Chapter 2

FunGIMS Design and Implementation

2.1. Overview

The FunGIMS (Functional Genomics Information Management System) is a web-based system designed to integrate most of the major data types that a researcher might encounter in a modern functional genomics experiment. These data types include sequence data, protein structure data, microarray data, small molecule data and literature data. In addition, it also provides online access to some of the more commonly used tools in each of the data type subsections. This allows the user access to data and analysis tools in one, centralized location as well as providing storage for the data generated by the analysis tools in FunGIMS.

The following sections will discuss the technologies used in FunGIMS as well as the design process and the data model used.

2.2. FunGIMS Design and Technologies

During the design phase of FunGIMS, every effort was made to find the most appropriate technologies for each section of the project. Every section involved exhaustive investigations and testing of the options currently provided by software manufacturers. Important decisions such as a specific programming language, were only made after extensive research into the support provided and the ability to allow the programmer to do a specific job.

2.2.1. Technologies

For the success of a large project such as FunGIMS, various technologies are needed to work in unison to produce the final outcome. Each of these technologies will be discussed shortly in the following few sections. For the programming languages, Java and Python were investigated extensively as well as the availability of software packages which allow for interaction with databases. Different language-dependant web frameworks were also investigated. These included JBoss, TurboGears, Java Struts and custom Python scripts on top of a CherryPy server or Apache web server. The ability of a language to interact with databases and facilitate easy data persistence led to investigations into Java Beans, Hibernate, SQLObject and SQLAlchemy. Architectures such as the Model-View-Controller and Server-Client designs were investigated to find the most suitable option for delivering data and interactivity to users. In the software world it is important to choose your technologies wisely due to the rapid rate of new developments and the decline of once-popular software. The following sections will discuss the choices made for each of the technology aspects of the project.

2.2.1.1. Python

The programming language chosen for this project was Python (http://python.org). Python has been developed by Guido van Rossum since 1991 and is a mature and stable development language. This maturity has led to it being used by the biggest search engine company at the moment, Google (http://www.google.com), on a wide range of services. The widespread use of Python and the ease with which it is learned has resulted in an extremely wide code base that caters for a vast amount of functionalities. In the last few years Python was used in developing games such as Civilization IV (Firaxis Games, http://www.2kgames.com/civ4/home.htm), high performance scientific computing packages (NumPy, http://numpy.scipy.org; SciPy, http://www.scipy.org), web development platforms (TurboGears, http://www.turbogears.org; Pylons, http://pylonshq.com), movie animations (Blender3D, http://www.blender.org) and being supported in commercial scientific packages such as Discovery Studio II (Accelrys Inc.). Python was chosen due to its stability, ease-of-use and multitude of packages.

Python is also widely used in Bioinformatics due to its ease of use. Examples over and above scripting include: PySCeS (Olivier *et al.*, 2005) that is used very successfully in modelling the kinetics and substrate flow through enzymatic pathways (Uys *et al.*, 2006), PyMol (http://pymol.sourceforge.net) that is a very successful open source python-based 3D protein structure viewer, and PyQuante (http://pyquante.sourceforge.net/) when doing quantum mechanics.

2.2.1.2. Web Development Framework

For FunGIMS it was decided to use the TurboGears web development platform. TurboGears is mature, well developed and written in Python and allows for development of projects using all the possibilities provided by the Python language. Development in TurboGears takes some time to master but should a person have previous Python programming skills, the process is far quicker. TurboGears is based on the Model-View-Controller architecture (see section 2.2.2) and uses various other packages to perform the different functions. The use of Python and the MVC architecture in TurboGears made it the perfect choice for FunGIMS, which uses the same technologies and thus allows for easy integration. Figure 2.1 shows a diagrammatic layout of the functioning of TurboGears.

2.2.1.3. Object-Relational Mapper

Often a time consuming step in programming is constructing code to represent the data queried from a database. To overcome this problem, Object-Relational Mapping (ORM) was developed. This is a method whereby a query to a relational database can be represented in an object-orientated way in the code. The programmer defines all the tables in the database using code and also defines classes for working with the tables. The ORM then uses this information to transparently connect to the database, and provide the programmer with access to the data using the predefined classes. The ORM also provides some methods, native to the database, as normal methods owned by the classes. Thus the programmer does not have to learn the syntax needed to manage the database natively, only the concepts need to be known. These methods allow the programmer to continue programming in the same style, without the need to write his own mapper be-

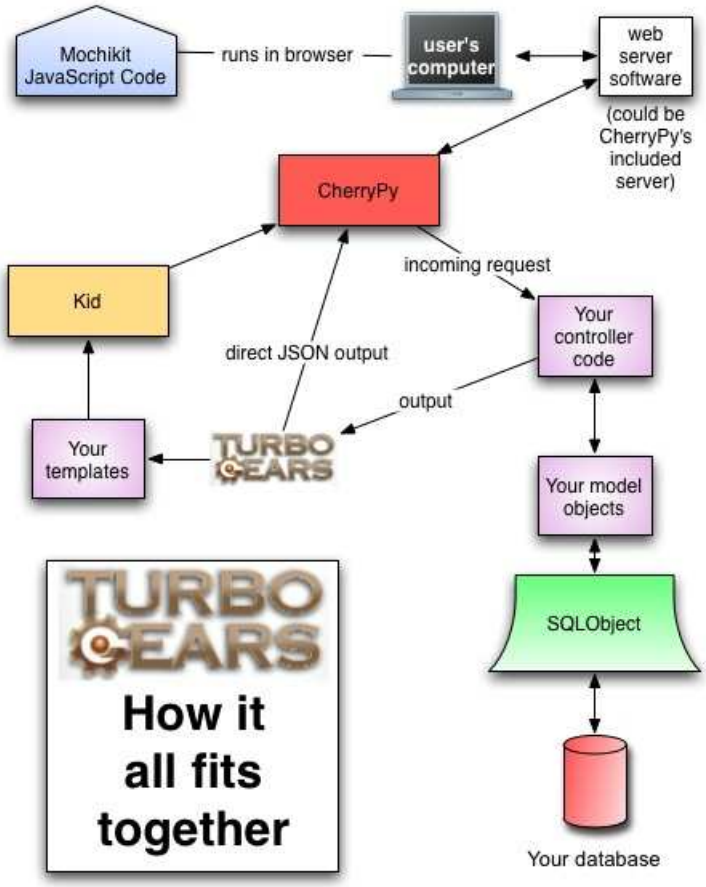

Figure 2.1: A schematic representation of how the different parts work together in TurboGears (http://docs.turbogears.org/1.0/GettingStarted/BigPicture). The user makes a request for data in the browser. This request gets directed by the controller to the model. The ORM then connects to the database, retrieves the data and returns it to the controller. The controller then provides the data to the appropriate template, which is served up as HTML code to the user's browser.

tween the database and the program. For FunGIMS it was decided to use SQLAlchemy (http://www.sqlalchemy.org). SQLAlchemy is supported in TurboGears and uses the `model.py` file to define the database, link the tables in the database to code classes and implement data class specific methods. SQLAlchemy was chosen in preference to SQLObject as it provided more advanced functions such as polymorphic joins and class creation via introspection of the database. At the time of writing, SQLAlchemy was also slated to become the default ORM for the TurboGears project. It was decided to use MySQL (http://mysql.org) as the relational database for FunGIMS. This was chosen the preferred choice rather than PostgreSQL as SQLAlchemy provided slightly better support

for MySQL than for PostgreSQL when the project was started. Most of the developers also had more exposure to MySQL than PostgreSQL. MySQL provides a way to store vast amounts of data, while providing extremely fast search access to the data. All the data are stored in rows in user-defined tables, and a user can search over all fields in the tables. This provides a very powerful way of storing and querying data.

2.2.1.4. Version Control

In a project of this scope, version control is essential. Version control provides a way for the system to be backed up in increments as each part of the system changes. A developer can check out a certain part of code, work on it and then check it back into the system. The system then checks whether there was any conflict in the code, and store the changes made to the code. It also tracks the changes each developer makes as well as any changes to files. Furthermore, it prevents changes made by the different developers on the same piece of code to be checked in prior to validation thereof. An essential feature is the ability to rollback changes made to the system. It was decided to use Subversion (http://subversion.tigris.org) for this project rather than Concurrent Version System (CVS).

2.2.1.5. Templating Language

Web browsers display pages written in HyperText Markup Language (HTML). HTML uses a static code to represent items on a web page. To overcome the static element of HTML, programmers developed templating languages. These languages allow a programmer to generate static HTML content based on decisions made by the algorithm or program or even based on user input. The Kid templating system (http://www.kid.org) was used for FunGIMS. Kid is a templating system that is based on eXtensible Markup Language (XML), of which HTML is a derivative, and allows for the incorporation of Python code in the template. KID will take the XML template and the data provided by the controller, combine it and render it into HTML that is then sent to the web server. The user will then see the page as normal HTML in his browser.

2.2.2. Development and Design

The design of a large system such as FunGIMS is a complex task and requires careful development and planning to prevent a cluttered and complex code base. This is especially important when there are multiple programmers working on a project and coordination between them is vital. The first step in planning such a project is to identify the potential users and analyze their requirements. These requirements must then be implemented in a logical way to benefit the user. The programming task must also be divided amongst the programmers to speed up development.

As a first step, the use of object-orientated programming was implemented. This results in code blocks that can be reused throughout the project and facilitates faster development. A Model-View-Controller architecture was also followed (Fig. 2.2) for the software design of FunGIMS. This architecture separates a project into three different sections on the basis of the function of each section:

- Model - this contains all the code necessary for the storage of results and managing the database back end as well as handling queries to the database.
- View - this section contains all the code used in displaying results/output from the system. It contains mostly templates and usually contains very little logic code.
- Controller - this is the section in which all the functionality and the majority of the code resides. All the decision making processes in the system are stored here, and it controls input and output to the model and view. It "controls" the entire system and directs traffic and requests to the appropriate subcontrollers.

Following the MVC architecture, the project was divided into three sections namely `model.py`, `controller.py` and a folder for all the templates entitled `templates`. These are each discussed in more detail in sections 2.2.2.1, 2.2.2.2 and 2.2.2.3. In Figure 2.3 the overall design and implementation of the MVC architecture in FunGIMS is shown. This high level overview provides a clear depiction of how each part of FunGIMS fits together.

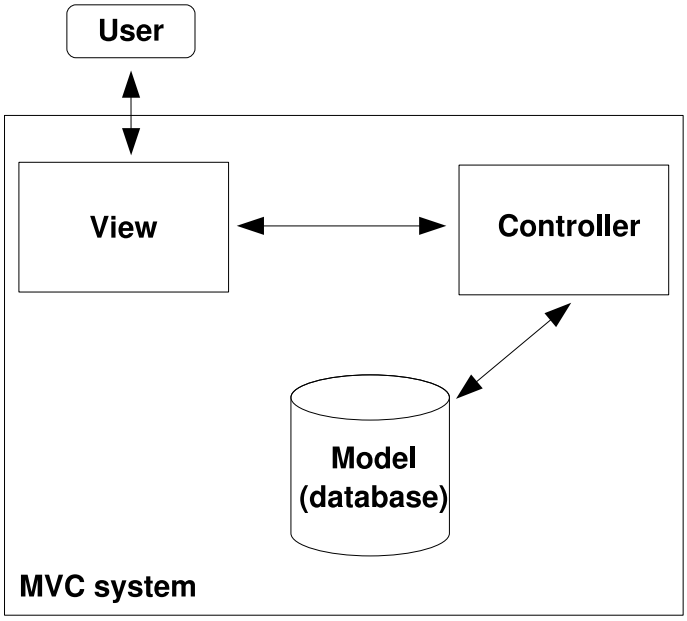

Figure 2.2: The Model-View-Controller (MVC) architecture. The Model contains the data model needed by the ORM to interact with the database. The View contain all the templates needed to display the data and the Controller controls and handles all communication between the Model and the View. The controller also calls any external programs that are needed.

During the development process, the spiral development methodology was followed. This methodology is based upon small improvements and step-wise additions of features, followed by rapid deployment and testing of the new features. This cycle is repeated as each new feature or functionality is added. The advantage of this methodology is that errors in the code and feedback from the users can be corrected and implemented quickly, which results in less effort compared to corrected errors in a project where the release and testing cycle is longer. Most of the modules were developed in conjunction with user input. Thus at each stage in the development, the user was consulted. The user was asked which functionalities he wanted, where after the programmer would implement it and the user would test it and give feedback.

During the design of FunGIMS, the usability and users of the system were always kept in mind. This forced the coding process, and the code itself, to be far more efficient and intelligent in the manner in which the different applications and functionalities were implemented. A good example of this is the System ID (sid) that is assigned to every entry of a data type. The sid should identify the specific record in such a way as to

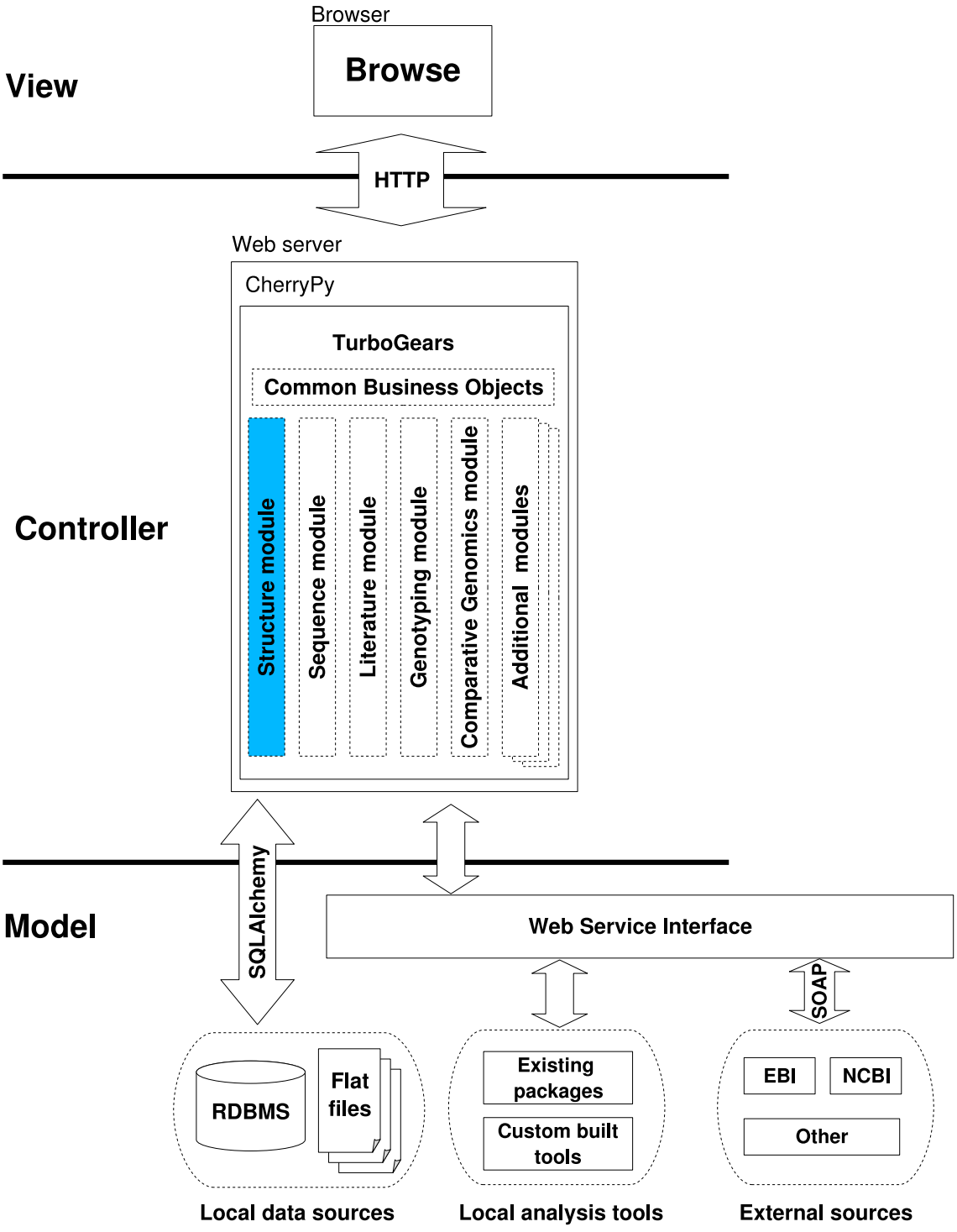

Figure 2.3: The overall design of the FunGIMS system. The design follows the Model-View-Controller architecture and uses TurboGears as the web development environment. Various other modules such SQLAlchemy provide interfaces and methods to access data and call external programs. The View provides the interface the user sees when using the system. The Controller controls and directs all requests within the system and the Model stores all the data.

facilitate easy use during coding, as well as for easy understanding thereof by the user. With FunGIMS the number of records of different data types was huge. To assist users as well as facilitate easier coding, it was decided to use a common sid format. The format, <*data type:id*>, consists of a data type identifier, followed by a :, followed by a unique number for user-generated data or the id assigned by the specific public database e.g. PDB file 1eye would have the sid: pdb:1eye. This identifies the record as a protein coordinate file and uses the more well known public database id as well. The PDB is a good example of the efficient use of a system-wide, unique id. The unique number is generated by taking the system time, in seconds since 1 January 1970, and multiplying it by a factor of ten million to get an integer number.

At the time of writing, FunGIMS catered for the following data type identifiers:

- seq - user generated/uploaded sequence
- gi - sequence from GenBank public database
- sp - sequence from SwissProt public database
- pri - user generated primer sequence
- pdb - protein structure file from the PDB
- pmid - article from the PubMed public database
- file - user uploaded generic file
- chebi - small molecule from the ChEBI database
- note - user generated note
- blast - BLAST results file
- go - Gene Ontology term
- taxon - NCBI taxon term
- trace - DNA sequence chromatogram files

These data type identifiers makes it easy for the user to see which entry they are currently working on or which entry's results they are looking at. To make the development process faster, each programmer was given responsibility for a module on FunGIMS, while core modules were developed together as they were needed.

Coding was not the main area where ease of use was of primary importance. Ease of use is the most important in the user interface. Throughout FunGIMS the interfaces were designed to be clean, intuitive and easy to use. This implies that pages do not show unnecessary information to the user. Future releases may have the option to display extra information contained in the relevant files. Each page is designed to show only the information the user needs at that moment. In the case of analysis tools, the user is asked for only the necessary information before the analysis is run.

2.2.2.1. The View

The views in FunGIMS are responsible for interacting with the user and presenting data to him. Although the views only present data, in some instances decisions on display items can only be made once the data is rendered or to alleviate more extensive coding of templates. Each view is written in the Kid templating language. Each module in FunGIMS has its own set of views and a shared subset deals with general, administrative displays such as headers, new user registration and shared items. The view files are stored in a separate directory (`templates`) and use the .kid extension. The views are compiled to Python code as needed using just-in-time (JIT) compilation.

The view also makes use of JavaScript for some visual effects and for managing the addition and deletion of notes through JSON, an AJAX library (Asynchronous JavaScript and XML) used in TurboGears to connect Python functions and JavaScript. The view also allows the inclusion of applets such as Jmol, which is used in the Structural module. These applets allow for extra functionality in the browser.

2.2.2.2. The Controller

The controller is that part of FunGIMS that regulates all the decisions regarding flow control. The controller decides what data must be retrieved, what data must be sent to the view and which commands to execute with regard to the given variables. In essence, the controller controls everything in the application. All code that make a decision resides in the controllers. In FunGIMS the responsibility of the controller has been split to facilitate collaborative coding as well as to decrease the amount of code residing in one main con-

Table 2.1: The technical specifications of FunGIMS.

Feature	
Programming Language	Python 2.4
Development Framework	TurboGears 1.0.2
Code Revision Control	Subversion 1.2.3
HTML Templating	Kid 0.9.6
Object Relational Mapping	SQLAlchemy 1.3.9
Documentation	Epydoc 3.0beta1
Back end Database	MySQL 5.0

troller. The main controller (`controller.py`) in FunGIMS decides which sub-controller (located either in the `view_controllers` or `search_controllers` folders) receives the data and which sub-controller is responsible for executing the user's commands.

In FunGIMS the following tasks are under the direct responsibility/control of the **main** controller:

- Deciding which view to present to the user
- Managing the search functionality
- Managing user access (logging in/out) and security
- Making decisions on which analysis interface to send data to
- Upload/download of files
- Generic saving of results produced by analysis methods
- Web services

The technical specifications of FunGIMS are given in Table 2.1. The choice of language (2.2.1.1), development platform (2.2.1.2) and other decisions have been discussed in the relevant sections.

2.2.2.3. The Model

The model forms the basis of all the interactions between the controller and the database in the MVC architecture. All the table definitions, table-class mappings and class-specific methods are defined in the `model.py` file. This file is used by the ORM to interact with the database and return the relevant data to the controller. The details of the data

model will be discussed in section 2.4.1. There are a few main model-related methods that are used across FunGIMS. These include retrieving data for a specific entry while considering security and access restrictions on the entry, deleting privately owned data and generating new, unique identifiers for data inserted into the system.

2.3. FunGIMS Core Functionalities

FunGIMS contains a few core functionalities that are used across the board in all the different modules. These include managing users and groups, new registrations and searching of data.

2.3.1. User and Group Management

Common practice in laboratories is to divide people into work-related groups. This concept was also used in FunGIMS to manage access to data. When starting a TurboGears project, it provides you with default identity handlers. These are divided into users and groups. Each user can belong to one or multiple groups. For FunGIMS this definition was extended so that groups can also belong to other groups e.g. the different groups in an academic department. An example would be a supervisor who wants to share data with her students as well as between the students, but also wants her own private group. Under the FunGIMS identity scheme this would mean that the supervisor belongs to two groups, her own private group and the student group. This would allow the students to share data but also allow the supervisor to have private data. It is basically a concept of group of groups. Although this complicates the identity management, the advantages thereof are far more than the extra effort required to program it.

In FunGIMS each data entry belongs to either a specific user or group or, in the case of publicly available data, to the "world" group. The "world" group is accessible to everyone and all users can view and use entries belonging to this group. When data belongs to a certain group, all the users who are members of that group may access, view and use the data. This hierarchical implementation of access restrictions allows for the separation of visible data to each group. A user may also decide to browse and analyse

data anonymously. This will allow him to see all public data and do analysis, but not save any results, or add notes to any entries.

To manage users, a registration section was included. This enables the user to add new users, add users to groups and to create groups. Some restrictions are also implemented, which gives only certain users the right to add or delete users.

2.3.2. Result Management

When users generate results in FunGIMS, they are presented with the option of either storing the results in the FunGIMS database or viewing them without saving. This functionality allows users to use the FunGIMS database as a data repository. User-generated results are stored as uploaded files in the database. When the user wants to save results, they are presented with an option of selecting to which group the results will belong. The group listing includes all the groups to which the user belongs . This allows the user to share generated results with other members of the group. These results are included in any future searches that might be done against the database. If a user is browsing and analyzing data while not logged in, results cannot be saved.

2.3.3. Searching of Data and Results

FunGIMS contains a large amount of data and the best way to access a specific piece of data is to search for it. FunGIMS provides a search facility across all the data and results saved by the user. This allows the user to search for entries by means of a keyword or phrase, or simply access stored results. A user can select to search across all the data types with a keyword or a specific identifier can be entered e.g. search for “dihydropteroate synthase” or search for PDB id “1eye”. The search is implemented on two levels. The first level is a case insensitive text search across all the fields in `Identifiable` and `Description`. The results from this search are then filtered in the second level of the search, to exclude entries that the user may not see. Users can search a keyword or sid against a specific data type or across all data types. At the time of writing, FunGIMS provided searches across protein structures, sequences, literature and small molecule data sets. A keyword search across all data types will produce a page

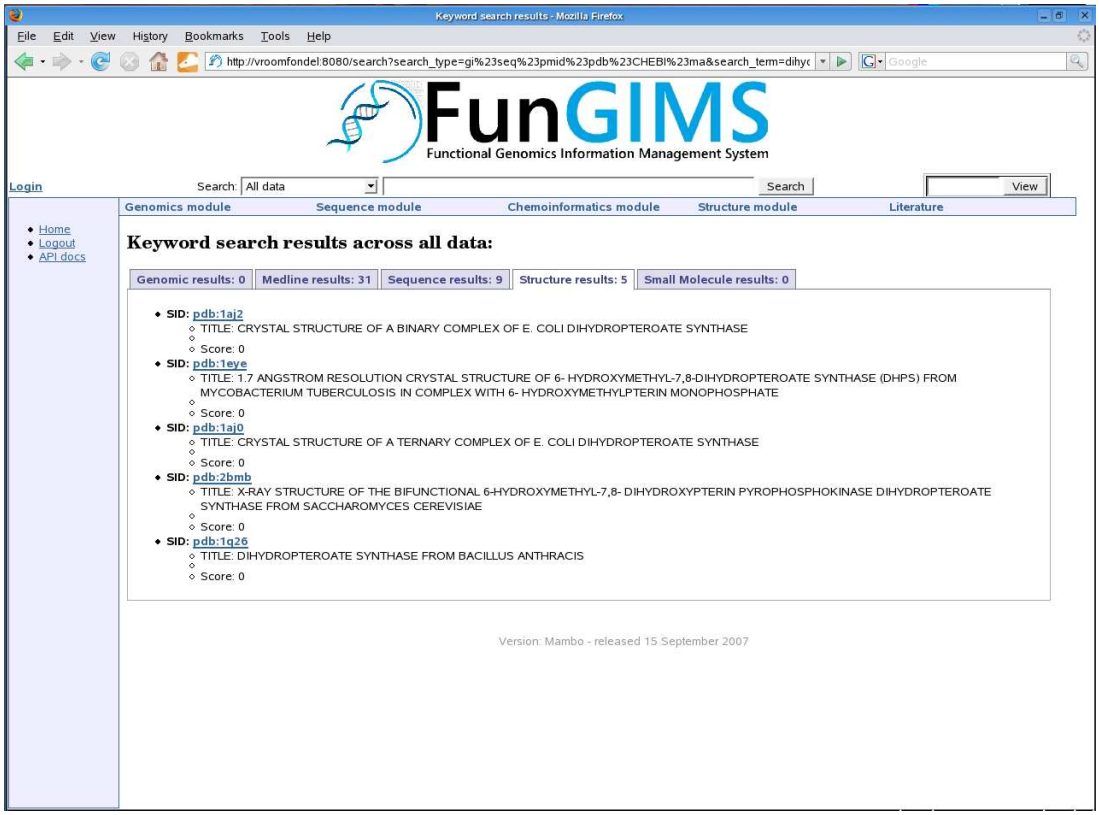

Figure 2.4: The result of a search for "dihydropteroate synthase". The results are ordered according to data type.

with results sorted according to the section they belong to e.g. sequences in the Sequence section and any structure hits in the Structure section. Should a user search for a specific identifier and it is found to be unique, the user will automatically be redirected to a view of the requested entry. Access restrictions are implemented on the searches and thus a user will not see any matches in restricted data. Figure 2.4 shows the results of a search for the keywords "dihydropteroate synthase".

2.4. FunGIMS Data Model

2.4.1. The Data Model

FunGIMS was designed to use one database that contains all the data for each data type in separate tables. In order to incorporate the large amount of data and relationships in FunGIMS, an extensive data model had to be developed. The Functional Genomics

Experiment (FuGE) data model was used as a starting point (Jones *et al.*, 2007, Jones *et al.*, 2006) as discussed in Chapter 1. The FunGIMS data model was extended by inheriting from the `Identifiable` class in FuGE. This allowed for features in FuGE such as `Security`, `Description` and `Audit` to be accommodated in FunGIMS. `Security` implements various features related to the FuGE data model with regard to ownership of the record. `Audit` tracks changes made to a record and `Description` provides a way to add free text descriptions of the record. `Identifiable` consists of a sid, data typename, user id, group id and description id fields. These fields link an `Identifiable` entry to a user, a group, a specific description (which is linked to the `Description` class) and a specific data type. The data typename field is used when constructing the polymorphic joins for a specific module. When a new file or data entry is created in `Identifiable`, the user must also supply the fields required for `Description`. `Description` implements fields for id, description text, keywords and synonyms. When searching the database using a keyword, it is searched against `Description`.

The core data model for FunGIMS extended the FuGE data model by including additional classes to FunGIMS, all of which all inherited from `Identifiable`. These classes include `Note`, `File` and `Relationship`. `Note` is a free text field that allows a user to add free text notes to an entry. More than one `Note` may be associated with a unique `Identifiable` entry. `File` is a class that caters for any files uploaded by the user such as protein models, documents or sequences. One `File` object is linked to one `Identifiable` object. `Relationship` is a class used to link two `Identifiable` entries. This relationship is either user generated or automatically generated from the parsed data. Each specific module extends the FunGIMS data model further and by inheriting from the `Identifiable` class, allows a consistent data model to be maintained. FunGIMS currently implements the following main data type classes: `Structure`, `Sequence`, `MedlineReference` and `Compound`. The specific data model used for the Structural module will be discussed in section 2.5.2. The information in `Identifiable` was also used by SQLAlchemy to create groups of tables in the data model that contains only a certain data type using polymorphic identity joins (creating one object by joining different subclasses from the database).

The TurboGears user tracking/validation data model was used to allow the login of users and to maintain session ids during usage. TurboGears employs a set of tables for users and groups and allows users to belong to more than one group. When a user logs in, they are validated against this data model. When retrieving data belonging to a certain group, the group table is checked to assess whether a user may see the data. A unique session id is generated every time a user logs in and this allows the user to remain logged in to the system for a set amount of time (default is 20 minutes).

2.5. Structural Module

2.5.1. Overview

The Structural module caters for all protein structure data. It allows the user to investigate the protein structures, to conduct analysis on the protein sequences and structure and to generate simulation scripts for proteins. The design of the Structural module was based on the MVC design as shown and used in the rest of FunGIMS. This allows for an extensible and easily upgradable system and further allows for a maintainable code base.

The vast majority of the data in the Structural module is parsed from the MSD discussed in Chapter 1. Most protein structure data is represented in a standard column-based format known as the PDB format (http://www.pdb.org/docs.html). This text format provides structural and administrative information about the protein as well as the Cartesian coordinates of every atom in the protein. Figure 2.5 shows the column layout and an example of the latest PDB file format.

2.5.2. Data Model

The main data model used for the Structural module is based on the MSD (Boutselakis *et al.*, 2003) from the EBI at Cambridge. The MSD provides a very extensive data model to deal with protein structure data. All the data are parsed from PDB and are also linked to primary sequence providers such as GenBank.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
...
ATOM     66 N    VAL A  14      22.866   0.219  42.591  1.00 20.77 N
ATOM     67 CA   VAL A  14      21.639  -0.157  43.253  1.00 26.59 C
ATOM     68 C    VAL A  14      20.898   1.039  43.832  1.00 43.97 C
ATOM     69 O    VAL A  14      19.894   0.894  44.535  1.00 44.07 O
ATOM     70 CB   VAL A  14      21.834  -1.310  44.228  1.00 29.30 C
ATOM     71 CG1  VAL A  14      22.197  -2.582  43.471  1.00 28.10 C
ATOM     72 CG2  VAL A  14      23.022  -0.961  45.095  1.00 36.14 C
...
```

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM "	Record name
7 -11	Integer	serial	Atom serial number
13-16	Atom	name	Atom name
17	Character	altLoc	Alternate location indicator
18-20	Residue name	resName	Residue name
22	Character	chainID	Chain identifier
23-26	Integer	resSeq	Residue sequence number
27	AChar	iCode	Code for insertion of residues
31-38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms
39-46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms
47-54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms
55-60	Real(6.2)	occupancy	Occupancy
61-66	Real(6.2)	tempFactor	Temperature factor
77-78	LString(2)	element	Element symbol, right-justified
79-80	LString(2)	charge	Charge on the atom

Figure 2.5: Top: A protein structure file example (Valine residue 14 from 1eye.pdb). Bottom: the PDB file format specification for ATOM entries.

The MSD data model tries to provide a logical view of protein structure. It is organized into one main entity (`Structure`) that consists of 6 sub-entities (`Active Sites`, `Secondary Structure`, `External Database Links`, `Header`, `Taxonomy` and `Ligands`). Each of these sub-entities are divided into logical groups e.g. `Header` is made up of tables containing information on authors, keywords, X-ray data, etc. In this fashion each sub-entity contains different levels of information. What makes MSD unique and different from the PDB is that for every different feature in MSD, detailed data are available e.g. for every protein atom, the binding order, predicted atom valence, atom type, residue it

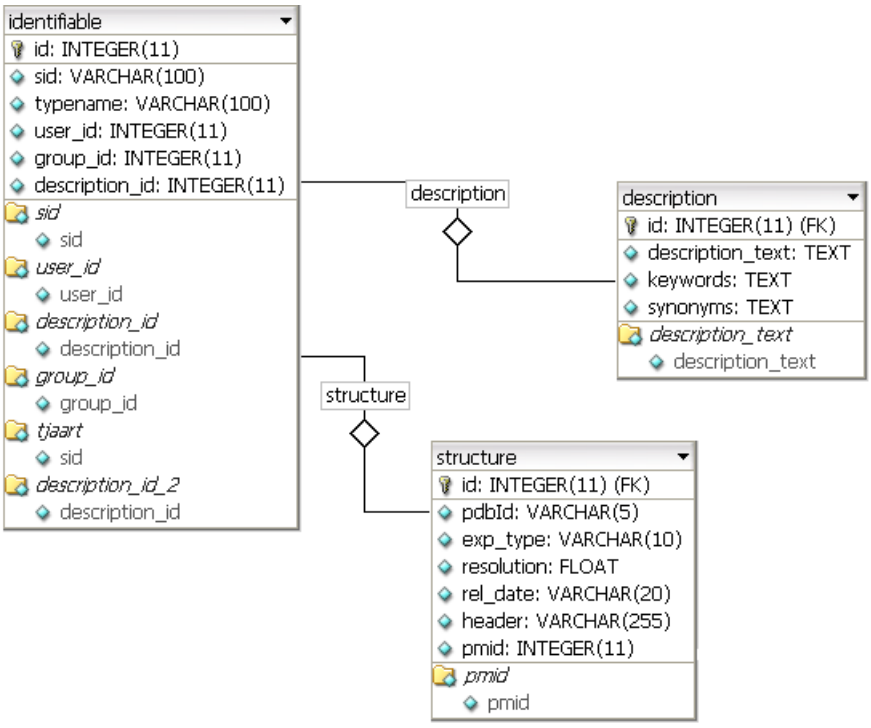

Figure 2.6: The relationship between the `Structure` object and the FuGE data model. `Identifiable` is the main data object in FuGE. `Description` provides some additional data about `Identifiable`. The `Structure` object inherits from `Identifiable` and thus also has `Description` data.

belongs to, other atoms it makes contact with, etc. This makes it one of the most complete structure databases currently available. A complete user-friendly web accessible front end to MSD has been written and is accessible at the EBI's website.

The MSD data model (figure 1.2) was extensively modified before being incorporated into FunGIMS. The Structural module data model consists of the following classes: `Residue`, `Helix`, `Sheet`, `Strand`, `Turn`, `SecondarySummary`, `Tstruc`, `Chain`, `PfamInt`, `ScopInt`, `Go`, `Ec`,`CathInt`, `SwissprotInt` and `Interpro`. All the classes inherit from `Structure` either directly or indirectly from another class. The data extracted and stored from MSD are PDB entry information (`Structure`), protein secondary structure (`SecondarySummary`) including α-helices (`Helix`), β-strands (`Strand`), β-sheets (`Sheet`) and β-turns (`Turn`), protein fold (`Tstruc`) information from CATH (`CathInt`) and SCOP (`ScopInt`), protein classification information from GO (`Go`), Interpro (`Interpro`), Pfam (`PfamInt`) and Swissprot (`SwissprotInt`) as well as EC numbers (`Ec`). Information such as the energy types

of each atom and atom types were not extracted, as the Structural module only caters for a higher level of protein structure. A second set of scripts was then run on the MSD data to extract basic relationships between data types such as linking the Pubmed id with a protein entry and these were stored in the `Relationship` class. Stored relationships are between the protein, Swissprot and GO numbers as well as between the protein and Pubmed. All these generated links were also added to the FunGIMS database. Section 2.5.2.1 discusses other data sources. Most data relating to the detail such as atoms, residue planarity and energy types were omitted. This was due to the fact that the Structural module provides a basic introduction to a structure. Its main purpose is for exploratory analysis and investigation.

The FunGIMS structure data model was constructed to closely represent the actual structure levels in a protein in a top down fashion. This ensures that a protein model can be browsed by starting with the assembly, followed by the local fold, the chain specific secondary structure and finally by residue data (Figs. 2.6, 2.8 and 2.7).

2.5.2.1. Data Sources

The majority of the data in the Structure module, and also FunGIMS, are derived and parsed from public databases such as the PDB, GenBank and SwissProt. In the case of the Structure module, Python scripts were used to parse the flat file format of MSD and to add the data to the FunGIMS database.

FunGIMS also caters for user-generated data. In the Structure module specifically, user-generated data makes up a very small portion of the stored data. This is due to the fact that a model that a user generates will not be parsed and stored in the database as there is no experimental validation of the structure. All generated modelling scripts and models will be stored as files belonging to a specific user and group should the user choose to save the files.

2.5.3. Functionalities

The Structural module has various different functionalities. A user can investigate a protein structure and retrieve information about structural elements, perform motif searches

and structural analysis on a protein sequence, generate homology models or generate scripts for modelling and molecular dynamics. Each of these features will be discussed separately. For the first release of the Structural module it was decided to include tools that are often used by biologists and some tools that are less used but equally valuable and that can provide new insights into their work. The design of FunGIMS and the Structural module allows for the easy addition of new tools by programmers.

The browser-based molecular viewer known as Jmol (http://jmol.sourceforge.net) is one of the features that makes the Structural module very useful. Jmol is a Java-based three dimensional molecular view that can run inside a browser as a Java applet. It uses software to render the proteins and thus does not need expensive hardware such as graphics cards. Jmol was specifically written to allow protein structure files to be displayed and manipulated inside browsers. The user can rotate the protein, zoom in, select different representations of the protein, and various other miscellaneous functions. Jmol can also be run as a standalone Java application, which allows users to download the protein files and work with them in a familiar environment.

In the Structural module, Python is used to parse the data such as residue start and end numbers in a turn or helix, and then use this data to generate buttons which controls various Jmol representations.

2.5.3.1. Structural Data Representation

The Structural module includes all structural data such as primary structure, secondary structure, tertiary structure and atomic coordinates. The first view a user would see when querying a protein is the primary sequence data. This includes the sequence of the protein, the name of the protein and other data parsed from the header such as resolution (Fig. 2.9). The primary view also shows any notes added to the specific protein as well as an atom representation (based on the coordinates in the crystallized structure) of the protein loaded into Jmol.

From the primary view the user can navigate to the secondary and tertiary structure views. The main secondary view contains a summary of all the secondary structure features found in each chain in the protein and provides links to a more detailed view

of each feature. When a specific chain is selected, it takes the user to a summary of the secondary structural features for that specific chain (Fig. 2.10). This includes data on α-helices, β-strands, sheets, turns and other chain features.

A user can also see a summary of all the strands in a specific protein chain by clicking on the strand link in the secondary structure summary (Fig. 2.11). This will provide a page with a summary of the strands found in the protein chain together with their position, length and sheet id as classified in the MSD. A cartoon representation is presented in Jmol and buttons are provided to select the specific strands. These buttons are not always 100% accurate as Jmol interprets residue numbers differently than those found in the MSD due to missing residues in the protein crystal structure. This is due to the fact that sometimes part of the protein does not crystallize or only a truncated peptide was used. Thus, those residues do not get used when assigning numbers to the residues found in the crystal structure. A user can also select the sheet link and see the number of sheets in a protein structure.

A user can also access data about the α-helices in the protein chain (Fig. 2.12) from the secondary structure summary. This view gives an overview of the number of helices as well as their length, start and end residue numbers. A cartoon representation is also displayed with Jmol buttons for highlighting the helices. Information about β-strands and β-sheets can also be accessed from the secondary structure summary.

Information about all the turns in a protein can also be accessed from the secondary structure summary page. This option presents a user with a table of all the turns that occur in the protein as well as the turn type and class, start residue, end residue and a Jmol representation with Jmol buttons to select all the turns (Fig. 2.13).

In addition to the secondary structure summary, a user can also access information about the tertiary structure of the protein (Fig. 2.14). This view includes the Pfam (Finn *et al.*, 2006), CATH (Pearl *et al.*, 2005), SCOP (Conte *et al.*, 2000), GO (Ashburner *et al.*, 2000) and Interpro id's (Zdobnov and Apweiler, 2001) associated with each chain. Once again Jmol is also present but in this case the protein is shown in a ribbon representation coloured by chain.

The Structural module of FunGIMS contains tools related to secondary and tertiary structure as well as protein sequence feature prediction. Although the database (see section 2.5.2.1) provides most of the structurally derived data, a user may want to do a re-analysis of a structure or use the tools to analyze a new structure or model or protein sequence. At the time of writing, only X-ray data was supported. The structural module can be divided into roughly two parts, a structural data part and a analysis tools part.

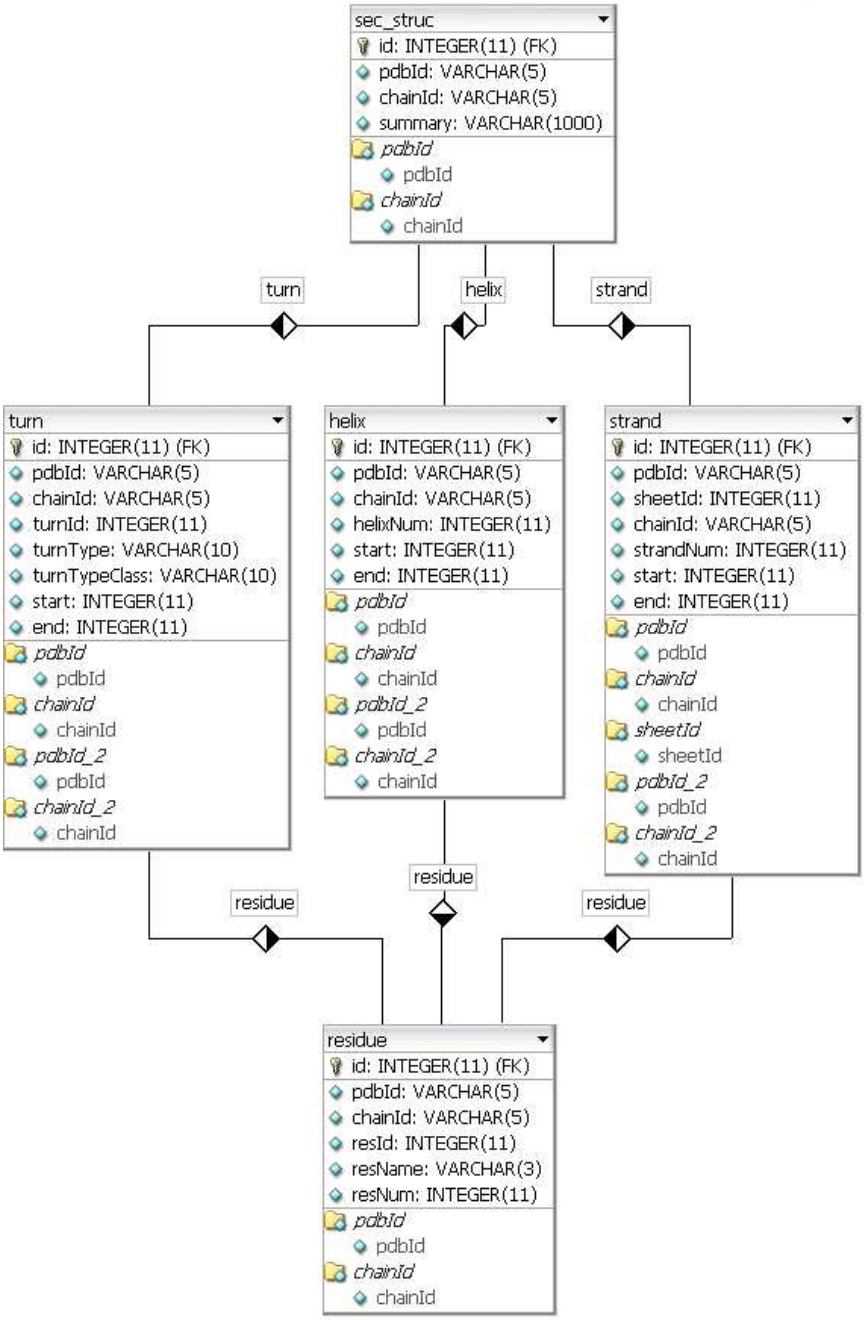

Figure 2.7: The relationship between different secondary structures in a chain and the residues in a protein. This provides the clearest example of how the data model organization follows the logical, hierarchical organization seen in a protein structure. Each secondary structure (`sec_struc`) object has several features such as a `helix` or a `strand` or a `turn`. And each of these specific secondary structural features also consists of a `residue` thus following the inherent logic in a protein structure. Due to the levels of inheritance, each `residue` object still has an `identifiable` and `description` object associated with it.

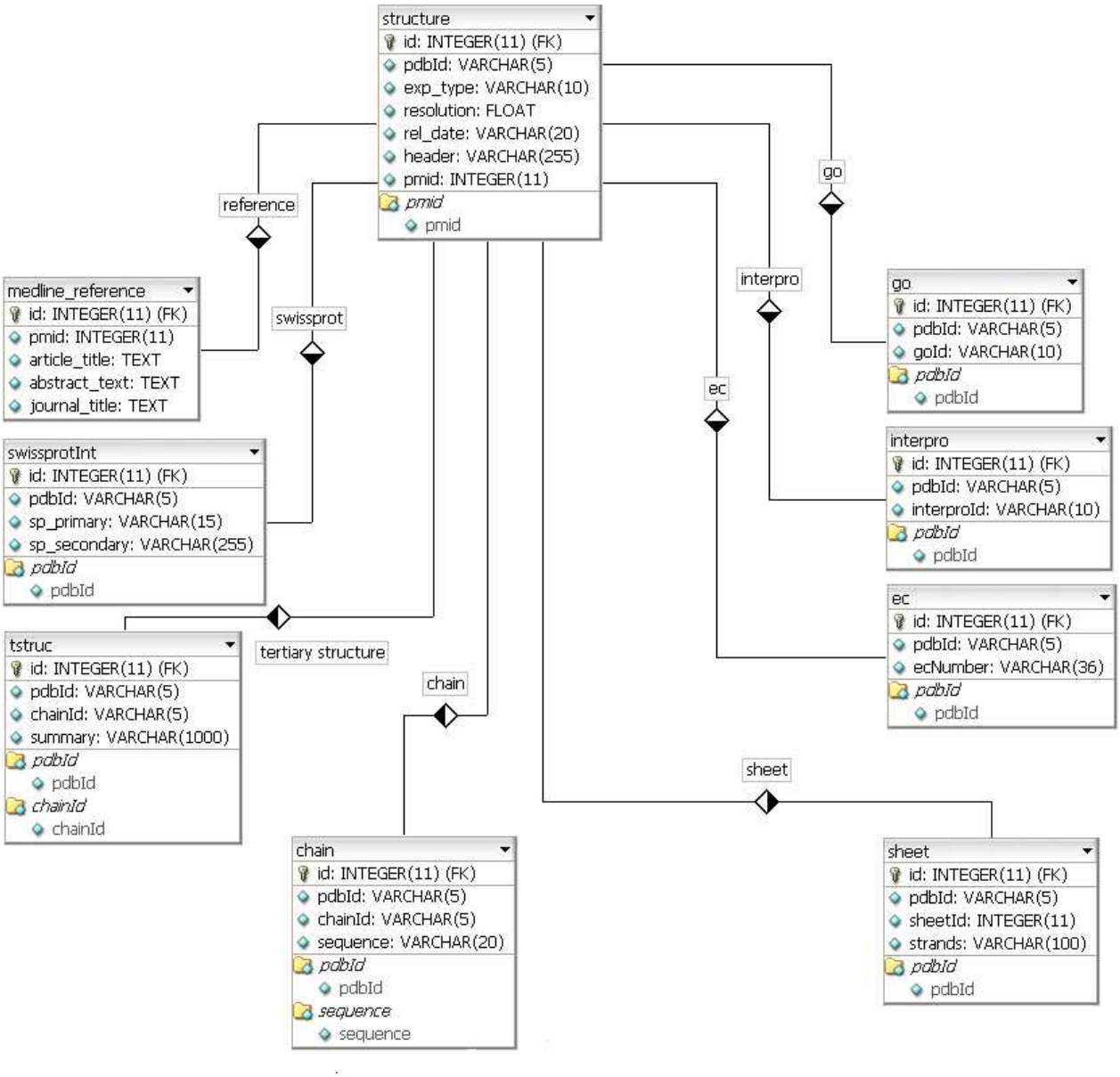

Figure 2.8: The data model for the high level `Structure` class. A `Structure` entry is linked to its reference (Pubmed) as well to high level classifiers such as `Interpro` and `GO`. The different organization levels can be seen clearly e.g. a `Structure` consists of one/many `Chain` objects and each `Structure` object also has other high level features such as a SwissProt id (`swissprotid`).

2.5.3.2. Data Analysis

The second part of the structural module is the data analysis tools (Fig. 2.15). This provides web interfaces to some commonly used tools in protein structural analysis. All these tools are external programs that are called using Python 2.4 system calls, and the

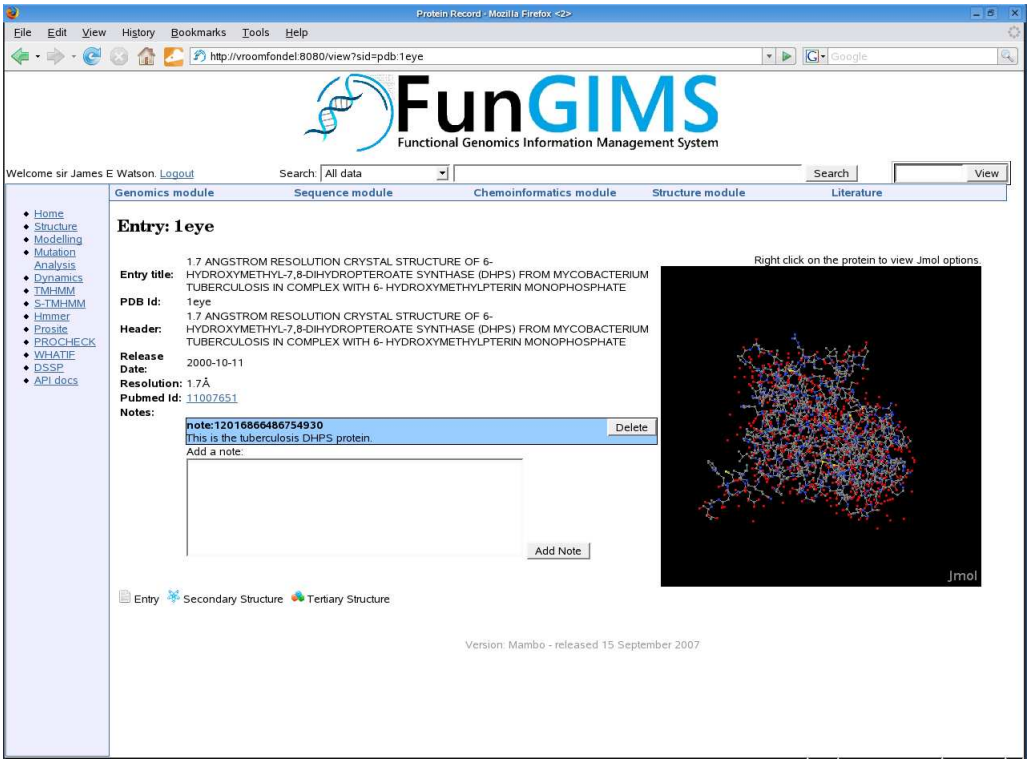

Figure 2.9: The primary view when a user views a protein. Note the general FunGIMS feature where an entry can be annotated by a note.

results are displayed to the user. Each program has a unique script located in the `utils` folder of the FunGIMS.

Users are able to analyze a protein sequence using these tools. The tools currently implemented in the Structural module are:

- Hmmer search against Pfam - Hmmer is a hidden markov model-based (HMM) search tool that tries to identify a protein sequence by matching it to a database of protein families (Finn *et al.*, 2006). Hmmer takes the sequence, an E-value cut-off and a database to search against. The output contains a list of families that matches the user submitted sequence. It also includes confidence values for every hit found to a protein family. The `hmmer.py` script in `utils` is used.
- TMHMM - TMHMM is a HMM-based tool for searching for transmembrane helices based on the amino acid sequence found in a protein sequence (Sonnhammer *et al.*, 1998). It takes a protein sequence as input and produces a graph showing which areas

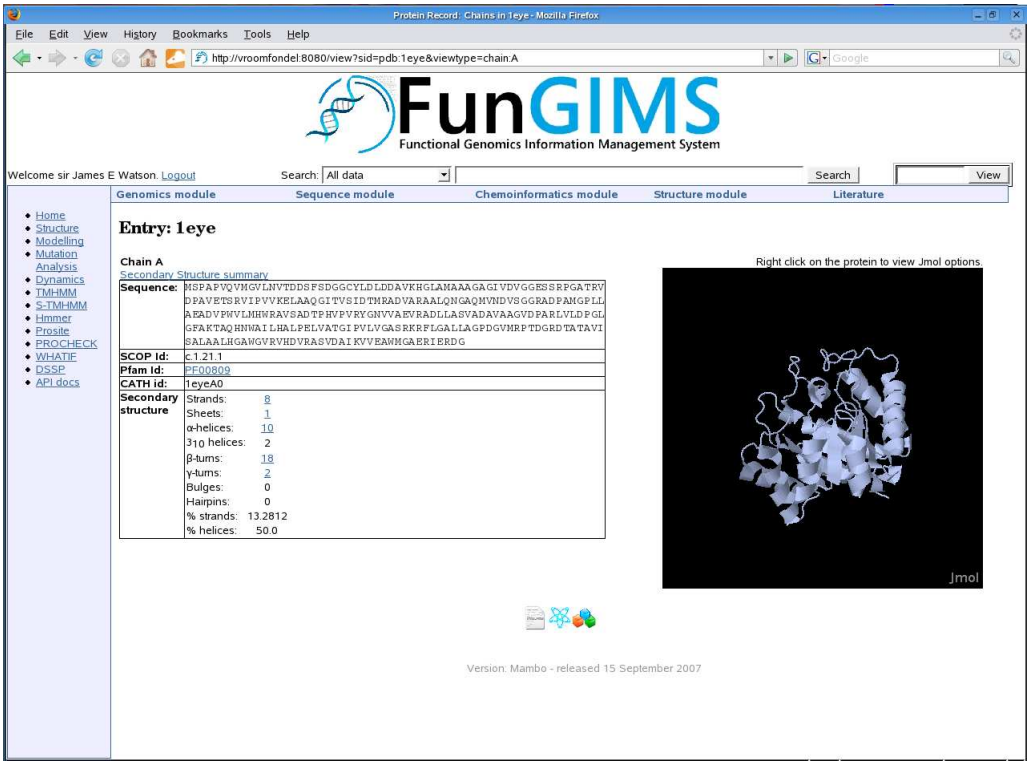

Figure 2.10: The chain summary view for a specific chain in a protein.

are predicted to contain transmembrane helices. The `tmhmm.py` script in `utils` is used.

- S-TMHMM - This tool tries to predict the topology (inside/outside) of any transmembrane helices found in a protein sequence (Viklund and Elofsson, 2004). It takes a protein sequence as input and produces a table showing the probability of each residue being inside or outside the membrane. The `stmhmm.py` script in `utils` is used.
- Prosite - Prosite is a database of protein motifs (de Castro *et al.*, 2006). These include short motifs such as glycosylation sites as well as longer motifs that can identify a specific protein family. To search Prosite, the ps_scan.pl script from the EBI is used. Using a protein sequence as input, it produces a list of motifs found in the protein. Flags can be set to exclude motifs with a high probability of occurrence, but this has not been implemented in the Structural module. The `prosite.py` script in `utils` is used.
- PROCHECK - This allows a user to check a protein structure file for any abnormal structural errors (Laskowski *et al.*, 1993). The checks are based on a set of normal

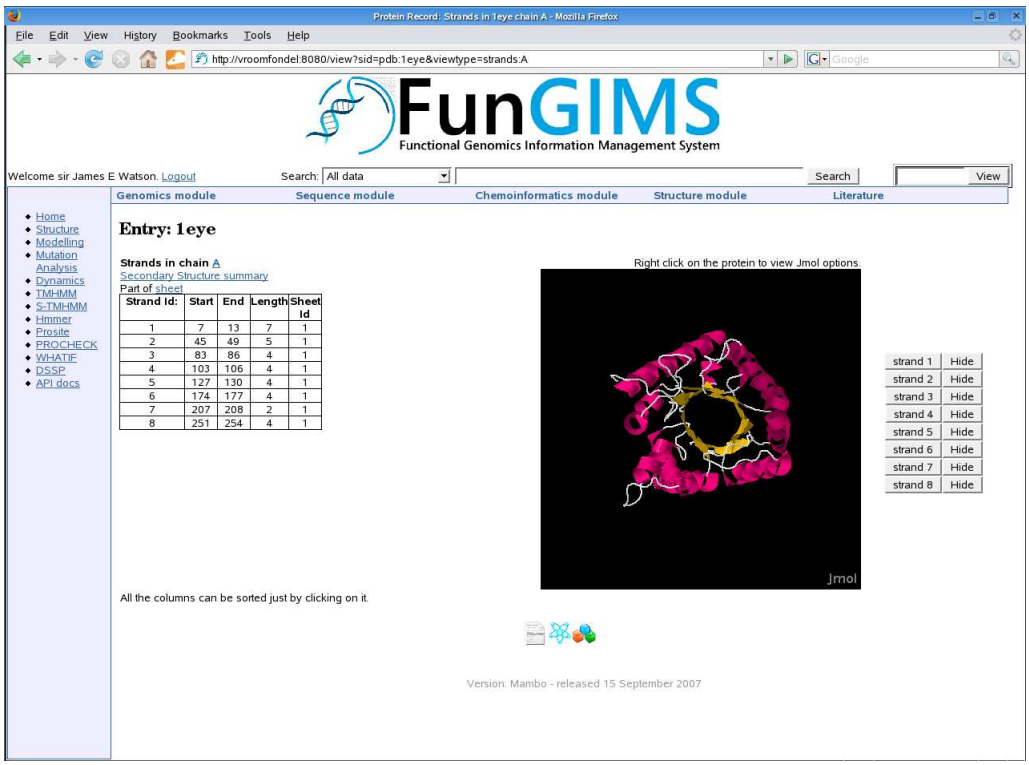

Figure 2.11: The strand summary page for a protein chain.

structural parameters derived from the PDB. The input is a protein coordinate file and it produces a set of ten files that include Ramachandran plots, graphs plotting the deviation of each amino acid type from normal as well as a summary. In the Structural module the user can download each file for later use. The `procheck.py` script in `utils` is used.

- WHAT IF - WHAT IF is a comprehensive set of tools for molecular modelling and for analyzing proteins in their native environments (Vriend, 1990). The structure checking tool was implemented in the Structural module and this does a range of checks on a submitted protein file to identify possible errors and warnings. It produces a detailed report on the structure analysis that the user can download. The `whatif.py` script in `utils` is used.
- DSSP - This program calculates secondary structure based on the coordinates of the atoms in a PDB file (Kabsch and Sander, 1983). The program takes a pdb file as input and produces a report that gives the secondary structure of each amino acid. The `dssp.py` script in `utils` is used.

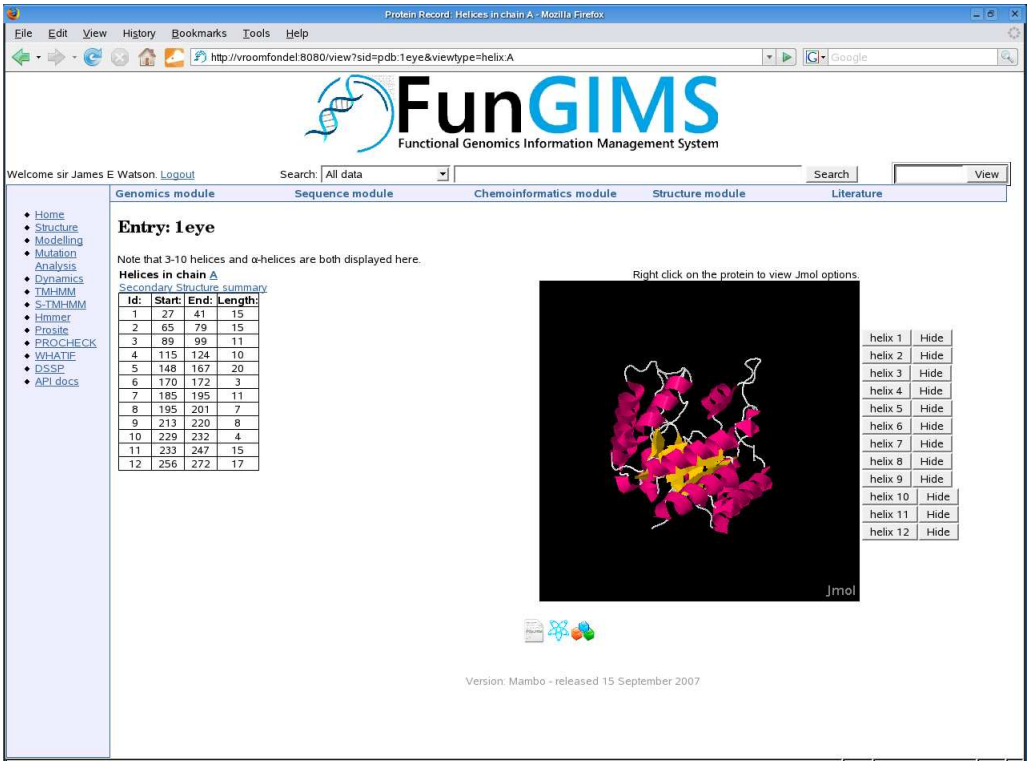

Figure 2.12: The α-helix summary view for a protein chain.

All these tools accept either a file or a sequence from the user. The selected tool is then run via a tool-specific Python script, which thereafter uses Python system calls to run the appropriate tool on the sequence or file. The scripts for each tool are saved under the `utils` directory. All the results are saved on disk during the session. The results are also displayed to the user and the option to save the results to a certain group is available. Figures 2.16, 2.17 and 2.18 show the results from an analysis run of TMHMM, Hmmer against Pfam and a PROCHECK analysis.

2.5.3.3. Modelling and Molecular Dynamics

The third section of the Structural module has functions that allow the user to generate scripts for homology modelling and molecular dynamics (Fig. 2.15) and build models. For protein homology modelling the user has a choice between two programs, Modeller (Fiser and Sali, 2003) and WHAT IF (Vriend, 1990). The module will ask for the relevant information, pass it to the specific script located in the `utils` folder, and produce a script, using Python, which the user can download and run on his or her local machine. This

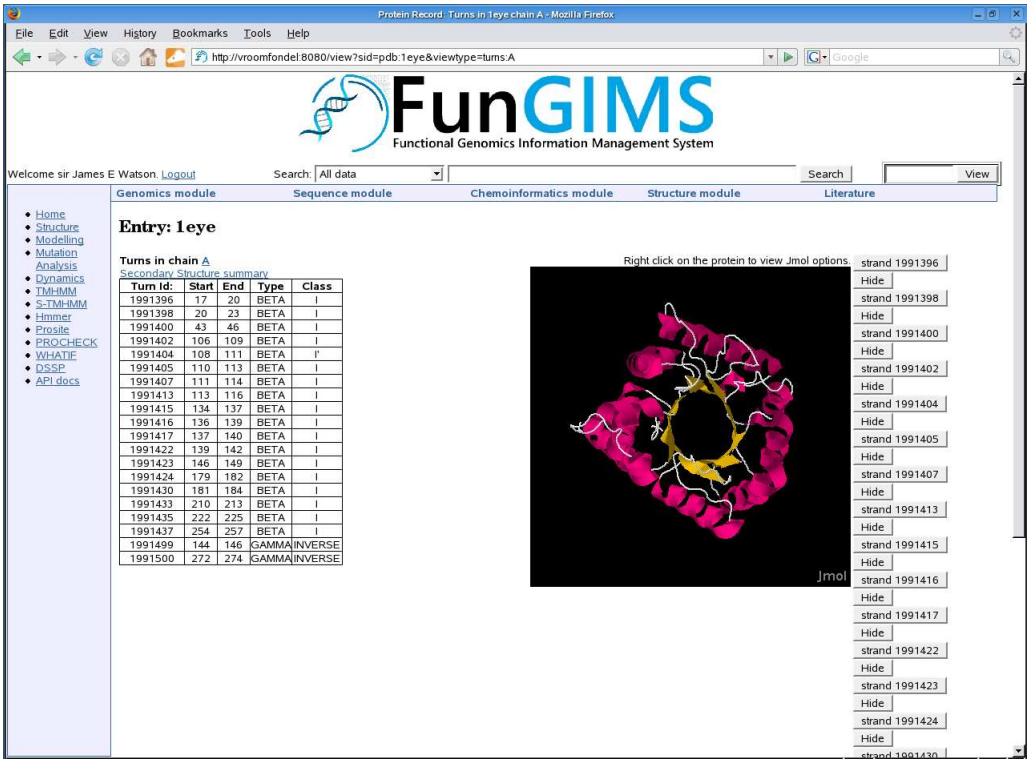

Figure 2.13: The summary view for all the turns that occur in a protein chain.

precludes the user having to actually set up and understand the scripts and scripting language. In addition to the modelling scripts, the user may also decide to construct a model using the automatic method in the Structural module (Fig. 2.20). The user enters a template PDB id, target name, target sequence and refinement level. This will be passed to Modeller (version 9v1), which will perform an automatic alignment of the two sequences and then proceed to build a model. Currently the automated modelling process uses the first chain in a multi-chain protein as a template. When the model is ready, the user is alerted and presented with a page to download the model, modelling script and alignment file. A drawback of the automated modelling is the automated alignment performed by Modeller. When the sequences display a high identity, alignment is easy and should be accurate. However in lower identity ranges (less than 40%), automated alignment is not as accurate and it is advisable to do the alignment with manual curation of the results.

The module can also generate basic scripts, using Python, for three different molecular dynamics suites, (NAMD (Phillips *et al.*, 2005), CHARMM (Brooks *et al.*, 1983) and

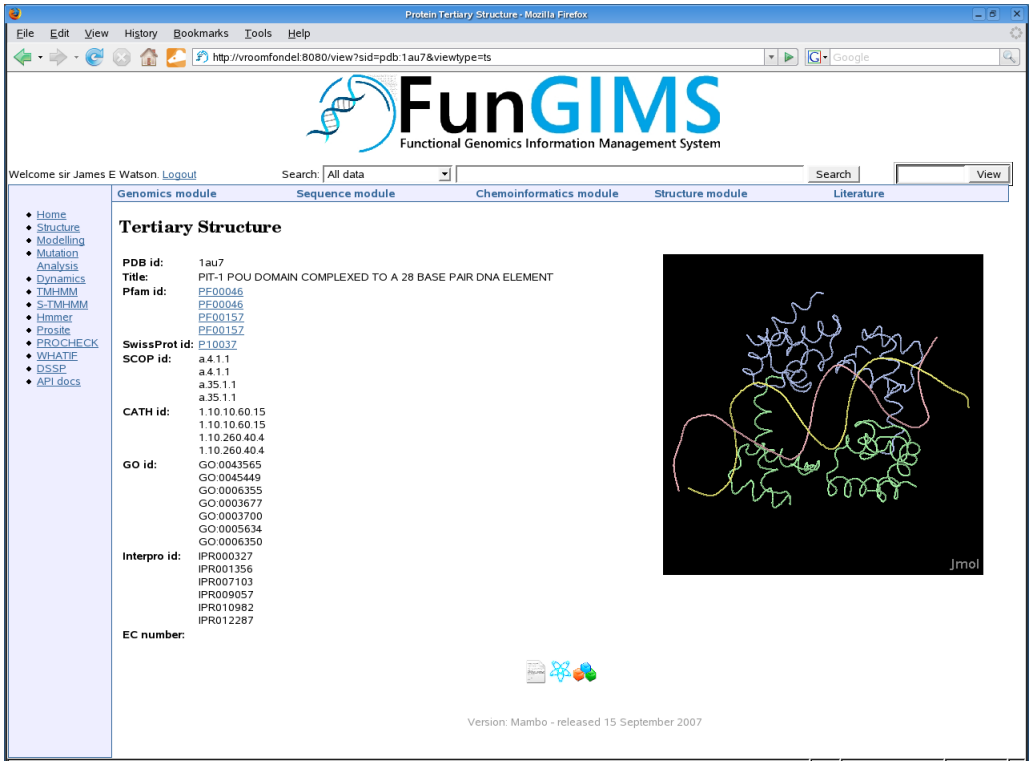

Figure 2.14: The tertiary structure view of a protein. This shows information for the complete protein complex.

Yasara (http://www.yasara.com) given user input. The dynamics section only supports script generation, not running the actual simulations as this is extremely resource intensive. This allows the user to focus on the research questions without the need for technical knowledge. Figure 2.20 shows the interface for the molecular dynamics script generation section. The molecular dynamics scripts will need further editing depending on the molecule the user wants to investigate and the type of dynamics. All the modelling functionalities are located in the `utils` folder and the `modelling.py` script is used. For dynamics the `dynamics.py` script in `utils` is used. While validated homology programs are used, the quality of a model is determined by various factors such as template resolution, template-target alignment and the specific algorithm used.

The running of simulations in a UNIX environment will still require some skills and UNIX knowledge but an IT support person should be able to assist with the installation of the programs. The interpretation of the dynamics results are up to the user as automated analysis is not really a possibility yet. The intent is to provide the user with basic access

Analysis

Tool	User Input	FunGIMS DB	Method	Output
TMHMM	Protein sequence		tmhmm	.png figure
S-TMHMM	Protein sequence		stmhmm	Table
Hmmer vs Pfam	Protein sequence E-value Database	PDB id	hmmer	Table with hits
Prosite	Protein sequence		prosite	Table with hits
PROCHECK		PDB id	procheck	.png+.ps figures
WHAT IF model analysis	Protein model		whatif	.tex+.txt report
DSSP		PDB id	dssp	Table

Modelling

Tool	User Input	FunGIMS DB	Method	Output
Modeller - model	Protein sequence Protein id Refinement	PDB id	modelling.modeller_script	Protein model Script file Alignment file
WHAT IF - model	Protein sequence Protein id Model name	PDB id	modelling.whatif_script	WHAT IF script file
Modeller - mutation analysis	Template Mutations Script name		modelling.mutate_model	Modeller script file
Dynamics	Protein name Program Minimization steps Temperature Time step size Simulation time Solvation shape		dynamics	Program specific dynamics script

Figure 2.15: The different tools available in the Structural module. Shown are the input (user and FunGIMS supplied) required for each of the tools, the specific method called in the `utils` folder as well as the type of output the tool generates.

to molecular dynamics functionality but guidance in the interpretation of the results is currently outside the scope of the system. It is always recommended that the user consult suitable literature when engaging in any form of advanced simulations.

2.5.3.4. Help Section

FunGIMS was designed to assist biologists to conduct faster and easier analysis and exploration of data. To further this goal, a help page is provided for each function in the Structural module. This can be accessed by clicking on the link found on each page. To increase visibility it has been labeled in red. Figure 2.19 shows a typical result when a user clicked on a help link for a specific function. The help link provides a brief synopsis

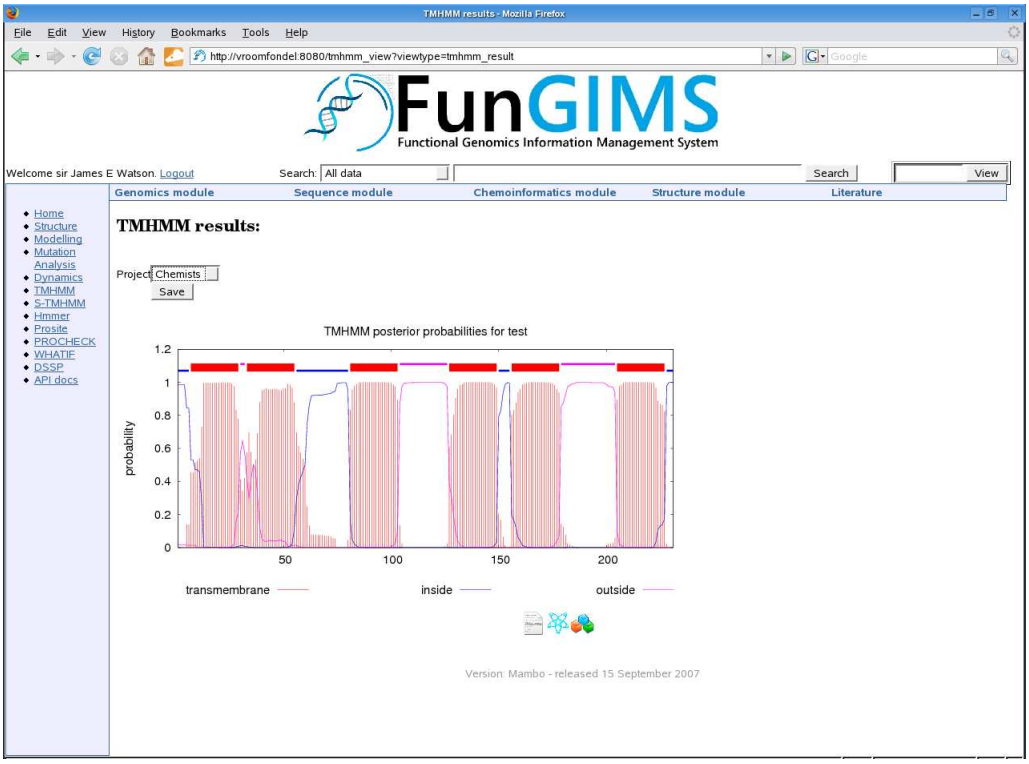

Figure 2.16: The results from a transmembrane helix prediction on a submitted protein sequence. The drop-down menu allows the user to save the results to a specific group.

of the tool and the inputs required, as well as the output a user might expect when the tool runs successfully.

2.5.3.5. Configuration

The Structural module relies on various external programs to provide analysis methods. Installation locations and execution of these programs usually differ between machines and programs. To overcome this, a configuration file (`utils/config.py`) was created that stores all the program specific settings. This file can be edited by hand to change program properties. For each program the following properties are specified: the path to the program (executable file), a program-specific temporary directory for output, and other program specific parameters and settings. These programs are then called from inside the Structural module simply by referencing these variables. This makes system administration far easier as program settings have only to be specified and changed in one file.

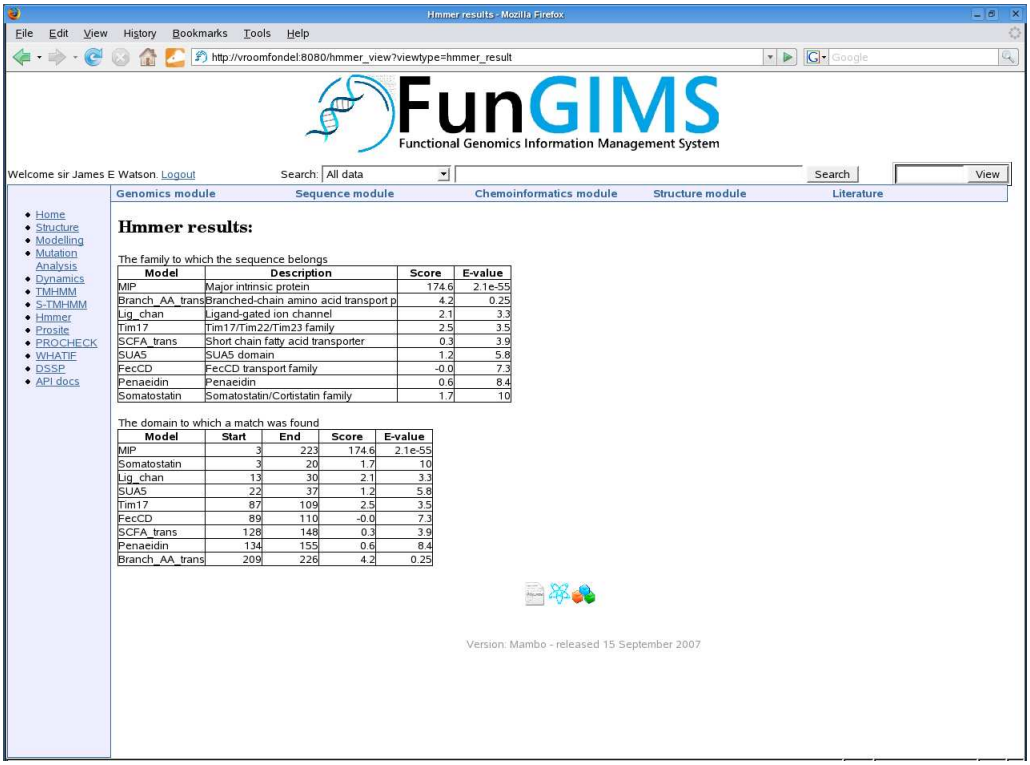

Figure 2.17: The results from a Hmmer search across Pfam using the structural module.

2.6. Future Improvements

2.6.1. FunGIMS

A system such as FunGIMS is in a constant flux of development. FunGIMS was designed to allow for the easy addition of new tools and features. There are a number of areas that can be improved upon, the database being one of them. Database table optimization would allow for queries to be dealt with faster. Distributed databases would lessen the load on the server when the database size increases significantly. In the current implementation of FunGIMS, the database size presented some challenges and smart indexing of often-queried columns in tables resulted in a decrease in query time. The database should also be expanded to include more detailed data types such as protein chip array data.

Furthermore, smart file recognition and improved file parsers would enable the user to upload a file, allow FunGIMS to parse it entirely and then insert the data into the

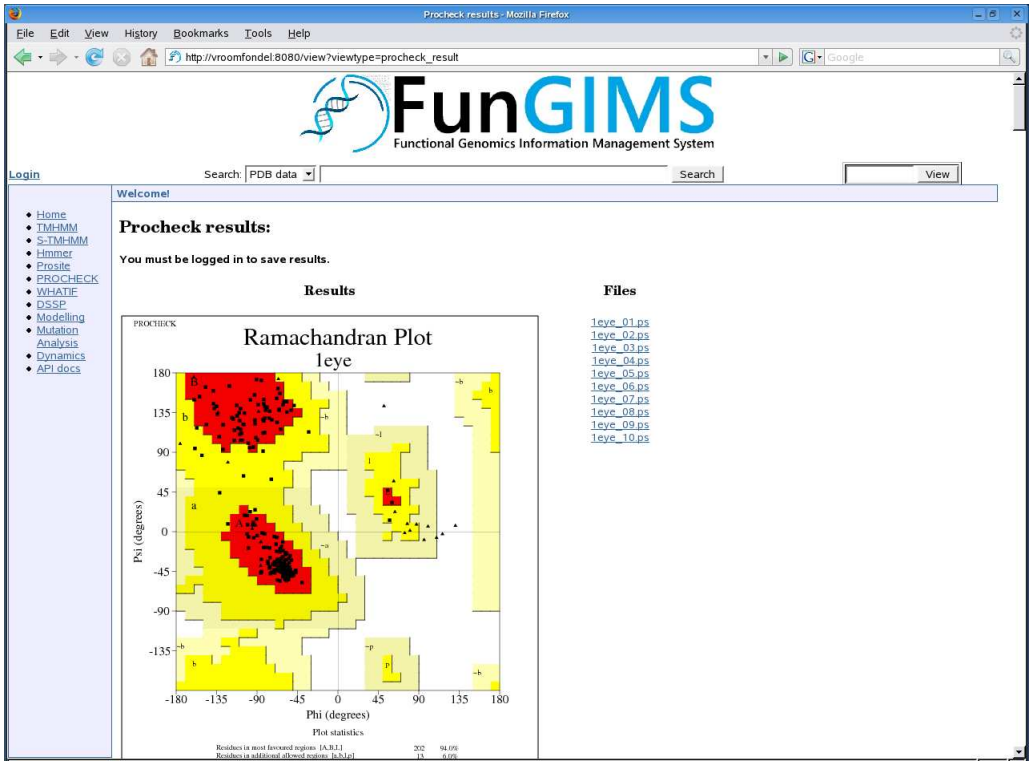

Figure 2.18: The results from a PROCHECK analysis run on PDB 1EYE.

database, not merely as a file but as a full data type. This allows queries to be more accurate as uploaded files will be parsed and stored in a data type specific manner. Automatic link generation between entries would be another major benefit to FunGIMS. Currently links between entries are generated when the database is first populated with public data and when a user links to entries with a note. Automatic link generation would navigate free text fields, notes and description text and then create the appropriate links. This automatic link generation tool should run on a daily basis so that links are always up to date.

2.6.2. Structural Module

In addition to the improvements to FunGIMS mentioned in the previous section, the Structural module also has some possible improvements.

More analysis methods can be included for different features. Tools such as consensus sec-

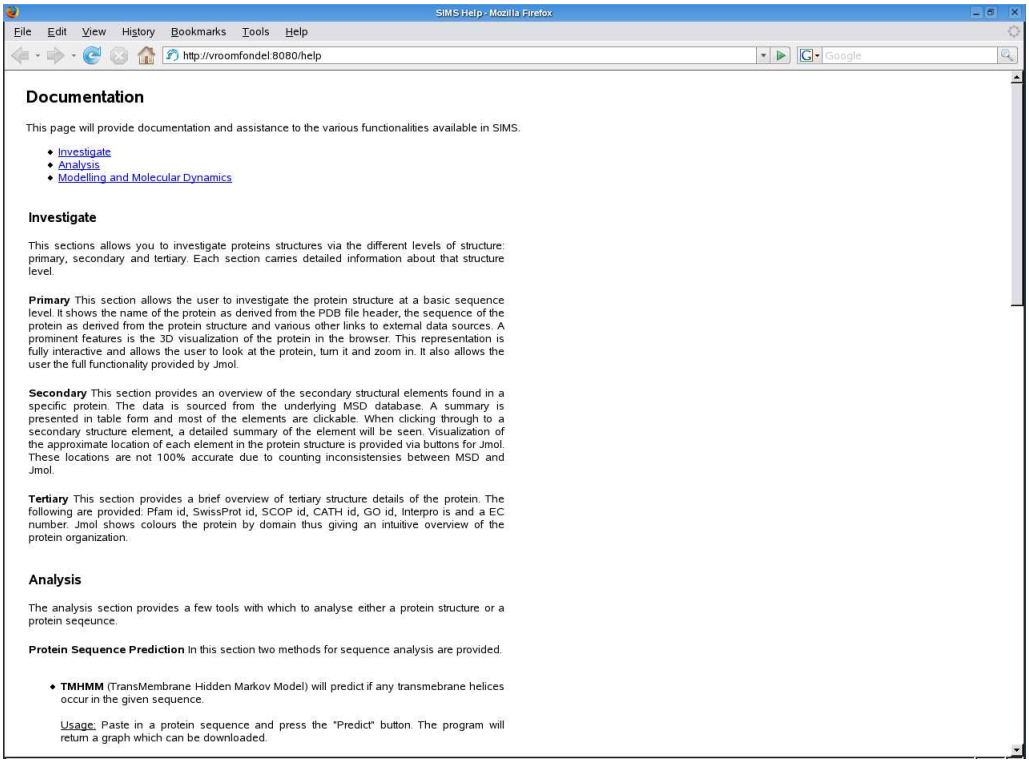

Figure 2.19: The help section for the Investigate section. Each function has its own help section on the Help page.

ondary structure prediction, protein export signal prediction and other protein sequence analysis tools will be a benefit to the system.

The most improvement is probably in the modelling and simulation section. The current scripts can be modified to include modelling on the selected chain of a protein, on multiple templates as well as including ligands in the modelling process. A feature could also be implemented to use alignments provided by the user. More simulation scripts with different parameters and environments could also possibly be added. A possible addition could be the implementation of a module whereby a user can start a simulation on a cluster or another computer while being able to control it from the FunGIMS system. This will allow the user to run simulations on various machines without needing the technical knowledge.

There is scope for the improvement of the user interface of the Structural module. Jmol buttons for secondary structure elements can be made more accurate. In addition a visualization library can also be included to generate scalable images of a summary of

the secondary structure elements found in a protein and present them to the user in a downloadable format. A useful improvement would be scripts that facilitate a more automatic update of the database as soon as the data sources used, are updated. This would lessen the load on the site administrator and would keep the database up to date.

2.7. Conclusion

FunGIMS consists of various modules dedicated to different data types. The Structural module currently provides functions to explore structural data for a specific protein, conduct analysis on a user-submitted protein structure, including analysis such as transmembrane helix prediction, Prosite motif search and also allows the user to create homology modelling and molecular dynamics scripts. The application of the Structural module to various problems in FMDV will be discussed in the next three chapters.

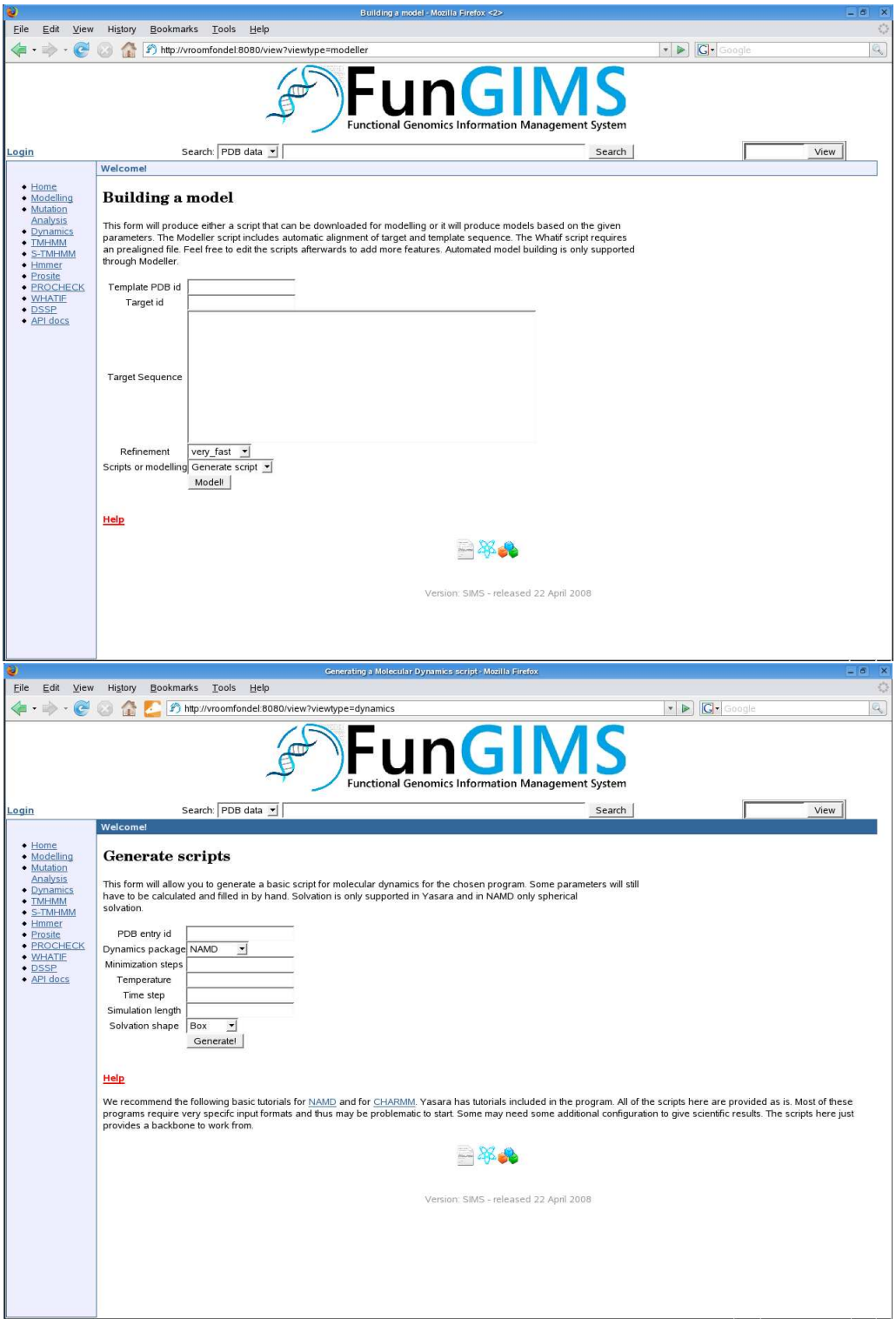

Figure 2.20: Top: The automated modelling interface when building a model using Modeller. The user can decide to generate homology modelling scripts for Modeller or WHAT IF. Bottom: The molecular dynamics script-generating interface. Users can select between the different programs from the drop-down menu in the form.