to guarantee that the system is flexible enough to accommodate differences in three main aspects: the ability to decide which functionalities are made available in each phase, a way to establish a specific access control policy, and the possibility to build a visual identity by indicating some design elements.

So, a major revision of all the software has been undertaken to fulfil those requirements and produce a configurable

product. Other tasks have also been important, like improving the documentation via a contextual help system able to guide the user with a reduced intervention of the helpdesk staff, and improving the administration and data quality monitoring tools. Those, however, will not be addressed in this paper.

Actually, the genesis of the system has been very much influenced by the school where it has been first developed (FEUP). The prototype approach has led to a specifically tailored IS, which although embodying the main functionalities of a generic academic IS sometimes displayed some bias towards the FEUP particularities; the proper operation of the system relied to a certain extent on the presence of the development team, specially for those small tasks required once a year or once a semester, for which an appropriate interface had never been designed; and, above all, the user manuals and on-line helps lacked details because it was easier to call the development team and ask than search the answer by reading the documentation. Thus, the refactoring process has pursued a secondary goal of making the system more robust in the sense of being able to deal with a wider range of processes in an automatic way.

The name of the system has changed and it is now called SIGARRA, reflecting a new level of integration with the University Human Resources and Student management systems, offering more and more functionalities in a consistent way and bringing closer to the user a larger set of services that used to be very apart both from the physical and informational viewpoints.

3 Adding flexibility

The requirement that the system possesses the ability to dynamically decide which functionalities are made available in each phase is important in two folds: it tackles the differences among institutions as well as it allows a tight control of the evolution of a single institution.

The main step to a robust and scalable solution to this problem has been the inclusion in the runtime system of metadata describing the system's own structure and contents combined with the current status of the specific instance.

The whole system has been organized in modules and pages, with identification. Most pages have a similar structure (see figure 1): a common header, a left menu corresponding to the main areas, a central area for the current page, and a right options menu with links to related pages. A selection of the most recent news and highlights is usually displayed.

Which links are presented in the general left menu, i.e. which main modules, and which options are available in each page, as well as the corresponding labels, are easily defined without programming by using a configuration facility called GESSI (IS Management). This way, the pace of growth of a specific installation, from the user viewpoint, may be controlled by an information manager.

Figure 1: A typical page structure.

The (meta)data model supporting the configuration information (see figure 2) includes the main concept of **Application**, a largely autonomous software artefact, composed by **Modules**, which are logical units supporting major functions of the IS. A **Module** combines a number of **Pages**, each with a specific purpose and corresponding to a conceptual unit of interaction with the user. This is why the **Page** is the central concept in the model, as it organizes the contents of a specific instance of the IS and constitutes the basic item of the access control policy.

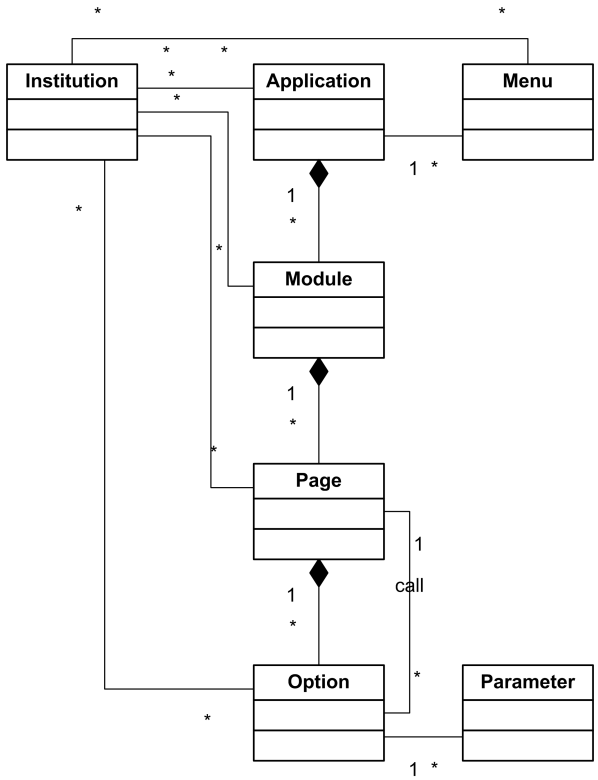

Figure 2: Schema of the data dictionary.

Which applications, modules and pages are available to a specific instance of the system at a certain moment is registered by the associations between the corresponding classes and the class **Institution**. It is therefore straightforward to control in each application which modules are in force and then fine tune the set of pages that are active. To serve as a reference for the whole system, there is a generic institution "SI" that includes all the modules and all the pages which are part of the current standard version.

The instance information manager is entitled to specify just the differences with respect to the reference institution, which is taken as the default. The differences may be declaring a page inactive or even to add a new page specific to the institution and not included in the standard distribution. In such case, the page identification clearly states the institution where it is meaningful, preventing it from overloading the standard distribution with special cases. This possibility should nevertheless be used with parsimony to ease the future maintenance work and developers should always try to assess whether a new idea initially developed in one institution is valuable to others and thus deserve a fuller treatment and posterior inclusion in the standard distribution.

The left menu is designed as a stable element in the interface to give direct access to the entry points of the main modules. Its composition is controlled by the class **Menu**, also constrained by the **Institution**. On the right of most pages there is a second set of navigation links, but this is very dependent on the current page and seen as a set of complementary options. Which links are actually displayed in each page is established in the **Option** class, which is a component of **Page** and constrained as well by the **Institution**. Both in the left menu and in the right options menu it is possible to specify the actual labels used. Two languages are supported and it is possible to have different configurations depending on the language.

To accomplish this variability, every page must be built dynamically, according to the current definitions. So a page is in fact the result of calling a procedure written, in most cases, in PL/SQL, the procedural language of Oracle. This approach had been followed before [2] in pages including the display of database query answers. Typically, the execution of such a procedure accesses the database, executes some required logic, and formats its output as an HTML page. The alternative technique, appropriate for static contents not fetched from the database, has been to display directly an HTML page stored in the file system. However, the improvement in configurability described above requires the ability to assemble the menus at runtime, thus pushing all the pages inside the database regardless of the nature of their primary contents. The name of the procedure that builds the HTML is used as the page name and one of the things it does is to call a support procedure to assemble the current menus for the specific page, from the information in the data dictionary. Besides selecting just the active options and displaying the chosen labels, it uses the second association (*call*) from **Option** to **Page** to know which page must be called when following the link in each option. This can be specified by the information manager, who gets another way to control the behaviour of the software.

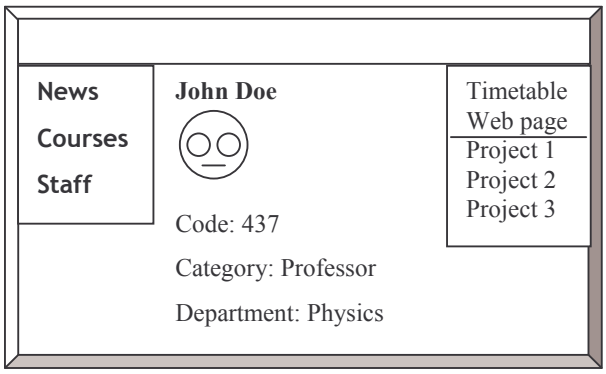

Figure 3: Example page.

When the page (the procedure name) to be called by a designated option is known at configuration time, it is stored in the data dictionary and used directly. Hence, this is the direct type of link. See in figure 3, an instance of page *professor.display* where the option Timetable is a call to the procedure *timetable.display*.

Sometimes, the actual link is not known until runtime. This is the case of the option *Web Page* meant to navigate outside the institutional site to a private personal area. As the URL of such a page is an attribute in a table, where it can be updated at any time, it represents a different type of link, designated parameter because, at configuration time, one just records the name of the parameter, a kind of global variable that, at runtime, must be filled with the current value of the URL in the DB.

A third type of link is called SQL. In this case, the programmer wants to present as a set of options the result of an SQL query, for which the length of the result set is not known beforehand. For instance, the links to the active projects come from a query stored as an attribute in **Option**. Modifying this query enables the fine tuning of the list of options.

There is a single *timetable.display* procedure to visualize the timetable of any professor. Which one to pick, the procedure gets it from a named parameter that the code for the calling page is in charge of providing. The names of the parameters, for instance *P_code*, to be used in a concrete option are stored in the class **Parameter**. So the URL associated with the option *Timetable* is *timetable.display?P_code=437*, where *timetable.display* comes from an option linked to page *professor.display*, *P_code* comes from a parameter linked to that option, and *437* is the value of variable *P_code* in the procedure *professor.display*. This is how the timetable of that specific professor is visualized.

There is still a last missing point in the presentation of this architecture, related to its ability to deal with exceptions. The mechanisms just described that puts an option link in the *professor.display* page, accessing his timetable is personalized but appears in the pages of all professors. Suppose a specific professor, for some reason, refuses to have his timetable displayed on the IS and the management agrees with this requirement. There is in **Option** an attribute where the information manager can specify an extra condition that the option must fulfil in order to be displayed. In this case, the condition would state that the professor id should be different from the id of that specific professor.

The ability of the a page that should look the same for a whole class of entities (professors, courses, rooms, etc.) to display specific behaviour depending on the concrete individual can be used to hide options, as in the above example, or to add options, for instance to present a temporary committee existing only in a specific programme.

The conclusion is that the current version of the system is highly configurable with some features useful for an information manager but some more adequate for DB administrators or programmers.

4 Controlling the access

It is possible to use the system in an anonymous way and access many resources but, if the user is known to the system, a personal bookmarks page is added and several options may become available in the right menu, according to his responsibilities in the school. For instance, a professor has different capabilities than a student regarding a course page, and programme directors and administrative staff members are granted different accesses.

Web-based systems follow basically a single page connection protocol. So, there is the need of controlling the access before the display of each page. That is done by verifying whether the user calling the **Page** belongs to the validation **Group** associated to it (see figure 4). If the user is not authenticated, only pages without validation group are displayed. To avoid repeatedly asking for the authentication, a session mechanism has been developed that keeps a session active since the user authenticates and while the browser instance is open.

The composition of a **Group** is the reunion of the persons defined in all its **GroupDetail**s. A **GroupDetail** may name the **Persons** in it or specify the quality that belonging **Persons** possess. In other words, it may have **Persons** (students, teachers, or staff) directly associated or specify a SQL query which result set contains the allowed **Persons**. For example, a group *CSYearlyReport* associated with the pages of the Annual Report of the Computer Science Programme may be the union of one detail for the query giving the set of teachers of the programme with another detail for the two or three members of the staff helping with the report. There is a third way to add **Persons** to a **Group**, by indicating previously defined **Groups** (association **Parent**) and inheriting the **Persons** in them. This promotes structuring the roles represented by the groups and reusing them, a crucial facility to keep such a complex system manageable and support access policies.

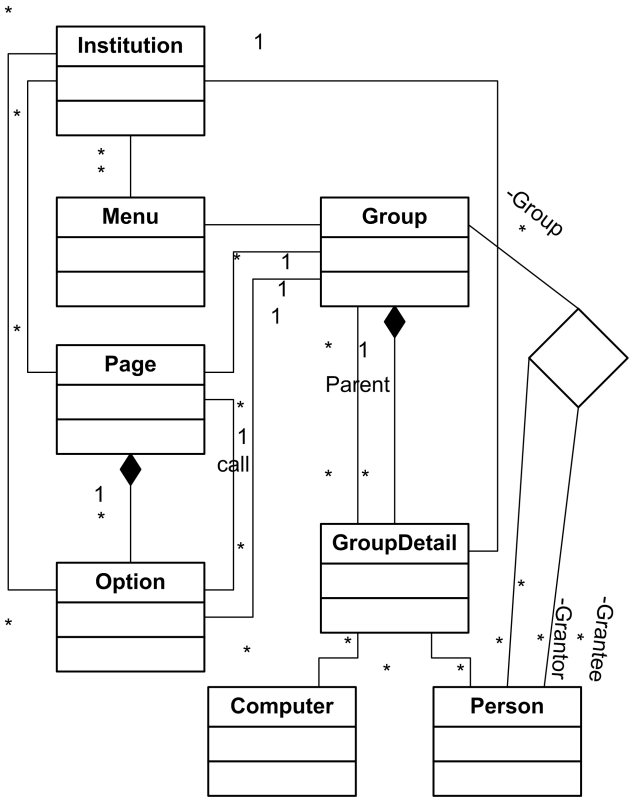

Figure 4: Schema of the access control.

Each **GroupDetail** can be constrained by a set of computers, also enumerated or defined implicitly by a SQL query, thereby adding another level of security. Certain operations may be required, for instance, to be performed in the computers of the central services. This possibility has been used mainly in the printing control module.

The basic access control mechanism relies on the notions of **Page** and **Group**. However, in figure 4, there are two more associations involving **Group**: one with left **Menu** items and the other with right menu **Options**. The idea is that, in addition to forbidding the access to the pages to non-authorized persons, the system may not even show the menu items or the options linking to such pages. The result is to avoid cluttering the screen with options that are not accessible and to reduce the feeling of "false promises" that certain labels could cause. This means that the access control facility is intimately connected with the construction of the actual pages presented to the user, acting as a second filter on what is displayed.

The assembly algorithm for page P now includes:

- Check whether the user belongs to the group of P and block the access to P if not;
- Get the header row of the application of P.
- Get the left menu items of the application of P;
- For each item, check whether the user belongs to the group of that item and display it if it does;
- Call the procedure that generates P;
- Get the options of P;
- For each option O
    - Check whether the user belongs to the group of O and skip it if not;
    - Check if there are conditions which must be satisfied to display the button and that they are indeed satisfied;
    - Get the parameters of O and assemble the link to associate with O;
- Display P.

There are several criteria to decide whether or not to include an option in a page or, if included, whether it can be followed by the user. Some examples of reasons to include persons in an authentication group follow:

- Category: student, teacher, staff; typically an attribute of the corresponding record;
- Responsibility: department or programme director, staff in a certain service, etc.; usually represented in an association between the organizational unit where the responsibility is effective and the person in charge;
- Arbitrary criteria: any condition that is expressible as a SQL query which selects all the users satisfying the corresponding filter, like being a teacher of the course under consideration, or the student whose academic record is being called;
- Individual assignment: specific users can be added to a group;
- Secretary: a user is able to grant his own capabilities with respect to a module to another designated user so it may act as his surrogate, enabling the sharing of work in a controlled manner.

The ternary association between Group and Person is the core to support this last example. It records the situations where a person (grantor), belonging to a certain group, explicitly delegates that capability to another person (grantee). The grantee may then acquire the rights of the grantor with respect to the pages and options indicating that validation group. These rights are added to his own rights so he may proceed with his work as before. An example of this is a professor who wants to have his class summaries typed by a secretary, without giving him his own password.

There is a special group named *all*, meaning any authenticated person. It is used wherever anonymous access is not acceptable as is the case with the dynamic mail module, which may require the ability to identify the source of a message.

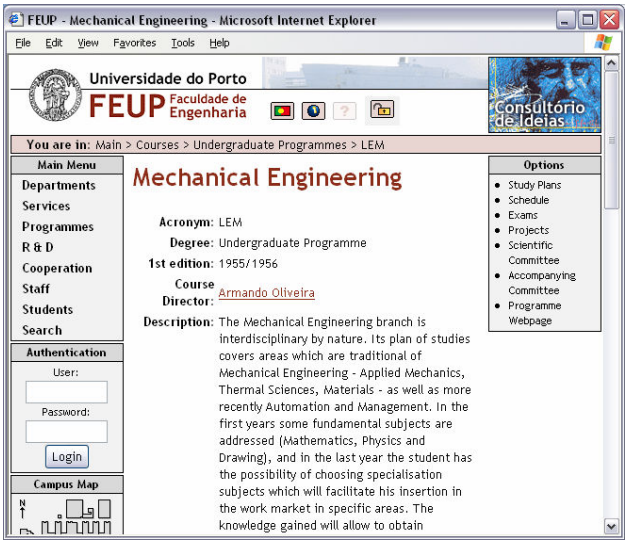

Figure 5: Page as seen by anonymous users.

All the above mentioned mechanisms are available in the configuration facility but the more sophisticated require certain knowledge of the system.

The final result is that different users see somewhat different versions of the pages in a manifestation of personalization. If the user is the director of a programme, when opening the page of that programme, he will see a lot more options than when opening the pages of other programmes. See in figures 5 and 6 the same page as seen by an anonymous Internet user and a member of the programme board, respectively. The latter includes some administration options, not available in the former one.

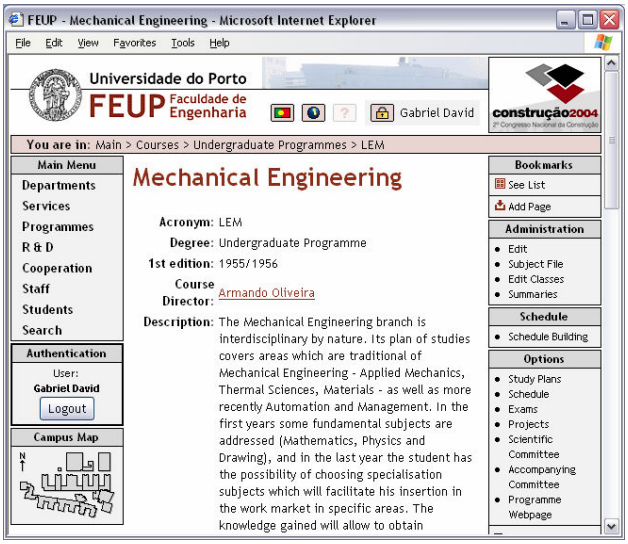

Figure 6: Page as seen by a member of the programme board.

Some complaints have been reported regarding this policy, from users that get lost in the interface. When using the system, but having forgotten to authenticate themselves, they cannot find some options and erroneously conclude that the options do not exist. For this reason, some especially important options may be shown even to users not allowed to follow them, just to mark that the options exist, the problem is with the user.

Another personalization element identifiable at the top right of figure 6 is the option *See List*, which displays a set of personal bookmarks that may be collected from any point in the system or outside it and are kept in the server and so are available at any Internet access point.

The complexity of the task of dynamically assembling a configurable page is even bigger than it seems because, besides incorporating a flexible access control policy, the system includes user dependent components. However, the processing overload introduced is usually very small or negligible from the user viewpoint, except in situations of very high load on the server.

5 Preserving identity

The mimicry between an organization and its IS, especially when it is available through the Web and thus has an important role in the external image of the organization, requires the ability to impress a visual identity that makes that IS somehow unique. So, an approach too rigid from the viewpoint of graphical presentation is not adequate for a system willing to serve a variety of institutions.

The third dimension of flexibility required for a configurable system is the ability to adapt the visual design to the image of the school, without ruining the overall goal of keeping a single version of the software.

The compromise found is to keep the overall organization of the page fixed but allowing many display elements to be changed like colours, fonts, images, and in general many aspects that can be specified in a Cascade Style Sheet. This is possible because the CSS actually used can also be changed with the configuration facility. This way a consistent design can be specified and simultaneously turns easy to substantially change the look of the site.

The separation between the contents, both static and dynamically generated, and the form, i.e. the display attributes, is a good design principle [3] that widens the applicability of the system.

As can be seen in figure 7, it is possible to associate a style sheet (**CSS**) either to an **Application** or to a **Module**. Style sheets are composed by **Tags**, which are specific to **Institutions**. As usual, there is a generic institution specifying all the relevant tags, which serves as default. Style sheets are recursive, allowing the definition of sub-sheets, each one contributing to the actual style to apply to a **Tag**.

The definitions relevant to the pages in a concrete **Module** or **Application** are combined in a virtual sheet at page generation time and sent to the browser in the usual way, without passing the stage of being a CSS file stored in the file system. The URL of the style sheet indicated in the header of the page to be displayed is actually a call to the procedure that generates a virtual CSS specifically for that page. Changing a style element is just changing its definition in the DB.

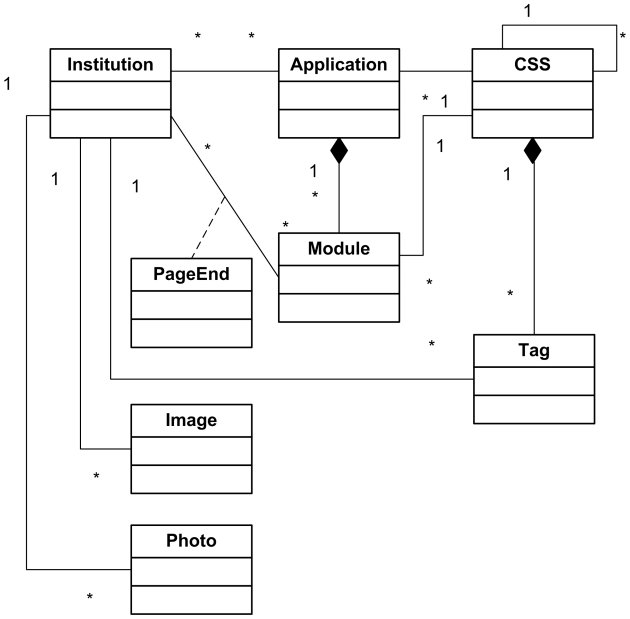

Figure 7: Schema of the visual configuration.

An example of the result of applying different style sheets to the same page can be seen in figure 8, to be contrasted with figure 1, in a different institution.

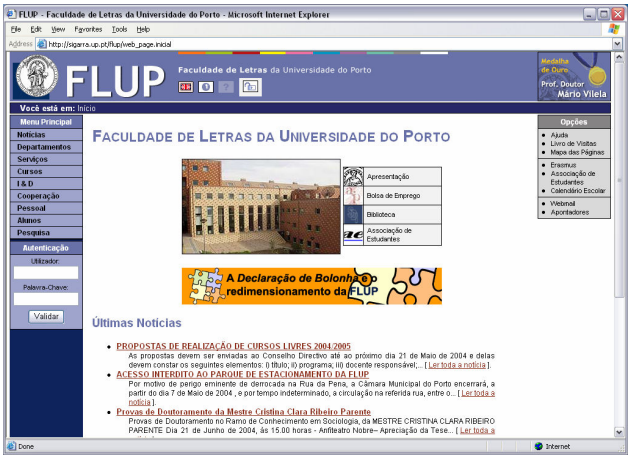

Table 8: Initial page with different style sheet (see figure 1).

Images and **Photos** are important elements in the design of a page. All of them can naturally differ from one **Institution** to another. **Photos** are collected in a portfolio meant to present

the institution. The set of **Images** include icons, symbols, and other graphical page components.

One of the more visible components is the highlight square, always present at the top right corner of the page. It draws the user attention to the news considered more relevant in each day. If there is more than one, the corresponding images rotate on a periodic basis.

Another use of images is shown in figure 8, immediately below the school photo. This banner component is not present in figure one, the entry page of another institution.

Some designs include a recurrent footer in every page. It is possible to specify such an element, different for each **Module** and **Institution**, in **PageEnd**.

6 Updating the software

The amount of information under the control of the configuration facility is large enough to justify the inclusion in the system distribution of a basic configuration that can be progressively tuned to the target school. However, the problem of preserving this tuning work through new releases of the software required the development of a migration tool that compares the upgraded basic configuration with the current installed version and helps on managing the updating process.

7 Conclusion

The temptation of replicating the prototype approach in each school of the university or in other universities would be unmanageable. Although at first sight that could mean a best fit for each institution, in fact it would be so expensive to maintain that the most likely situation would be a set of different systems having difficulty to benefit from the generalizations of improvements introduced in one of them and subject to a lengthy process of dissemination of new versions.

On the other hand, using the same model in the several faculties has the important advantage of making the information coordination easier at the university level. Detailed data can be obtained through a mechanism of redirecting the user to the faculty level systems, but the overview, aggregated data, and generic communication facilities are directly accessible in a consolidated system for the whole university.

The architecture presented in this paper guarantees that the system applicability does not get impaired by an unexpected requirement in a specific institution, that it is able to answer elegantly to the variability in IS contents, access policy, and display style and that it induces the sharing of new developments by all the installations.

In conclusion, the growing crisis that generalizing the IS from one school to the whole university represented, forced an upgrade in the system that made it more flexible and manageable for everybody, including the original installation. Nowadays the system is online on nine of the Organic Units of the University of Porto, other three following in the near future.

References

[1] Gabriel David, Lígia Maria Ribeiro. "Getting Management Support from an University Information System", *Proceedings of the European Cooperation in Higher Education Information Systems, EUNIS99*, Espoo, Finland, (1999). http://www.fe.up.pt/si/file_get.publ_artigo?p_id=6898 (2004-05-01).

[2] Filipe José Silva, Armando Jorge Oliveira, Lígia Maria Ribeiro, Gabriel David. "Searching Dynamic Web Pages With Semi-structured Contents", *9th International Conference of European University Information Systems, EUNIS 2003*, Amsterdam, The Netherlands, (2003). http://www.fe.up.pt/si/file_get.publ_artigo?p_id=9269 (2004-05-01).

[3] Jeffrey Zeldman. "Designing With Web Standards", New Riders, 456p., ISBN 0735712018, (2003).