An extensible web interface for databases and its application to storing biochemical data

Nicos Angelopoulos and Paul Taylor

Department of Biological Sciences, University of Edinburgh, Edinburgh, Scotland, UK
{n.angelopoulos,p.taylor}@ed.ac.uk

Abstract. This paper presents a generic web-based database interface implemented in Prolog. We discuss the advantages of the implementation platform and demonstrate the system's applicability in providing access to integrated biochemical data. Our system exploits two libraries of *SWI-Prolog* to create a schema-transparent interface within a relational setting. As is expected in declarative programming, the interface was written with minimal programming effort due to the high level of the language and its suitability to the task. We highlight two of Prolog's features that are well suited to the task at hand: term representation of structured documents and relational nature of Prolog which facilitates transparent integration of relational databases. Although we developed the system for accessing in-house biochemical and genomic data the interface is generic and provides a number of extensible features. We describe some of these features with references to our research databases. Finally we outline an in-house library that facilitates interaction between Prolog and the *R* statistical package. We describe how it has been employed in the present context to store output from statistical analysis on to the database.
Keywords: user interface, web services, Prolog programming, biochemical data.

1 Introduction

Declarative programming in general and logic programming (LP) in particular when compared to other paradigms, present a much higher level at which programs can be composed. The resulting programs are typically written with less programming effort and are easier to understand.

Interfaces for relational databases are usually the subject of corporate development and absorb substantial programming effort. It is often the case, that the languages these are implemented in are more suitable for the graphical aspects of presentation rather than for capturing the underlying declarative model of the databases. The HTML language, (see for instance [Raggett et al., 1999]), presents a structured approach to user interaction. By design, it is meant for the focus to be on the content and the relations across content rather than on exact graphical coordination.

We constructed a high level web-based interface by exploiting two libraries of the *SWI-Prolog* engine. The first library facilitates the seamless integration

of databases via the *odbc* interface. Prolog is particularly well suited both for manipulating database meta-data (table structure and fields) and for reasoning with the primary data. The second library presents a uniform way for translating between terms and HTML code and an interface for low level communication with the operating system. The former facilitates web-page composition via term manipulation while the latter deals with non-logical aspects of the interaction with the web-services. In tandem the two libraries allow a comfortable high-level style of programming used for rapid-prototyping and the refinement to a stable system that is fully implemented in Prolog.

Logic programming for web services have been advocated in a number of approaches. Notably, in the Pillow library ([Cabeza and Hermenegildo, 2001]) which is available for a number of logic engines. Representing HTML code as term structures is an appealing proposition. Prolog programs can construct these at run-time with correct HTML code generated by library predicates.

The suitability of HTML as an interface platform for databases is not unique to our approach. It is a very popular choice particularly through the PHP scripting language, [Achour et al., 2007]. The benefits of our approach relative to PHP is its relational and high-level nature and the fact that Prolog is a general purpose programming language. On the other hand PHP provides more dedicated features and a large community of practitioners.

The paper is structured as follows. The main components of the system are presented in Section 2 followed by its main database-independent functions in Section 3. Section 4 presents some specific characteristics of using the interface for biochemical data. The concluding remarks are in Section 5.

2 System architecture

This section describes the architecture of our system (*hdb*) and its interaction with the operating system. The overall flow of data is shown in Fig. 1. Semi-circles show interaction software between systems. The round-edged rectangular labelled 'hdb' depicts our system and the three rectangular boxes show its main components.

Requests to service a page arrive at the server machine. The operating system invokes *hdb* via *xinetd* and the *http* library is used to collect the request. This is parsed and in turn *SQL* or queries about meta-data are constructed. Using *odbc* the system interacts with the databases and the results of this interaction are processed into term structures that will form the HTML page to serve. The page is generated by the *http* library. In the specific application we have employed the generic interface, *http* requests are also parsed for references to statistical analysis to be performed by *R* [R Team, 2006]. This is typically the scenario when either a new mass spectrometry experiment is uploaded, or a comparative analysis table entry is requested by the user. The *R.pl* library will handle any such interactions communicating any results from the analysis that need to be saved, to the SQL generator.
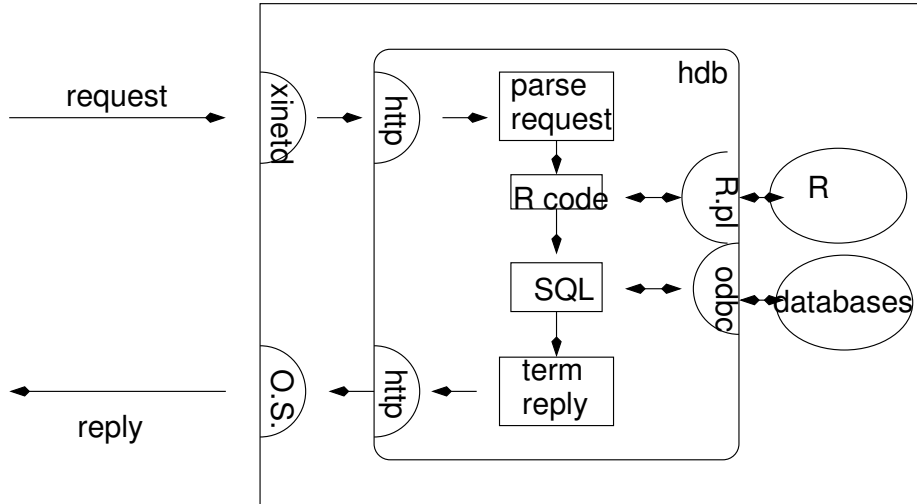
Fig. 1. System architecture and interactions

The system presented here is fully implemented within *SWI-Prolog* (see [Wielemaker, 2010]). The two crucial parts of this Prolog engine that enable this approach are the *http* and *odbc* libraries. As can be seen in Fig. 1 the *http* library plays two complementary roles. On one hand it provides the server machinery for reading in requests, while its other role is to translate special Herbrand terms to HTML documents. The full capabilities of *SWI-Prolog* with regard to web-services is discussed in detail in [Wielemaker et al., 2008].

The *hdb* server is based on the *http* library and can be started in two ways. The simpler method is by presenting a call to the engine, such as:

```
?-http_server(reply,port(8080),timeout(30)]).
```

As long as the engine that runs this query is active, the port 8080 will be serviced by the predicate `reply(+Request)`. The library instantiates `Request` to a Prolog representation of the incoming request. This method is particularly useful during software development as it provides a terminal at which messages about the computation in progress can be delivered to.

Alternatively and more conveniently for non-developmental deployments, the server can be started through an intermediary piece of software such as *inetd* or as is common in our set-up and as illustrated in Fig.1, by *xinetd*. These are daemons in the terminology of operating systems, programs that run continuously listening to the internet ports. A typical *xinetd* entry will reside in `/etc/xinetd.d/hdb` and contain the entries shown in Fig. 2.

Requests to the relevant port (8080 in our example) need to be allowed connection to the server. Typically this is via an entry to the services (`/etc/services'`) file such as 'hdb 8080/tcp' [Postel, 1980].

3

```
service hdb
{
        port            =  8080
        socket_type     =  stream
        protocol        =  tcp
        wait            =  no
        user            =  nicos
        server          =  /srv/www/html/hdb/hdb_xinetd
        log_on_failure  += USERID
        log_on_success  += PID HOST EXIT
}
```

Fig. 2. A typical *hdb xinetd* file

Once the request has been passed to *hdb* and parsed, generation of a reply page will typically instigate some interaction with the serviced databases. The actual interaction is facilitated by the *odbc* library. There are two major types of requests to be served. Ones that require meta-data and ones that manipulate primary data.

The facilities of the *odbc* with regard to meta-data are particularly useful as it means the core interface is completely free of references to any specific database. For instance by using:

```
odbc_current_connection(-Connection,-DSN)

odbc_get_connection(+Connection,database_name(-DB))
```

all open connections can be found and via that all the related databases. Similarly, use of

```
odbc_current_table(+Connection,-Table)

odbc_table_column(+Connection,+Table,-Column)
```

provides access to the database dictionary. With these calls, the database structure and standard operations can be displayed without any hard-wired dependencies. In the following Section we will show in detail how these are used in our system. The *odbc* library depends on operating system connectivity to *ODBC* (open database connectivity) software. An instance of such software is *unixODBC* [Harvey, 2007]. In our experience this has worked well with the Prolog libraries discussed here.

The organisation of the code reflects the two main components of the system. Core functionality predicates are stored in directory `src` while extensibility predicates are in directory `site`. The distributed code is ready to be deployed as soon as a minimal site specific configuration is set-up. This should describe the

4

Fig. 3. Home page

location of the databases to be served and the users of the interface. Note that no information about the database schemata is needed for the core interface to operate correctly.

Our final note in this section is that *SWI-Prolog* in tandem with other open source software provides a powerful platform for serving HTML pages. The high-level of programming, the independence of specific data-sources and the structured nature of Herbrand terms which maps well to correct HTML code ensure that effective interfaces can be built with minimal programming effort.

3 Core functionality

In this Section we describe the main functions of the interface. We present and discuss features that are generic and can be employed as-is in any installation of the system. We distinguish six types of pages in this category: (a) authentication, (b) overview of databases (c) view of database, (d) single table, (e) single view and (f) profile data. In what follows we detail these six categories.

The first point of contact a user will have with the interface is the authentication page, where they are asked for a username and password. There are two types of authentication services provided. The site administrator must choose one of them at installation. The first, uses the Prolog internal mechanism that keeps a thread alive for a specified amount of time. Usually the amount of time is set to a few minutes. Within that period the connection is authenticated and the user can interact with the system. Each interaction resets the thread to the start of the interval. If there is inactivity for a period longer than the interval the process dies and the user will need to log-in again.

The second type of authentication services is IP address based. In this mode of operation, the credentials of a user are associated to the IP address from which they last log-in. The interval during which this activation is valid is usually set to a few hours, typically the duration of a working day. This provides a less secure model but which is much easier to use in practice. In both authentication scenarios, *hdb* is by necessity of a more or equal restrictiveness to that of the underlying databases with regard to database operations. Each *hdb* user is assigned to a db user in a modular way and the exact mapping is a matter of site administration, although typically both credentials will be identical. It is worth noting that authentication has been designed with ease of use in mind rather than maximal security from determined attacks.

Once users have tackled authentication and gained access to the system they are presented with the home *hdb* page which shows an overview of the resources they can access. An example is shown in Fig. 3. At the top right, there are persistent navigation options that allow logging-out, access to session profile and a link to the home page. The main body of the page gives access to the databases and views accessible to the user.

Each database page presents the tables which are contained in the database source along with a subset of standard operations that can be performed on each. A partial example is shown below:

Experiment	[input]	[update]	[delete]	[query]	[all]
ExternalDataSource	[input]	[update]	[delete]	[query]	[all]
Input	[query]	[all]			
Mix	[input]	[update]	[delete]	[query]	[all]
MixIngredient	[input]	[update]	[delete]	[query]	[all]
Plate	[input]	[update]	[delete]	[query]	[all]

Note that in our example table 'Input' is read-only and it is thus only associated with read operations. This table is dedicated to recording the changes that occur to the databases from within the *hdb* interface. Our system supports explicit storage of such information. Navigating via the table name will present a single table page while the link on each operation links to the operation page for the specified table.

A page detailing a single table shows the fields of the table along with their types and provides access to operations on the table. These are identical to those that appear along side the table name in the database page. They will usually be a subset of: 'input', 'update', 'delete', 'query' and 'all'. Typically it will be the maximal subset, and it is also possible to hook table or database specific operations. An extract from a table page is as follows:

Scibs DBs 0.2

[home] [profile] [logout]

Table: Compound

Table columns:

Name	Defn. Type	Null	Key	Def	Extra	
CompID	bigint(20) uns.	NO	PRI		autoinc	autoinc
CompName	tinytext	YES				
CompMr	float uns.	NO				
pKa	float	YES				
EduID	bigint(20) uns.	YES				
CompNote	text	YES				

input

Compound [input] [update] [delete] [query] [all]

hdb_v0:0:2

Fri Aug 24 16:13:04

Fig. 4. Input data to a table.

scibsdb.Compound has 210 rows.
Table columns:

Name	Defn. Type	Null	Key	Def	Extra
CompID	bigint(20) uns.	NO	PRI		autoinc
CompName	tinytext	YES			
CompMr	float uns.	NO			
pKa	float	YES			
EduID	bigint(20) uns.	YES			
CompNote	text	YES			

The first line gives the database and table name along with its population size. The headings of the columns appearing on the third line of the example above are, in left-to-right order: the field name, the type of the field, a Boolean value signifying whether or not a null value is allowed, a value reflecting whether field is a key for the table, the default value for the field and finally, any extra information about the field. In our example the first field, 'CompID', is an auto-incrementing integer field. Also note that 'uns' is an abbreviation for unsigned.

Most operations follow intuitively after their name. The 'all' operation is the maximal query that allows the user to view all data in the table. (Standard site-wide defaults of presentation style apply.) One of the more useful operations is inserting data into tables. An example is shown in Fig. 4. The interface auto-matically handles auto increment fields (integer fields which are routinely used as unique identifiers) and enumeration types that are displayed as dropped down menus.

Returning to describing the 'home' page, we have another type of object, namely, views. Views are convenient higher level objects which allow operations that manipulate more than one table, or more than one row of a single table within a single step. In Fig. 3 a number of specialised views are offered to the user. For instance 'sdf_3d' displays the 3D representation of an SDF entry. Each SDF is a textual representation of a chemical substance ([Dalby et al., 1992]). The view

Fig. 5. Input of multiple experiments.

'observations' has one operation which allows results from many experiments taken at a single time-point to be inputted together. As most of the views and operations are specific to a given site and makes extensive use of the extensibility features of our system they will be discussed in further detail in the next section. An example of an operation to such a view is given in Fig. 5, the details of which will also be discussed in what follows. It suffices here to say that *hdb* provides convenient hooks for new views and associated operations to be defined within Prolog.

Finally, the user has access to information profiling his session. Information regarding the user names, IP address, session ID and the server associated with this session is presented, as illustrated in the following example:

User Profile

Logged-in on hdb server: Scibs DBs 0.2
With user name: nicos
Database user name: nicos
Login time: at(2007, 8, 24, 14, 22, 40, 10)
Peer: 129.215.137.168

Pages are served by : hdb 0:0:2
Server: scibsfs.bch.ed.ac.uk:8080
Session: 5807-da08-fbaa-fe69

Warnings and errors about *hdb*'s operation are reported by the system at the top of the first generated page after the error was caught. Messages are asserted as `diagnostic/1` terms via the `http_session_assert/1` primitive. The example

8

```
% hdb_hook_column_input_def_value(+DB, +Table, +Column, -Def ).
hdb_hook_column_input_def_value(ni_lhh, _, Column, Date) :-
     atom_concat( _, 'Date', Column),
     get_time(Time),
     convert_time(Time, Yr, Mo, Dy, _Hr, _Mn, _Se, _Ml),
     number_codes(Yr, YrCs),
     number_codes(Mo, MoCs),
     number_codes(Dy, DyCs),
     flatten([YrCs,"-",MoCs,"-",DyCs], DateCs),
     atom_codes(Date, DateCs).
```

**Fig. 6.** Date default values in database 'ni_lhh'.

message

$$\texttt{unable\_to\_connect\_to\_db\_source(nilhhloc-ni\_lhh)}$$

informs of the unavailability of database `ni_lhh` right after log-in. The message appears near the top of the page in orange colour as to draw the user's attention and is non persistent.

4 Storing biochemical data

The interface can be tailored to a site's specific needs through a number of hooks and handlers. Both of these can alter the HTML generated by the system. Hooks are optional parts that when present alter a specific part of the interaction. For instance, a particular field's output can be linked automatically to a live HTML link, or certain fields for a table are auto-filled by some scripts. Hooks are appropriate in extending the system in a way that allows database table evolution. The addition of extra tables or columns (fields) will usually require no changes to hooks of existing columns. Handlers on the other hand, are predicates that deliver more substantial extensions to the interface. For instance a handler may produce the HTML code from a non-standard operation on a view (such as the 'disp' operation on view 'sdf_3d' detailed below).

For example, hooks are used in views to declare the participating columns (`db_view_has_column(+View,+DB,+Table,+Columns,+JnKeys)`) and the operations defined on a particular view (`db_view_has_ops(+View,+Op)`). For tables, hooks are very useful in defining input, output settings that can take into account local administration issues. For instance, `hdb_hook_column_input_textarea(+DB,+Table,+Clmn,-Rows,-Cols)` can be used to overwrite the default size for the input text box at an insert operations. For a more concrete example, consider the code in Fig. 6. Predicate `hdb_hook_column_input_def_value(+DB,+Table,+Column,-Def)` when defined for a specific column, table and database combination, it dictates a default input value for the said column. The shown code defines the current date as the default value for each column in

9
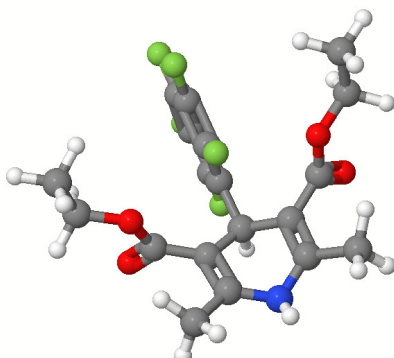
Fig. 7. Jmol invocation. NemadipineA, (hdb) Eduliss ID: 108525.

database 'ni_lhh' that has a name containing 'Date' as a suffix. In our example this is true irrespective of the table's name.

In the remainder of this Section we will present some database-specific aspects of the system and our local installation which stores lab results and biochemical data. Briefly described the main requirements are: storage of experimental data from (a) C. elegans growth, (b)micro-array and spectrometry data for samples resulting from the C. elegans experiments, and (c) information about associated molecules.

Although a limited number of target ligands are used in our experiments we are able to include into our system a much broader selection of chemical substances. In total 1.7 million are available and they are drawn from the EDULISS database [Hinton, 2005] (version 1.0). A structured schema has been devised for storing the atomic information that reside in flat format SDF files. Individual molecules can then be inspected by retrieving all relevant entries writing those out in the usual flat file format and call the Jmol java applet, [Jmol, 2007], to display the three or two dimensional representation of the molecule. An example of a displayed ligand (small molecule) is in Fig. 7. The display operation is labelled 'disp' and is available for views 'sdf_2d' and 'sdf_3d'. The example shown is Nemadipine-A and was shown to induce a marked phenotype in C. elegans ([Kwok et al., 2006]).

It is often the case that complications of database interfaces arise from the disparity between the best way to hold the data in the database and the users' way of organising the data in their minds. Inputting data is one crucial part in which a reconciliation is vital. In our system views can be used to insert data in ways that are more intuitive and faster for the user. As an example consider Fig. 5. The user is able to input a number of observations corresponding to 'well' experiments that were started at the same time. This is an intuitive way of viewing the set of experiments as all wells come from a single plate and have
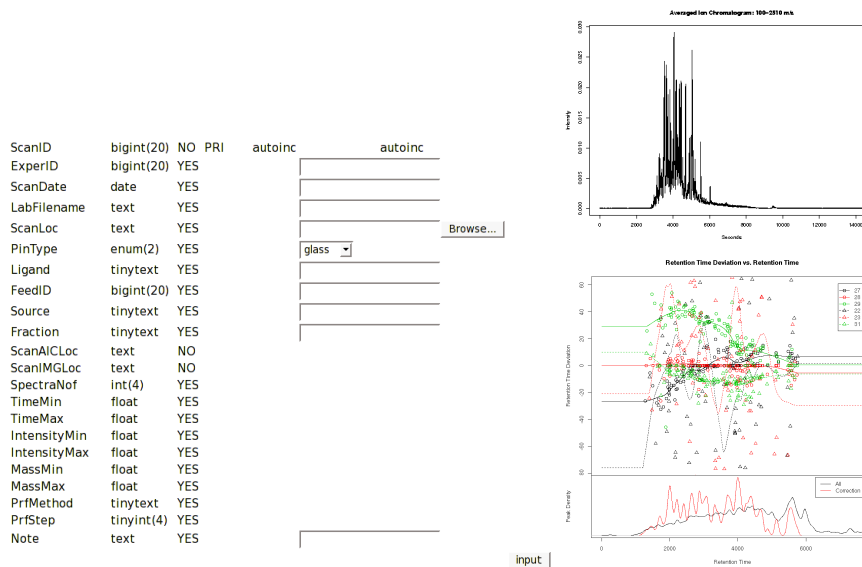
ScanID   bigint(20)   NO   PRI   autoinc   autoinc
ExperID   bigint(20)   YES
ScanDate   date   YES
LabFilename   text   YES
ScanLoc   text   YES   Browse...
PinType   enum(2)   YES   glass ▾
Ligand   tinytext   YES
FeedID   bigint(20)   YES
Source   tinytext   YES
Fraction   tinytext   YES
ScanAlCLoc   text   NO
ScanIMGLoc   text   NO
SpectraNof   int(4)   YES
TimeMin   float   YES
TimeMax   float   YES
IntensityMin float   YES
IntensityMax float   YES
MassMin   float   YES
MassMax   float   YES
PrfMethod   tinytext   YES
PrfStep   tinyint(4)   YES
Note   text   YES

input

Fig. 8. Left: automatically filled fields. Right: generated averaged ion count (top) and retention time correction (bottom).

synchronised start times. On submitting the data a number of separate table entries will be stored in the database. The majority of the entries in Fig.5 are in the form of selection menus as they are of enumeration type further easing the input process.

A major consideration in experimental settings is the management of raw datafiles as well as the tracking of large files as they go through various stages of analysis. Our approach is to use the uploading capabilities of the *http* library and store the files in canonical locations with a file link stored in the actual database. The library has been performing robustly and has coped with files of substantial sizes.

In addition to storing primary data we have used auto filling capabilities in the system to automatically generate store derived information on the data. In particular output from mass-spectrometry experiments that are exported in NetCDF format are uploaded to the database. The *xmcs* [Smith et al., 2006] part of the *Bioconductor* [Bioconductor, 2009] package for the R [R Team, 2006] statistical system is started through a Prolog interface. We have built a simple *Prolog* to R interface that runs the latter as a slave with the command:

```
R --slave --no-environment
```

The input, output and error streams of this process are handled by Prolog as to enable interaction with the R shell. The in-house library facilitates the translation of term structures to atoms that are written on to the R process' input.

Also it reads in and translates output that appears in the R process' output and error streams. It thus allows Prolog to access the vast wealth of functions and packages available in R. The Prolog-to-R interface is also made available as an independent module ([Angelopoulos, 2010]) and is included into *SWI* main source distribution as a contributed package.

Fig. 8 shows an example of automatically filled table entries. In entering data to table 'SpecScan' the user can only input values for 7 out of the 18 fields. The remaining are filled with: (a) visual summaries of the data- a postscript output of the averaged ion count for each scan (ScanAICLoc) and a heatmap image (ScanIMGLoc), (b) information about the sample that exist in the bundled input (fields SpectraNof to MassMax) and basic parameters for some of the visual summary fields- here PrfMethod and PrfStep for the heatmap generation. All derived information are produced by invoking the *xcms* R package on the data uploaded by the mass-spectrometrist (field ScanLoc).

Our approach has the benefit that output from specialised software is stored in accessible forms that can be viewed by all partners in the project. It is also the case that it improves quality control tasks and that by automatically filling the form there is less mundane typing and data-entry and thus less chances of an error to occur. Additionally in the scenario we have employed the interface, the statistical analysis with *xcms* takes a substantial amount of time to run. By storing the results of the analysis in a central repository with all associated information managed automatically, we ensured that subsequent inspections of the analysis are handled promptly and correctly.

The top left of Fig. 8 shows the main part of an input operation with the values of the fields that can be changed filled. At the top right is the generated averaged ion count (AIC) which plots average ion count against time and provides an overall picture of activity within the spectra. The bottom right of Fig. 8 shows the retention time correction as generated by *xcms* when creating a comparison among spectra from two conditions. Retention time correction is performed as to align multiple mass-spectra from two distinct conditions so that their peaks can be compared in a meaningful way. Visual inspection of the correction is a good indicator for the quality of the alignment and thus a crucial information to include in the stored derived information information.

5 Conclusions

We presented a system that implements a high level interface for data sources by exploiting two *SWI-Prolog* libraries. The two main advantages of our implementation platform are the relational nature of Prolog and the compositional nature of Herbard terms. The former allows seamless integration of relational databases and the latter facilitates construction of structured HTML code.

To our experience the main benefits of *hdb* are (a) that it implements a well separated organisation of what is generic and what must be tailored for a particular installation, and (b) that the generic part of the system is schema-

driven in the sense that it is automatically constructed from the underlying databases with no hard-wired dependencies.

Our work has demonstrated the usefulness of a relational programming language for providing interfaces to databases via served HTML pages. The generic part of the interface can be deployed directly to any existing database. The installation can then slowly be ramified and tailored to the particular site via a number of extensibility features the system provides.

We have identified four directions in which we will direct future work. First is the ability to have communication between different pages via collections of database items. These can be through tick boxes at query result pages. Also to provide generic mechanisms for background processes filling of derived information. This will allow the interface to work as a front-end and hiding a lot of dedicated background processing. In our databases, for instance, we can allow mass-spectrometry staff to create views that explore differentially expressed mass over charge peaks from a set of controls against a set of screens. The third direction in which to develop the system is in providing simpler query mechanisms such as ability to choose a list of fields and provide values or regular expressions for those. Finally, to modularising the database access and allowing alternative methods of interaction such as via the BerkleyDB interface (package 'db' in *SWI-Prolog*).

The source for the system described in this paper can be downloaded from `http://scibsfs.bch.ed.ac.uk/~nicos/sware/hdb`. It has been developed and tested on Linux systems. The in-house library that handles communication with *R*, *r_session,* is included as a contributed package in current releases of *SWI-Prolog* (*library('R')*).

6 Acknowledgements

References

[Achour et al., 2007] Achour, M., Betz, F., Dovgal, A., Lopes, N., Richter, G., Seguy, D., and Vrana, J. (2007). *PHP 5 Manual*. http://www.php.net/.

[Angelopoulos, 2010] Angelopoulos, N. (2010). r_session (0:0:3). an inteface to R for Prolog. `http://scibsfs.bch.ed.ac.uk/~nicos/sware/r_session`.

[Bioconductor, 2009] Bioconductor (2009). Bioconductor 2.5: open source and open development software project for the analysis and comprehension of genomic data. `http://www.bioconductor.org/`.

[Cabeza and Hermenegildo, 2001] Cabeza, D. and Hermenegildo, M. (2001). Distributed www programming using (ciao-)prolog and the pillow library. *Theory and Practice of Logic Programming*, 1(3):251–282.

[Dalby et al., 1992] Dalby, A., Nourse, J. G., Hounshell, W. D., Gushurst, A. K. I., Grier, D. L., Leland, B., and Laufer, J. (1992). Description of several chemical structure file formats used by computer programs developed at molecular design limited. *Journal of Chemical Information and Computer Sciences*, 32:244–255.

[Harvey, 2007] Harvey, P. (2007). *unixODBC manual.* `http://www.unixodbc.org/`.

[Hinton, 2005] Hinton, A. C. (2005). *Database Mining: EDULISS a Descriptor Based Approach.* PhD thesis, Structural Biochemistry Group, Institute of Structural and Molecular Biology, School of Biological Sciences, University of Edinburgh.

[Jmol, 2007] Jmol (2007). Jmol: an open-source Java viewer for chemical structures in 3D. `http://www.jmol.org/`.

[Kwok et al., 2006] Kwok, T. C., Ricker, N., Fraser, R., n W. Chan, A., Burns, A., Stanley, E. F., McCourt, P., tler, S. R. C., and Roy, P. J. (2006). A small-molecule screen in c. elegans yields a new ca lcium channel antagonist. *Nature*, 441(4).

[Postel, 1980] Postel, J. (1980). User datagram protocol. Technical Report STD 6, RFC 768, USC/Information Sciences Institute.

[R Team, 2006] R Team (2006). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria.

[Raggett et al., 1999] Raggett, D., Hors, A. L., and Jacobs, I., editors (1999). *HTML 4.01 Specification.* W3C. `http://www.w3.org/TR/html401/`.

[Smith et al., 2006] Smith, C., Want, E., Tong, G., Abagyan, R., and Siuzdak, G. (2006). Xcms: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching, and identification. *Analytical Chemistry.*

[Wielemaker, 2010] Wielemaker, J. (2010). *SWI-Prolog Manual, 5.9.10.* `http://www.swi-prolog.org`.

[Wielemaker et al., 2008] Wielemaker, J., Huang, Z., and van der Meij, L. (2008). Swi-prolog and the web. *Theory and Practice of Logic Programming*, 8(3):363–392.