A Logic-Based Information System

Salvador Pinto Abreu

Departamento de Informática, Universidade de Évora and
CENTRIA (FCT/UNL)
Portugal
spa@di.uevora.pt

Abstract. In this article we present the University of Evora's Integrated Information System (SIIUE), which is meant to model most of the information necessary for the management and day-to-day operation of an institution such as a public University. SIIUE is centered around a logic-based representation of all intervenients and processes, which is used to generate the more efficient and specific representations for the actual use. This includes extended SQL, PHP3 and Java code generation. SIIUE also interacts with an etherogenous set of partial information systems, both to supply and collect information.

Keywords: Information Systems, Logic Programming, Object-Relational Databases, Deductive Databases, Web Interfaces.

1 Introduction

With the increasing pervasiveness of networked computing resources, people come to expect that a growing number of services be available on-line. This applies to all sorts of usages from prospective and actual students, researchers, faculty and staff.

The existing information repositories at our university were very scattered, hiding many inconsistencies and redundancies. These problems were exacerbated because of the lack of network-awareness of these systems. This situation becomes painfully obvious when, for instance, faculty members are asked by several different sectors of the administration for the same piece information, albeit with slight variations. This particular situation was at the source of much institutional dis-functionality, as faculty members would frequently fail to produce the information that was required of them, purporting that they had already given it out.

From the decision-support point of view, the availability of a general-purpose information repository is desirable, structured so that that complex queries can be made pertaining to the whole system. The full benefits of the availability of the information can only be reaped if it is integrated.

Having a Logic Programming representation for an information system goes a long way in allowing for maximal flexibility in expressing concepts as well as in providing a natural specification language for complex queries, as the full power of first-order logic may be employed.

It is our belief that, in an actual implementation, the benefits of Logic Programming can indeed be made effective, by allowing incremental extensions to be developed with relatively little work, when compared with other methods.

In section 2 we introduce the overall system architecture, of which some components are further detailed in section 3. In section 4 some applications of SIIUE are briefly described. In section 5 compares the approaches used in SIIUE to others. In sections 6 and 7 we discuss our experience of actually using this system in the day-to-day operation of our University and propose directions for on-going and future development.

2 Architecture

The main goals for the development of SIIUE are:

1. To promote a declarative approach to the definition and manipulation of the structure and data in the information system.
   This need, together with the requirement that the system be easily prototyped, led to the use of a Prolog-based system both as a modeling language and as the source for the generation of further components.
2. While allowing for multiple forms of access to the information, a single canonical representation should be maintained for all items to be represented, in order to ensure consistency.
   An implication of this approach is that, while there may be several representations for some piece of information, each suited for one specific use, SIIUE must provide mechanisms to ensure consistency and guarantee that every change maps back to the canonical representation.
3. Provide means to represent both structured and unstructured information, tying them up when appropriate.
   The structured information component itself has two classes of information: that which is essentially immutable (for instance the departmental structure of the University) and the information that is subject to change, for instance the students' grades, the researchers' publications or the registry of scientific and cultural events.

We now proceed with a description of the main components of SIIUE, namely the Logic Programming dialect DL in section 2.1, some features of the generated SQL in section 3.2, the generated PHP3 code in section 3.3 and the Java code in section 3.4.

2.1 The Logic Description Language DL

To describe the system's classes, a modeling language can be used. Instead of resorting to an existing modeling language such as UML [1], we decided to implement a new such language, based on a first-order logic description of classes and inheritance, attributes and the values used to populate the classes. We feel that this approach provides more flexibility at the meta-level than would be feasible with other tools. Presently the only form of programming is textual but there is on-going work geared towards developing a visual programming language to satisfy the same design goals (see in section 6).

The approach chosen involves a stylized logic program where the relation between the entities is described, in an object-oriented fashion. The entity taxonomy used for the information system are introduced as a type system with inheritance.

For example, suppose we want to model the hierarchy described by figure 1.[1] The (simplified) correspondig DL syntax could be that given in figure 2. As can be gathered from this example, the DL syntax is fairly simple and can be expressed as Prolog terms with a few operator definitions.

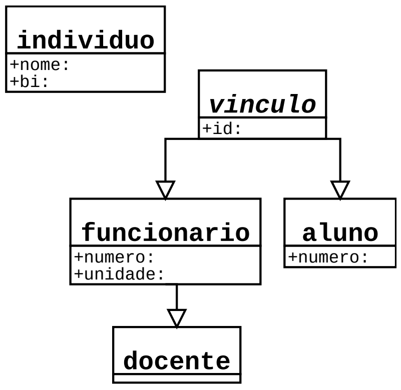

Fig. 1. Sample hierarchy

From the logic description (DL) specification, several processings can be made to generate different types of derived operating components:

1. An SQL file with all the instructions necessary to construct an object-relational database which corresponds to the information described in the DL source. This can map to several database backends, but is currently only implemented for PostgreSQL [6] as it already provides the necessary concepts, namely built-in inheritance. Future versions may interface to another database such as Oracle 8, which also has object-relational features.
2. A set of PHP3 [5] class definitions, to be used for web page construction. PHP3 (PHP Hypertext Processor) is an embedded scripting language for web servers.

[1] This example is a fragment of the hierarchy actually used in SIIUE.

```
class individuo: entidade.
 field nome. attr unique.
 field bi.

class vinculo.
 abstract.
 field id: int. domain individuo.id.

class funcionario: vinculo.
 field numero: int.
 field unidade. domain unidade.sigla.

class docente: funcionario.
 field carreira.

class aluno: vinculo.
 field numero: int. attr unique.
```

Fig. 2. DL syntax for the hierarchy from figure 1

3. A Java package, basically a collection of Java classes which mirror the DL hierarchy. These are fitted with constructors which interface to the SQL database previously generated using JDBC. The goal is to simplify the construction of applications which make use of the information system.
4. A document describing the class hierarchy, with comments on every class' usage, on the attributes that are used as external keys, etc. This is especially useful in describing the actual system to non-technical persons.

The derived components are detailed in sections 3.2 through 3.4, with particular emphasis on the "programming components".

2.2 Organization

In order to meet its stated goals, the design for SIIUE makes the following assumptions:

1. The authoritative representation is the Logic (DL) one.
2. Updates to data (instances of classes) are performed through any interface to the (object-)relational database.
3. Changes in the schema can only be performed at the DL level, possibly through the use of more sophisticated tools such as structural editors.

3 Implementation

This section describes some of the issues involved in implementing SIIUE. These have mostly to do with translating one representation into another.

Section 3.5 addresses the issue of ensuring the global coherence of all representations and section 3.7 deals with the aspect of coordinating the information held within SIIUE with external representations thereof, contained in other, more specialized, systems which do not directly rely on SIIUE's information.

3.1 Preprocessing

All the derived components are generated from the DL representation with a translator written in Prolog. This approach provides a large degree of freedom at a stage where the exact syntax and semantics of DL are still in a state of flux.

The logic description language is preprocessed into several operational forms, targeted at different uses:

– An SQL database. This will subsequently be used in the actual applications to perform the work of accessing the information.
– Components for other programming languages. These will mimic some of the information described in DL, but in a form more suitable for specific usages, namely the construction of WWW interfaces. This issue is further discussed in sections 3.3 through 3.4.

This process is presently carried out statically: the DL source is loaded into a Prolog processor which then runs several different queries which will produce the different derived programs.

3.2 SQL Generation

The classes introduced in section 2.1 are preprocessed into some dialect of SQL. In the implemented prototype, we're using PostgreSQL because of its object-oriented features, basically inheritance.

PostgreSQL, as of version 6.4, has some restrictions as to the subset of SQL92 it implements, in particular it does not allow foreign keys. This and a few other restrictions can be worked around by generating the appropriate domain-checking functions and constraints.[2]

The approach of using a logic description pays off especially well in the situation of having to generate SQL because:

– Classes may be directly translated to tables with inheritance in an object-relational database such as PostgreSQL but, should we opt for a traditional relational database without inheritance, the accessibility of the Prolog description of the hierarchy can be used to produce a flattened version of the tables, possibly using views to (artificially) simulate superclasses.

[2] This approach could be kept even in the situation where the database can provide foreign keys, because it allows for a very fine-grained control over what is actually performed as an SQL constraint.

- Issues such as dependencies between otherwise unrelated classes (eg. by means of integrity constraints) which may require a given class to be defined *before* it is referenced, may be dealt with by performing a topological sort on the *references* relation between attributes of different classes. This approach is particularly simple to implement in a language such as Prolog.
  A class A (which will map to an SQL table) is said to depend on another class B if either one of the following conditions is true:
  - A has an attribute whose domain is in the range of an attribute from any superclass of B, provided A is itself not a subclass of B.
  - A is a subclass of B.

  This information can then be used to initialize the database but is also important when it is necessary to rebuild it partially or entirely.

3.3 PHP Code Generation

Universidade de Évora adopted PHP3 [5] as a scripting language for its dynamic web pages. PHP is used extensively to construct server-side applications with access to the data and structures available through the information system.

As the structure of the information system is described as a class hierarchy which can be altered, it is important to keep this form of access synchronized with the definitions actually in effect.

From each class declared in DL, a corresponding PHP3 class is created which follows the DL inheritance structure, provides each attribute as an instance variable and defines a few automatically configured methods, which include:

- A constructor method, which can optionally behave as a query to the underlying database in which the query result is used to initialize the class instance. Access to the database is performed using the corresponding PHP library, for PostgreSQL it's the `pg_*` functions.
  The constructor method may be directed to perform the database query over the set of all subclasses of the given class or to stick to the instances of the class itself only.
  The queries performed are restricted to simple ones, in which all the attributes of the table are fetched, allowing for an optional `where` clause.[3]
- Traversal methods (`more()` and `next()`), which provide an interator construct for use in PHP scripts, similar to a cursor. These are useful for instance to produce listings or selection lists.
- Update methods (`update()` and `insert()`). These map to their SQL counterparts and are used to alter the data in the underlying database.

The generated PHP code relies upon the availability of database access functions. In the prototype implementation, the database is PostgreSQL but PHP3 provides access functions which allow for other database engines to be used.

[3] In fact it's a suffix of the SQL `select` query which may be specified, thereby allowing slightly more complex queries to be constructed.

Another category of PHP code that is generated are components for the construction of user interfaces. Initially this includes methods which generate a list-box (or a pop-up menu or two list-boxes) for selecting one or more items from a set, defined by the instances of a given class.

3.4 Java Code Generation

Generating Java classes is not much different from generating PHP classes. The advantage of these is to allow for more client-side work to be performed, thereby relieving the server system from the duty of having to parse long declarations for each page, as happens with the PHP code.

The access to the database component is acheived through JDBC.

Besides creating a Java class for each DL class, the java code base includes classes which use the Swing [3] toolkit in order to create interface components which can be used throughout a wide range of different applications.

With the perspective of having a Java constraint solver, [4] it will be increasingly interesting to utilize the Java interfaces as these will be endowed with a powerful computational mechanism, akin to what can be done in DL.

3.5 Synchronisation

The issue of maintaining the information in its various representations (the database as well as the Logic Programming versions) coherent has to be addressed carefully because the information system (both its schema and the data therein) can be updated frequently.

The main form of updates is to the database, with incidence on a limited set of relations. Each update made must be reflected on the logic representation. This goal can be acheived in one of two ways:

1. An "off-line" method, where the database is periodically queried for changes and these are translated by an external program into logic form, to be subsequently integrated into the logic representation.
2. An "on-line" method which requires a direct SQL/Prolog interface.

In either case, and considering that the updates are performed exclusively on the database[5] it is reasonable to have the updates at the logic level initiated from updates and insertions and deletions at the database level.

The relations which can be updated are explicitly marked as such at the DL level. For these, the SIIUE engine will generate SQL statements which ensure that:

– The tables may be updated.

[4] We are presently developing such a system.

[5] The logic representation is presently used to rebuild the database and construct the operational interfaces described in sections 3.3 through 3.4. Changes at this level are not tought of as "updates" because they may be arbitrary.

– Changes get propagated back to the DL level.

With PostgreSQL this can be implemented via the trigger mechanism, in which auxiliary tables can be used to indicate what changes were made. These can then be read back to DL in order to guarantee that both representations are consistent with each other.

With the on-line approach, the trigger-based architecture remains applicable but the method used to convey the changes to the Logic Programming representation can be different: the DL system is made to appear as a *procedural language* in the PostgreSQL database, thereby allowing us to specify actions at the Prolog level from the database specification.

3.6 Authentication

Given the diverse nature of the information contained within SIIUE, it is crucial to ensure that it is correct. To enable this we constructed an authorization system which:

– Encodes within SIIUE itself the access rights to mutable data. The entities that can change information are themselves represented as data.
– Defines validation paths which are required in order for a change to a relation to become effective. This models the workflow process insofar as certain types of information may be provided by one entity, but require an approval by other entities before becoming effective.

At present, the entire authentication mechanism lies within SIIUE. We plan on integrating this with the Kerberos servers within the University.

3.7 Interface to Other Systems

SIIUE is not planned to be a stand-alone information system. It must allow for the interaction with legacy applications which will act as both producers and consumers of information.

This goal is presently satisfied through ad-hoc solutions, programs which import the definitions from SIIUE and interact with the external programs.

4 Applications

The goal of SIIUE is to provide a framework in which several applications can be developed. The main characteristic of the SIIUE-based applications is that the data they rely on is permanently kept up-to-date.

In this section we briefly describe some of the applications presently in use or in development.

4.1 The Academic Services System

The Academic Services at Universidade de Évora provide access to academic information and records for the whole university, namely for students and faculty. The present system was developed before SIIUE came into being, but has been reworked to integrate into SIIUE: the technology used is essentially compatible (relational database).

At present, the Academic Services application feeds SIIUE with information related to student performance as well as the academic calendar. It relies on SIIUE to provide it with the present constitution of the University, in terms of its institutional structure (eg. departments) and its personnel, especially faculty.

4.2 The ECTS Guides

The European Credit Transfer System (ECTS) establishes rules whereby the academic performance of undergraduate students in any adhering University may be assessed in any other such academic institution. The requirements are that the guides provide information — in at least two languages — to a prospective student on the courses offered for any undergraduate program.

The mechanisms that were provided to meet the ECTS guides requirements are such that the information on specific courses, provided by the faculty members, may be used for other purposes. Such is the case with the on-line course descriptions presented on the University's web site. This issue is further exemplified in section 4.5.

4.3 Institutional Evaluation

Portuguese public Universities are currently undergoing an external assessment process, which requires that much information (esp. academic) be available in a timely and flexible manner. This process focusses on specific graduation programs, which usually share many courses with other programs, and requires that all manner of statistics be gathered from the academic data provided in SIIUE.

The course and staff descriptions used for the evaluation is shared with the ECTS guides, thereby achieving one of the stated goals of SIIUE, ie. non-redundancy at the user level.

4.4 Computing Services User and System Support

One of the most obvious users of SIIUE are the University's computing services, who make use of the information contained within SIIUE to manage accounts for faculty, staff and students. This circumvents the previous requirement imposed upon users that they fill in a (paper) form with their personal information. This information is then used to automate the process of creating user profiles, covering aspects such as e-mail, personal web pages, remote access, etc.

A branch in SIIUE's type hierarchy which is currently being developed deals with all the information pertaining to hardware and networks: the goal is to be able to describe the University's network and produce the various configuration files needed for DNS, routing configurations and SNMP, among others. This work is currently well under way and will be the subject of another article.

It is also in our plans to use SIIUE to describe the University's network and maintain information on installed hardware and software. SIIUE will also be used in the next version of the Computing Services' helpdesk system.

4.5 The University's Web Site

One of the driving goals was to be able to provide up-to-date information on the University's web site, whilst avoiding the duplication of such efforts. The new web site relies heavily upon the information contained within SIIUE to refer to such concepts as Departments, faculty and staff, courses and their contents, graduation programs, research activities, etc.

From the SIIUE perspective, the web site (`http://www.uevora.pt/`) is a consumer-only application. It draws on the information provided by SIIUE to present a highly-connected set of pages, where just about every mention of a term that has a representation within SIIUE is also a hyperlink.

5 Related Work

The University of Porto's Faculty of Engineering has developed a similar information system [7]. However, their approach seems to be restricted to academic and personnel information, lacking the open nature of SIIUE.

At Universidade Nova de Lisboa, a system sharing some aspects with SIIUE is currently being developed: both SIIUE and UNL's system rely on a logic-based description of the type hierarchies involved and generate SQL for the actual application.

6 Future Directions

To promote SIIUE's development and general usefulness, we plan on extending its functionality along the following lines:

– Wider coverage of issues and concepts in an organization such as Universidade de Évora.
  The purpose is to extend the hierarchy described by SIIUE in order to cover concepts such as spaces (building, classrooms, etc.), timetables, research activities, internal document circulation, inventory-related information, network topology information, etc.
  This kind of development is continuously being made, as the system's use reaches further into the organization's structures.

- Further refinements of DL language.
  DL is presently very incomplete insofar as its ability to describe properties of the data model are concerned. We plan on revamping the DL language itself, so as to include abstract types, some form of modular design mechanism (such as contexts [4]).
- Implementation of the on-line DL.
  The present implementation of SIIUE ties the DL and SQL representations off-line only, with the updates performed on the database being propagated to the DL representation in batch and at periodic intervals. It is our goal to ensure that the synchronization is performed on-line, as described in section 3.5.
  To carry this out, we plan on using a lightweight implementation of Prolog such as GNU-Prolog [2], as the basis for our extensions to PostgreSQL. Ironically, Prolog will appear as a *procedural language* from the SQL point of view.
- Mapping of some SQL types as Finite Domains.
  The point is to be able to rely on a CLP system's ability to compute solutions for CSPs. This is highly desirable in an environment such as ours, where many problems may be formulated as CSPs; class schedules are only one such example.
- Higher level interface components on the generated (Java and PHP) code.
  The components presently being generated are fairly low-level; further experience with building applications with SIIUE suggests that more complex components may be constructed.
- Natural language interface.
  One of the most challenging issues when constructing database applications is the ability to automatically generate useful queries from a specification created by a non-technical user.
  It is within our plans to provide support for queries specified in a simplified natural language (we will be targeting Portuguese, obviously), with concepts and vocabulary appropriate for the information contained within SIIUE.
- Visual Programming language to edit the class hierarchy and other aspects of SIIUE.
  With grounds similar to those that motivate a natural language interface, to which we can add the desire to ease the definition process for the concepts which underlie SIIUE (ie. the class hierarchy), SIIUE may prove a fertile ground on which to experiment with visual programming. Work is presently underway to explore this line of research.

7 Conclusions

SIIUE is actively being used at Universidade de Évora; its flexibility has allowed us to address a variety of problems on relatively short notice. Even with a restricted subset of its applications, SIIUE has already gathered enough momentum to prove useful as exhibited by, for instance, the reuse of information between the ECTS guides (see section 4.2) and the Institutional Evaluation (see section 4.3).

When compared to a previous experience we had using the same lower-level tools (PHP3 and PostgreSQL) this approach has proved itself able to cope with the many incremental changes which are bound to happen when dealing with this kind of problem. In particular, changes to the database which can be quite chaotic when directly using the DBMS, tend to be much more controlled if they have to be performed at the DL level. This contrasts clearly with the previous experience we had when designing and implementing a web-based system for the University's Academic Services [8].

It is noteworthy that, so far, only two people have been involved part-time in the design and implementation of all of SIIUE, one of these dealt exclusively with user interface issues. The entire process of design and implementation has nevertheless led to a production release in about a year, even with such limited human resources. This would certainly not have been feasible if not for the expressiveness of the logic programming tools, as witness the related academic services system which required two full-time programmers to develop and maintain during a similar time period.

The system encompasses about 2500 lines of Prolog code, these concern the preprocessing, PHP, SQL and documentation generation as well as the DL source for SIIUE's classes. There are about 100 different classes in the present state of the implementation, these deal with only a fraction of the intended coverage of SIIUE (see the section on future work). The generated SQL schema amounts to approximately 2000 lines of PostgreSQL, including table definitions and various sorts of integrity constraint support functions. The generated PHP support code amounts to about 6000 lines, this is slightly larger than the size of the whole of the hand-written interface code for about 10 different applications, which is also in PHP3.

The issue that, so far, has proved the most troublesome relates to making changes in the schema which apply to information already in the database. The situation seems unavoidable in the incremental development of a system such as this, because the specification is not fully known or even understood by anyone beforehand. The issue has been dealt with manually (painfully so) but we are working towards automating the process: the ability to specify rewrite rules relating different versions of the schema will hopefully highlight our option of using a higher-level description tool such as DL.

As a concluding remark, it must be said that in a traditional university with little CS tradition such as ours, the implementation of a system such as SIIUE has led to a moderate culture-shock situation. Most users of the system (Faculty and Staff, but mostly the former as they constitute a more vocal group) initially failed to grasp the benefits of the electronic fulfillment of their academic duties, and took this requirement as yet another request to provide the information they already had supplied in the recent past. As of this writing and as the second "production" year begins, these misunderstandings are starting to clear up as much information which is common from one year to the next doesn't have to be provided anew.

8 Acknowledgements

The author would like to thank Luis Quintano (Universidade de Évora Computing Services) for his cooperation in the construction of the user interfaces for SIIUE. Universidade de Évora's administration is also acknowledged for the support they provided during the early stages of the project's deployment, without which the work described in this article would have remained a purely academic exercise.

References

1. Grady Booch, Jim Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide*. Addison Wesley, December 1997. ISBN: 0-201-57168-4. See URL http://www.awl.com/cp/uml/uml.html.
2. Daniel Diaz. GNU prolog. URL http://pauillac.inria.fr/d̃iaz/gnu-prolog/, 1999.
3. Sun Microsystems. The Java Foundation Classes. URL http://java.sun.com/products/jfc/tsc/index.html.
4. Luís Monteiro and António Porto. A Language for Contextual Logic Programming. In K.R. Apt, J.W. de Bakker, and J.J.M.M. Rutten, editors, *Logic Programming Languages: Constraints, Functions and Objects*, pages 115–147. MIT Press, 1993.
5. PHP hypertext processor. URL http://www.php.net/.
6. PostgreSQL www site. URL http://www.postgresql.org/.
7. Lígia Maria Ribeiro, Gabriel David, Ana Azevedo, and José Marques dos Santos. Developing an information system at the engineering faculty of porto university. In *Proceedings of the EUNIS'97 – European Cooperation in Higher Education Information Systems*, 1997.
8. Ana Graça Silva and Mário Filipe. O Sistema Informático dos Serviços Académicos da Universidade de Évora. Technical Report, Universidade de Évora, 1998. (in Portuguese).