A Comparative Anatomy Information System for the Foundational Model of Anatomy and Mouse Anatomy Ontology

Technical Report written by Kremena Diatchka
Undergraduate Student,
Oregon State University
For
CRA-W Distributed Mentor Project (DMP) Summer 2005 Final Report

Contributors, collaborators, and mentors:

Ravensara Travillian, Graduate Student, University of Washington
Katarzyna Wilamowska, Graduate Student**,** University of Washington
Dr. Linda Shapiro, Professor of Computer Science**,** Professor of Electrical Engineering, University of Washington
Tejinder Kaur Judge, Undergraduate Student, University of Wisconsin-Madison

Introduction

Research in bioinformatics, genomics, and animal models of human disease, as well as other fields, has shown an increasing need for extrapolating information from one species to another [1]. In other words, it would be very helpful for contemporary researchers to be able to compare anatomical entities of two different species. For example, if experiments are conducted on a certain part of a mouse, it is critical to know how closely the relevant anatomical entity of the mouse models the corresponding human entity, so that it is easier to evaluate the significance of the results for humans. Furthermore, the amount of anatomical and related medical data emerging from animal modeling experiments is growing at an exponential rate, calling for innovative methods of evaluating, organizing and managing this information.

Ravensara Travillian, a graduate student supervised by Dr. Linda Shapiro at University of Washington, is working on the design of a comparative anatomy information system that will allow users to issue queries to determine the similarities and differences between two species. The system will serve as a pilot project for cross-species anatomical information collection, storage, and retrieval. In her previous work with her collaborators, Ravensara proposed an approach called the Structural Difference Method (SDM) which uses the Foundational Model of Anatomy (FMA) as a framework (see **Background** section for details about the FMA), and graph matching as a method, for determining the similarities and differences between node attributes and relationships (edges) defined by the attributed graph of the FMA. [1]

A prototype of the proposed system needed to be implemented to demonstrate the functionality of the theoretical design; this was an ideal summer project for two undergraduate students, Tejinder and I, and we have worked on creating a working version of the application as our DMP Summer 2005 project at the University of

Washington. This paper describes the background knowledge that we acquired before we began our implementation, the details of our work, and the challenges and problems we faced.

Knowledge representation

The Foundational Model of Anatomy is developed and maintained by the Structural Informatics Group (SIG) at the University of Washington. The FMA is symbolic model of the physical organization of the human body. More specifically, it is an ontology which furnishes a comprehensive set of concepts and relationships which describe the human body at all levels of structural organization [6]. It is an evolving resource for bioinformatics projects that require anatomical information [2]. Therefore, the FMA is well suited as a framework for the knowledge base of the application we are developing.

The FMA was implemented using Protégé 3.0 - a free, open-source ontology editor and knowledge-base framework [4]. Protégé was developed by the medical informatics group at Stanford University.

Protégé represents information as *frames*. A frame represents a concept or a situation, and in the case of the FMA, a frame stands for an anatomical class. Different kinds of information can be attached to a frame to define or describe the purpose of the frame [5]. In Protégé, this information is represented as slots, which have slot values. The slots stand for attributes and relationships relevant to the anatomical class.

We can imagine the FMA as a graph, with each anatomical class being a node, nodes having attributes and attribute values, and edges of the graph representing relationships between nodes. In Protégé, slots of the class represent either node attributes or edges and slot values represent either node attribute values or edge attribute values. For example, the anatomical class `Prostate` has the Boolean attribute (slot) *has boundary,* and the attribute value (slot value) for that attribute is *true.* There is also a directed edge from the node `Prostate` to the node `Cavity of Male Pelvis`, which represents the relationship *contained in;* in Protégé, the class `Prostate` would then have slot called *contained in* which would have the slot value `Cavity of Male Pelvis.`

It was decided that initially, our application will be designed to compare structures from two species – the human and the mouse. Therefore, we are developing a partial Mouse Anatomy Ontology (MAO) [3]. Currently, the knowledge base that is used for making cross-species comparisons is a hybrid between the FMA and the MAO, as it contains anatomical information about both species under one general abstraction. Like the FMA, it is implemented in Protégé 3.0.

We began the development of the knowledge base by entering anatomical information about the mouse and human prostates and the mouse and human mammary glands. Those structures have been identified as cancer sites and were selected because of their medical importance. Currently, the knowledge base contains somewhat complete

information about the mouse and human prostate, but does not contain any mammary gland information.

Comparing structures

Ravensara developed the design of the comparative anatomy system to map the anatomical entities of one species to those of another species, and to determine the similarities and differences between these entities [1,3].

A mapping is a correspondence between a structure in one species and a structure in the other species. A mapping answers the question, "What is the structure in species1 that corresponds to a structure in species2?" For example, the answer to the question "What is the structure in the human that corresponds to the set of prostates in the mouse?" is the human `Prostate`, which means the human `Prostate` maps to a set of mouse prostates. The structures which are mapped across species are selected based on homology (evolutionary relatedness), and not on homoplasy (similarity of appearance) or analogy (similarity of function) [3].

It is important to note that a mapping is bidirectional i.e. if structure A in species 1 maps to structure B in species 2, then necessarily structure B in species 2 maps to structure A in species 1. The user expects the query "What is the difference between the human and mouse prostates?" to return the same result as the query "What is the difference between the mouse and human prostates?" and this implementation ensures this consistency in response [3].

We implement mappings in the knowledge base in Protégé by determining corresponding entities in the two species, creating a *maps to* slot for the appropriate anatomical class, and filling in the slot value with the corresponding class from the other species.

In order to gauge the difference between two structures, the following types of differences have been proposed: node set differences, node attribute differences, node attribute value differences, and relationship differences. A very brief explanation of each follows, as they are described more fully in [1]. A more detailed description of how the difference types were implemented is included in the **Implementing the proposed design** section.

Node set differences – the difference between the number of entities in the source and target species i.e. a mapping difference where a structure in one species corresponds to more that one structure in the other species, or there is no correspondence at all (a null mapping).

Node attribute differences – differences in the existence of an attribute between two corresponding structures in the source and target species.

Node attribute value differences – differences in values of corresponding attributes shared between corresponding nodes of two species.

Relationship differences – differences in relationships (edges) between structures across species.

Implementing the proposed design

Protégé is based on Java [4], and our application accesses the knowledge base through the Java methods of the Protégé Application Programming Interface (API). Therefore, our application is written in Java.

Query types

This section describes the query types we implemented and the approach we took comparing structures, based on the theory described in the previous sections. We imagine the generic structure of a query like this:

Anatomical entity A in species 1 <*query type*> anatomical entity B in species 2

We often refer to "Anatomical entity A in species 1" as the subject of the query and "anatomical entity B in species 2" as the object of the query for simplicity.

We implemented seven query types; two of them are Boolean queries, and five are non-Boolean queries that return a set of results. Below is a description of the results of each query type.

Boolean queries

The Boolean query operators are *is different?* and *is homologous?*.

- **Species1.anatomical-entity1 *is different?* species2.anatomical entity2**

  Returns true if species1.anatomical-entity1 does not map to species2.anatomical-entity2, and false if the two entities map to each other.

- **Species1.anatomical-entity1 is *homologous?* species2.anatomical entity2**

  Returns false if species1.anatomical-entity1 does not map to species2.anatomical-entity2, and true if the two entities map to each other. This operation is the inverse of the *is different?* query type.

Non-Boolean queries

The non-Boolean operators are *differs from*, *similar to*, *shared*, *not shared* and *union*.

- **Species1.anatomical-entity1 *differs from* species2.anatomical-entity2**

  Returns the difference between anatomical-entity1 in species1 and anatomical-entity2 in species2, as computed by the ***comparing structures*** method described below.

- **Species1.anatomical-entity1 *similar to* species2.anatomical-entity2**

Returns the similarities between anatomical-entity1 in species1 and anatomical-entity2 in species2, as computed by the ***comparing structures*** method described below.

- **Species1.anatomical-entity1 *shared* species2.anatomical-entity2**

Returns the set of parts in anatomical-entity1 and anatomical-entity2 that map to each other. In other words, the parts of the two entitities that are homologous.

- **Species1.anatomical-entity1 *not shared* species2.anatomical-entity2**

Returns the set of parts in anatomical-entity1 and anatomical-entity2 that do not map to each other. This is the inverse of the *shared* query type, and it returns the parts of the two entitities that are not homologous.

- **Species1.anatomical-entity1 *union* species2.anatomical-entity2**

In the current implementation, this query returns what anatomical-entity1 maps to and what anatomical-entity2 maps to. The implementation of this query type may need to be modified as the application evolves.

Comparing structures

The following approach is taken when comparing two structures to find similarities or differences:

1. The mapping results check whether the structures map to each other. If they do, proceed to the next step. If they do not, inform the user what each structure does map to.

2. The attribute and relationship results compare the attributes and relationships of the subject and object. If the user is looking for similarities, the shared attributes and relationships are returned. If the user is looking for differences, attributes and relationships that only the subject has and that only the object has are displayed.

3. The attribute value and relationship value results check whether the values for the attributes and relationships that the structures share are the same. For example, if both the subject and the object have the Boolean attribute *has mass* compare the value of *has mass* i.e. true or false. If the user is looking for similarities, display the values that both structures share. If the user is looking for differences, display what value the subject has that the object does not and what value the object has that the subject does not.

4. The user has the option of making queries with *unknown* structures by selecting *unknown* for either the subject or the object entity. A query that involves one known and one unknown entity returns what the known entity maps to. A query involving two unknowns returns an error message. For a query involving an unknown, the query type chosen is not considered.

Level specification

It has been found that similarities and differences can occur at all levels between two graphs [1]; performing a query down the part-of hierarchy of both species would show

the user all levels of similarities and differences of the structures of interest. Therefore, we implement an option for the user to select the number of levels of the part-of hierarchy that the query should be performed on. If the level is 0, only the subject and the object are compared and the results displayed. If the level is 1, the subject and object are compared with the specified relationship AND their parts are also compared using the same query type.

The number of levels of the part hierarchy that the query can be performed on is 0 to 4, inclusive. We limit the number of levels for performance reasons. If a number greater than 4 is entered, a message is displayed informing the user that it is not possible to search more than 4 levels deep, and the search defaults to 4 levels. Currently, this option is not implemented for queries with *unknown* structures, and it is also not implemented for the query type *union.*

The Graphical User Interface (GUI)

One of the big tasks of this project was designing and implementing the GUI for the application. We started off with a very basic prototype of the GUI and built up to a working version that supports the current needs of the application. Working on this aspect of the application gave us the opportunity to learn about developing user interfaces and working with the Java Swing classes.

The GUI is designed in three main sections arranged vertically:

1. The top section is the “mapping direction” panel
2. The middle section contains the necessary components to select the structures and query type and execute the query
3. The bottom section shows the results of the query

Changing the mapping direction

The user can specify which species should be the subject of the query and which species should be the object by selecting the appropriate species from the drop-down menu at the top of the application window.

The anatomical hierarchy display in the middle section

The anatomical classes are fetched from the knowledge base and displayed in a tree form for the user to search or browse through, and select the subject and object of the queries. Even though all the Protégé classes are displayed, not every one of them may be selected by the user. This is because the knowledge base is implemented in Protégé as a subclass hierarchy, and there are some Protégé classes that do not represent actual anatomical structures of a particular species but are abstractions that can be used to model more than one species. For example, the class "Organ" is an abstract class, and subclasses of it such as "Prostate (human)" are concrete classes which do not necessarily belong to the same species. It is useful to think of concrete classes as instances that exist in the real world. Therefore, even though we display the whole anatomical hierarchy, including the abstract classes, the user is only allowed to select concrete classes to perform queries on.

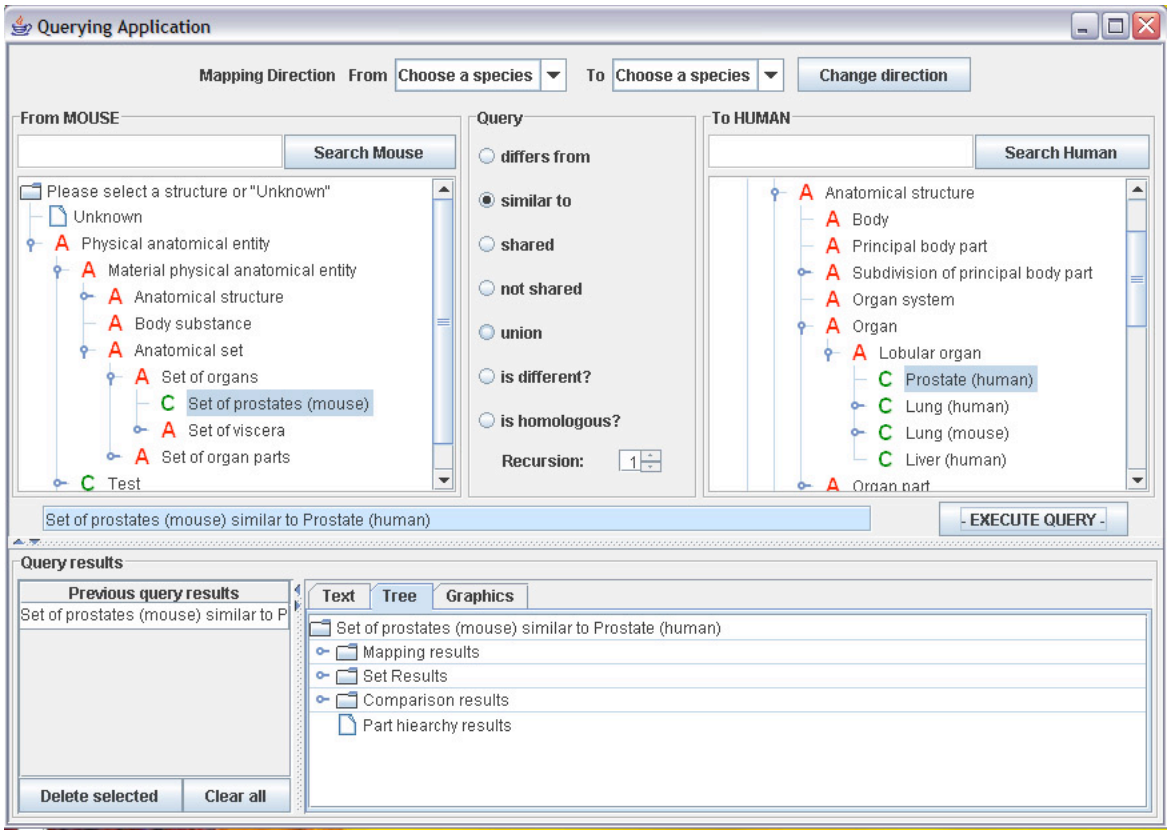

Figure 1: The GUI has three main sections: the top "Mapping direction" panel, the middle section containing the anatomical hierarchies and the bottom results section.

Results

The query results panel contains a previous queries pane and the actual results display.

Previous queries are saved and the results can be seen by clicking on the corresponding row in the "Previous queries" table. If the query is executed a second time by selecting its row, it is not added to the table again. Each query has a single entry in the table, and even if it is executed again, its place in the table does not change. As queries are executed, they are added to the bottom of the table.

There are also two buttons, a *Delete Selected* button and a *Clear all* button which are used to remove previous queries. Clicking the *Delete Selected* button removes the row of the table selected at that time, and clicking the *Clear all* button removes all entries from the table.

The display panel has three tabs - text, tree and graphics. The text tab displays results in a text format, the tree tab displays the results in a tree hierarchy and the graphics tab shows the results as a picture.

- *Graphics tab* - this feature is not yet implemented.

- *Tree tab* - the tree display is useful when a user is interested in searching multiple levels of the part hierarchy, because the results are displayed in an intuitively hierarchical fashion.
- *Text tab* - the text results are useful if the user would like to copy-and-paste the results in a different document. The text tab also displays all the error messages, and so it is the default tab. It is possible to switch between tabs by clicking on the tabs or by pressing Alt-1, Alt-2 or Alt-3, where:
  1 = text tab
  2 = tree tab
  3 = graphics tab

Further lessons learned

Knowledge base implementation

Because the development of the knowledge base is in its earliest stages, we found that it did not contain enough anatomical classes for us to perform tests on the implementation of the queries. We partially solved this problem by including very simple test classes in the knowledge base, which may be removed when they are no longer of use.

However, the fact that the knowledge base was not complete gave us the chance to learn the details of the Protégé knowledge base implementation, and we learned to enter some anatomical information ourselves. Furthermore, we had to chance to meet and interact with people outside the Computer Science department who work in the area of knowledge representation and biomedical informatics.

Design and implementation problems

While working on the implementation of the queries, we discovered previously unforeseen problems with the design. We learned that one of the best ways to test the design of a system is to try and implement a prototype, so that problems are detected early and the design can iteratively be improved throughout the lifecycle of the project.

Conclusions

This paper describes the design and implementation of a comparative anatomy information system, as well as briefly explaining the reasons why such a system would be useful. The first sections cover related concepts and the theory of the design of the system, which Tejinder and I had to learn before we could begin the implementation. The last sections describe how we implemented the system and the work we have done on the project during the summer of 2005.

We have created an initial prototype, but there is more work to do before the application becomes a truly functional comparative anatomy system. Working on this project has given Tejinder and I, both undergraduate students, an idea of what conducting research in

computer science means, and what it is like to work on developing a large-scale application.

References

1. Travillian, RS, Rosse C, Shapiro LG. An Approach to the Anatomical Correlation of Species through the Foundational Model of Anatomy. AMIA Annu Symp Proc. 2003: 669-73.

2. Mejino JLV, Rosse C. Symbolic modeling of structural relationships in the Foundational Model of Anatomy. In *Proceedings, First International Workshop on Formal Biomedical Knowledge Representation (KR-MED 2004)*, Whistler Mountain, Canada. To appear.

3. Travillian RS, Gennari JH, Shapiro LG. Of Mice and Men: Design of a Comparative Anatomy Information System.

4. Stanford Medical Informatics Group. The Protégé Ontology Editor and Knowledge Acquisition System. < http://protege.stanford.edu/index.html>

5. MIT CogNet. Frame-Based Systems. The MIT Press. 2003. <http://cognet.mit.edu/library/erefs/mitecs/nebel.html>

6. Travillian RS. General Examination Paper. February 25, 2005.