

DEVELOPMENT OF DYNAMIC DATABASE STRUCTURES
USING OWL ONTOLOGIES

by

Kurt R. Stiehl

Submitted to the Department of Mechanical Engineering in Partial
Fulfillment of the Requirements for the
Degree of

Bachelor of Science

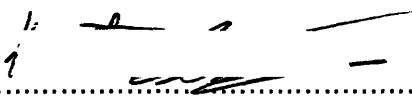
at the

Massachusetts Institute of Technology

June 2007

© 2007 Kurt Stiehl
All rights reserved

The author hereby grants to MIT permission to reproduce and to
distribute publicly paper and electronic copies of this thesis document in whole or in part
in any medium now known or hereafter created.

Signature of Author.....

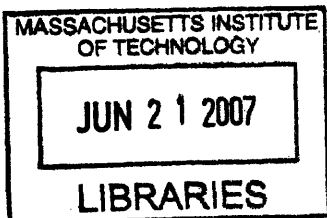
Department of Mechanical Engineering
February 28, 2007

Certified by.....

C. Forbes Dewey
Professor of Mechanical Engineering and Bioengineering
Thesis Supervisor

Accepted by

John H. Lienhard
Chairman, Undergraduate Thesis Committee



ARCHIVES

DEVELOPMENT OF DYNAMIC DATABASE STRUCTURES USING OWL ONTOLOGIES

by

Kurt R. Stiehl

Submitted to the Department of Mechanical Engineering
on February 28, 2007 in partial fulfillment of the requirements
of the Degree of Bachelor of Science in Mechanical Engineering

ABSTRACT

Standard query language database systems cannot provide the level of flexibility and functionality needed to store all of the data that a modern biologist requires. OWL ontologies provide much more information about how fields relate to each other including adding annotations with each field, but do not allow for the storage of mass amounts of data. By integrating OWL ontologies with modern database systems, scientists will be able to build ontology-based databases that combine the ease of development of ontologies with the power of the modern database. In addition, other scientists can now search ontology-based databases and know that even though the databases are different, the types of data are the same because they are based on a consistent ontology. The use of ontologies in building modern databases will create a new environment where users can customize databases to account for new methods in biology, while at the same time maintaining the ability for others to effectively search the modified databases.

Thesis Supervisor: C. Forbes Dewey

Title: Professor of Mechanical Engineering and Bioengineering

1. Introduction

The amount of information generated by a modern biologist is growing rapidly. Biologists are beginning to generate terabytes of data using modern imaging methods as well as complex analysis programs. The modern SQL database has failed to keep up with the rate of change of biology. Information is being lost because it is getting misplaced in databases, or it is not getting stored at all. To combat this problem, new database technology must be developed to allow for the growing need for data storage and organization. This paper shows how OWL ontologies developed by the W3C consortium will provide the solution to the current deficiencies in modern databases.

OWL ontologies are documents that store the relationship between different objects as well as storing the object description itself. SQL databases on the other hand will only store objects and their attributes. There is no way to really tell how database objects are linked to one another without independently discovering the functionality of a specific table. However, by combining SQL databases and OWL ontologies, it is possible for database designers to build structures that not only have objects, but have descriptive comments to define what those objects are and how they relate to one another. Because the ontologies use the descriptive power of the Rich Data Format (RDF) proposed by the W3C, the descriptions are parsable using machines rather than requiring human comparisons.

This paper describes the process of creating an ontology-based database and shows that a laboratory can successfully build a database system without the headaches of current database technology. Databases are no longer places to store static entries in a few tables, but can become dynamic resources where entries can be made and entire tables can be created as needed.

2. Theory

The design of dynamic databases using ontologies requires two parts. First the ontology must be able to create a database, as shown in Figure 1. Ontologies must be able to fully describe a relational database, so that there is no functionality compromised when using ontologies instead of making the database directly. Secondly, and more importantly, ontologies must be able to update and change the database on the fly, while maintaining other users' ability to search and navigate the database. Only then will the database become truly dynamic in nature.

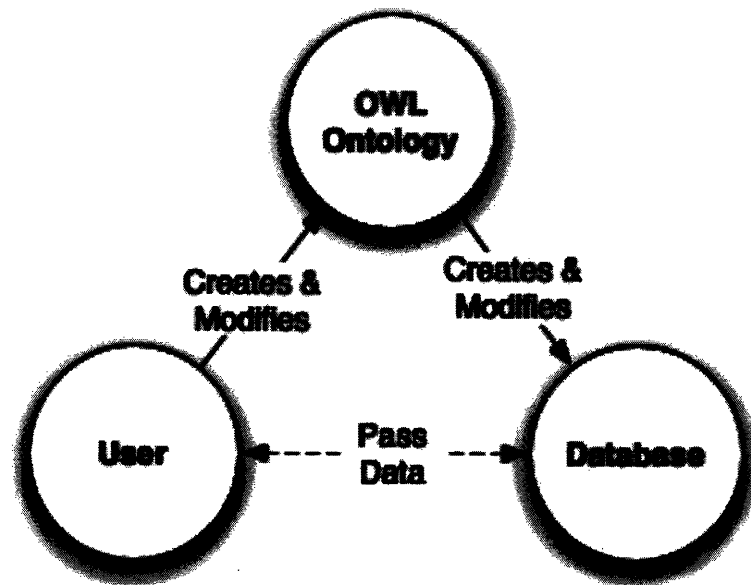


Figure 1 – The structure of the database is controlled by the OWL ontology. The only things exchanged between the database and the user are the data

In order to understand how the ontologies will create dynamic databases, it is first important to show how an ontology can map itself onto a SQL database structure. Ontologies are made up of a class structures to organize data into different classes. Each of these classes can have sub-classes and super-classes, thus creating an inheritance model. Each of these classes can very easily represent tables in a database. The tables in the database do not need to know the table that they inherit their properties from, so the connection is quite simple.

Ontology classes can also contain a number of properties. These properties can either hold data or links to items in other properties. In a relational database, properties map to columns in tables. Each class will have properties and each table will have columns. If an ontology property is a data-type property, then the connection is very simple. All data in that column will have the properties of the data-type property in the ontology. On the other hand, if the property is meant to hold objects from other classes, the connection is less direct. The database will be set up to store the key from the object in the matching table. For example, if a table of microscopes has a link to another table of lenses, then the link will be created by the key for the lens table. While the map is not as clean as the data-type properties, the link enables for a much more robust method to record items in a database than is currently available.

There are some features of ontologies that will not map to the database. The hierarchy of super-classes and sub-classes can never be written directly into a relational database, and can only be inferred by the inheritance of properties. In addition, the ontologies contain annotations for each of the properties and classes that do not transfer into a database structure. But both of these features are not necessarily lost in the user experience. A user can work directly with the database and look at only what the tables hold. But to really maintain the power behind the ontology based database, a user must look through the ontology as well as the database itself. There, the user will not only find the table format and the data, but can read how one table relates to others and find information about the tables that a relational database was never meant to hold. A perfect example is the units of a field. Ontologies will enable a field to have units stored with it, so a user will always know what the units are that apply to a field. In a relational database, the same user would only see the data and would never be able to infer in what units the data

were recorded. Thus when a user enters data into the database, that user must look at the annotations of each column to verify that they are entering the right data.

While ontologies can hold useful information about a relational database table, there are a few pieces of information that the ontology is missing. Ontologies do not contain the actual commands to create columns other than the property names themselves. The solution to this problem comes in two forms. First, properties do contain a range field, even though they do not contain a specific type field. Thus a hash map must be created to map from ontology range to database column type. For example, wherever the link to the standard integer of the W3C consortium is used, the integer form will be applied to the column in the relational database. These mappings can then be customized to create specialized types and mappings as the user sees fit. The other pieces of information will come from custom annotations in the database, as shown in Figure 2. Wherever a user wants to set the size of a field that is separate from the default size, they can add an annotation that will store this size. Then, in the conversion program, all the user has to do is point out which annotations are the size annotations, and the converter will read them into the relational database tables accordingly. If a field is never supposed to be null, an annotation can be added that will tell the converter to make all fields with this annotation non-null.

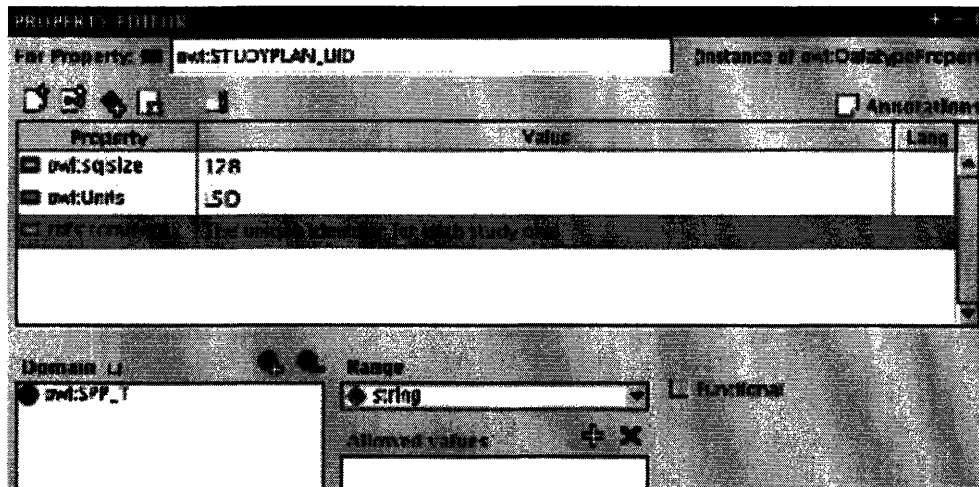


Figure 2 – This is an image from the Protégé property editor showing the annotations that can be added to complete the translation from OWL property to SQL column

Ontologies can easily create a database, but the crucial part of using ontologies is their ability to make the ontology dynamic. Unlike fields in a database, each table and field in an ontology represents a unique link. For example, whenever the property “Molarity” shows up as a column in a database, the meaning is always the molarity of the specified solution in that entry. There are no duplicates inside the database. The power of ontologies extends farther when you can use the synonym functionality. Not only is it possible to say that the property is the same in the specific database, but it is possible to say that the property is the same as the property in another database. Thus, “Molarity” means the same thing between databases and can be used accordingly. The result is that users can now perform searches over multiple databases, using terms that are defined over a large quantity of databases.

Currently, the solution to creating a global database system is to standardize all of the databases, so they look exactly the same, making it seem that ontologies are unnecessary. The problem with standardized databases is that users cannot alter the standard to meet their needs. But with ontologies, all of this is changed. Users can come in and alter the ontology as they see fit. They can change the units of “Molarity” to something that is more amenable to their lab (e.g.

nano-molar rather than micro-molar.) They can add and remove tables. The result is that they do not link to others' definitions, but instead are creating their own definitions others' can now link to. If one scientist adds a table for a new class of microscopes, other scientists can mimic those additions and say that their addition has the same form of the first scientists. Then whole new sets of tables are searchable without ever having to meet and confer that the standard is correct. In addition, the user has the ability to include annotations with the tables' new and changed properties, such that other users can better understand the changes without having to directly contact the first user. Scientists can alter their databases as they see fit, and other scientists can search these databases for information knowing that as long as the term for which they are searching is linked to the term they are considering, the values should match accordingly. Thus a fully dynamic standard can be created that consists of the standard derived by scientists modifying the database as needed.

3. Methods

Using the theory developed to integrate ontologies into relational databases, tools were assembled to perform all of the required functions. The first operation required by the user is the ability to edit and work with the ontology. Although, it is possible to write ontologies by hand in a text editor, the experiments performed utilized Protégé from Stanford as the ontology builder. Protégé offers the user the ability to easily visualize and work with the ontology, from class creation to property editing. Protégé also offers the ability to localize the ontology quickly and includes a set of standard data-type property ranges. Therefore it is easy to develop a map of a database in an OWL ontology format.

While Protégé already exists to edit ontologies, there is no tool currently available to convert them to relational databases. Thus, OWLdb was created to perform the missing task. OWLdb is

built entirely using java, like Protégé, allowing it to work on any platform. OWLdb will take a standard .owl file and will allow you to create and update SQL databases. Currently, OWLdb only supports MYSQL databases, but other major databases can be added by simply including their various JDBC plug-ins. A user will not only be platform independent, but will also be database system independent. As long as the database accepts common SQL commands, they will be able to create their database from the ontology.

OWLdb also provides functionality to maintain an ontology-based database. A user can compare an ontology to a database, and see what changes they have made to their ontology, as shown in Figure 3. Then, once these changes have been documented, a user can upload their new versions of the ontology. The result will be that only the affected tables and columns will be changed, keeping all of the other data. This method is opposed to having to reinstall the database every time. Users can add classes as needed, and all of the older classes with their respective data will be maintained.

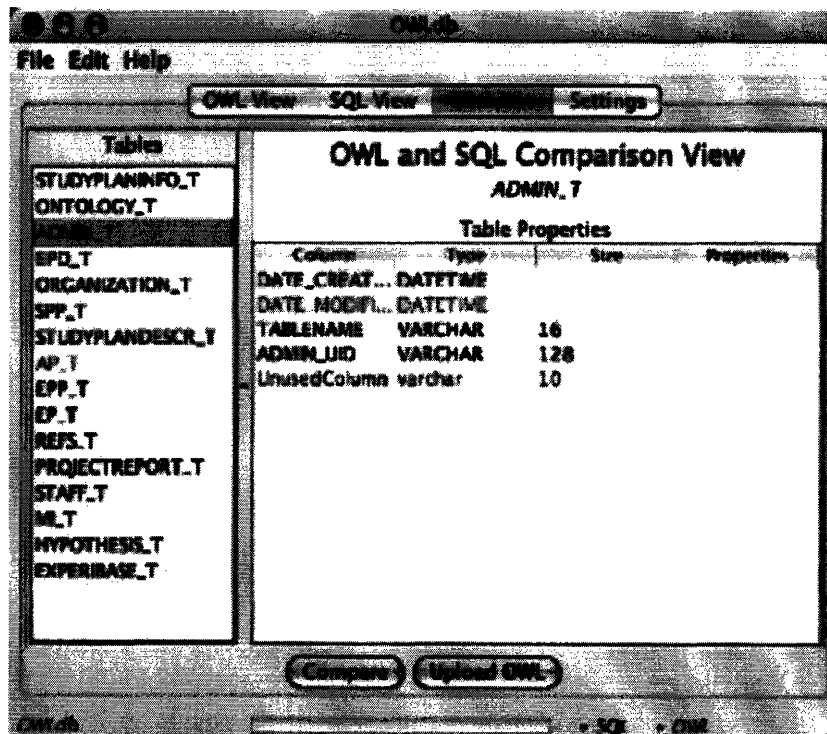


Figure 3 – Shows the *Compare* tab of OWLdb where a user can look at the differences between an OWL file and an uploaded database, and then can upload the OWL file to make the database match it.

Finally, OWLdb allows for the user to control how the ontology is interpreted and to view how the ontology will be transformed into tables. OWLdb provides a full interactive interface to look at not only the ontology version of the table as shown in Figure 4, but its converted SQL view. Users can test ontologies that they create and see if they create the desired tables before they ever load it into a live database. In addition, users will be able to edit the mapping of the ontology properties. If they choose to add a data type, or add an annotation for field size, OWLdb will be able to accept their changes and interpret appropriately. A more complete description of how OWLdb is built and organized is provided in Appendix A.

The user will be able to easily connect one ontology to another using LSIDs and a simple web hosting service. By hosting an ontology online, each definition in that ontology becomes a unique entity that can be found by its LSID. The LSID for an ontology consists of a link to the

ontology, and then a link to the individual objects inside of the ontology. Thus, each class, property, and annotation has its own address. Users can utilize the equivalent class functionality in the ontology standard to link their ontologies to other ontologies. Other users can then verify that two objects are the same and can be used in the same way in a search.

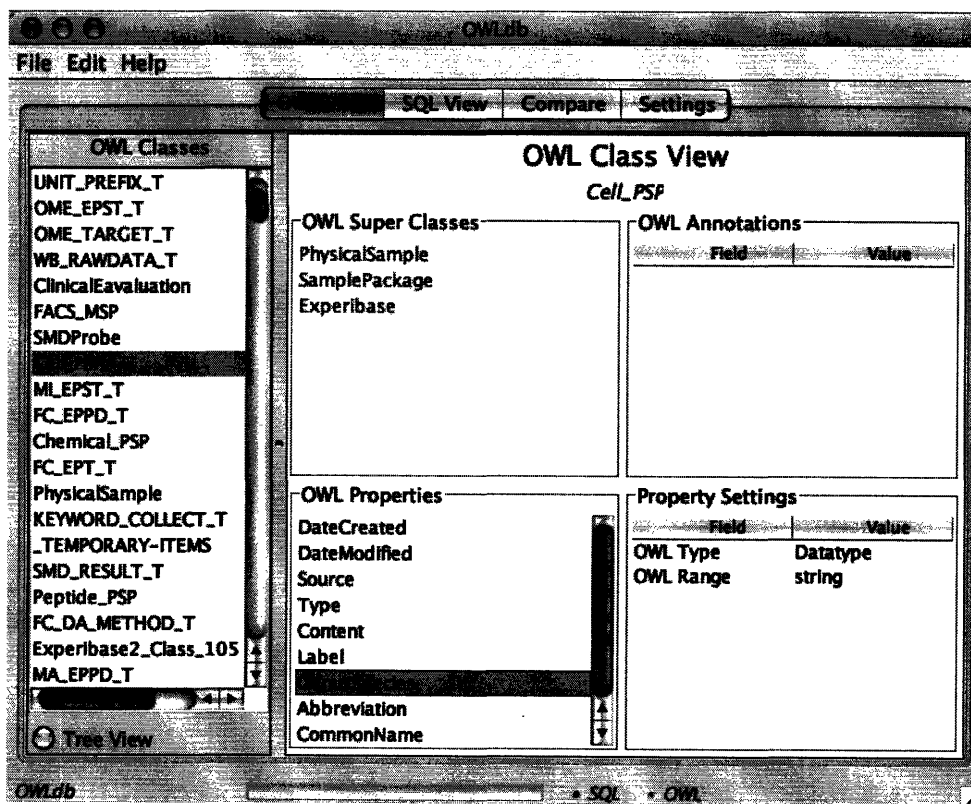


Figure 4 – The OWL viewer in OWLdb. The viewer allows you to look through classes and compare them to how the translated SQL versions of the classes look when you click the *SQL View* tab.

The last tool required is the database itself with OWLdb and Protégé. Users can choose any of the modern SQL databases and operating systems to run their data storage. At this point, there are no tools written to create pages to input and view the data, but a standard SQL system should allow for data input into the tables.

4. Results

Using all of the tools above, databases were built and modified to see how the system worked. There were two groups of people that were targeted. First, the perspective of the database designer was analyzed. It was crucial to figure out whether or not the designers would actually benefit from a system that may seem more complicated than what they already have. Secondly, it was important to look at the database from the perspective of a user who wants to simply input and search data. Is integrating OWL ontologies a worthwhile decision to make a database more usable?

4.1 *Creating and Modifying a Dynamic Database*

From the perspective of the database designer, making a database with OWL ontologies and the OWLdb converter is easier than most of the current alternatives. The designer can create tables and sub-tables using the OWLdb interface and then use them to create or modify the database. They do not have to worry about writing SQL commands, and they can gain from the class system that OWL offers. The problem of using ontologies for a designer is where to start to make sure the database is designed carefully.

While creating databases is easy, designing them is still slightly tricky. The standard must be very well thought out so that users are not constantly fixing bugs and then making databases dissimilar and hard to search. From the tests run with building different ontologies to make databases, the better the first version of the database, the more consistent the database would remain from iteration to iteration. For example, if a database is started and it only has a table for experiments, the database is essentially unusable on its own because users will have to create sub-tables for different experiments and operations. Then the only search functions that can be performed across a wide array of databases are those searches with the top-level experiment parameters, meaning that no specific experiment will be able to be queried. On the other hand, if

a standard database holds a number of pre-created experiments, then users will be able to start taking data on those experiments and get more information out of their basic searches without having to re-work their databases right away.

The conclusion from this finding is that users will not build many ontology structures all the way from scratch. Instead, different groups with different functions will set up base ontologies. There could be an ontology for bio-imaging as well as an ontology for DNA sequencing. Databases will start from a well defined mold instead of completely from scratch. While the task of making these initial ontologies may seem daunting, groups have already built standardized databases. Experibase from the Dewey lab is an excellent example of a complete database that creates all of the tables needed for bio-imaging. There are also other databases available that simply have to be converted into the OWL format so that they can be then be modified and built upon.

The real power of the system of using ontologies comes in the design of database modifications. There are two types of modifications that can be performed. The first is a modification that simply extends the current capability of a database. An example of this modification is a user who wants to make a table for a specific type of microscopes, so they create a subclass of microscopes. The second is a modification that adds a completely new set of features to the database, allowing the user to store an entirely new type of data.

The ability for a designer to extend a database by simply creating a subclass of a table is extremely powerful. First and foremost, the designer will inherit all of the properties of the base table. Any of the annotations on the super-class will be passed on. All of the properties will have the same sizes and types. All of the information stored on how to build the original table is saved. Then, all the designer has to do is add the fields that are not available in a normal

iteration of that table. To continue with the microscope example, if the original microscope does not have a property for a contrast setting, then the user can make a new class of these microscopes in one click, and then just add the new property titled *Contrast*. The other advantage to using ontologies is that other users looking at the database can still see the new microscopes in their searches even though they are in a new table. In a traditional database, the user will only be able to search for data table by table, because there is no way of guaranteeing that the columns in one microscope table are the same as the columns in another microscope table. With ontologies, the user knows that the columns are the same and can simply look through not only the microscope table, but every table sub-classed from it

. Database designers may start with a standard, but there is no real restriction to keep it absolutely consistent with other databases.

The other advantage of using ontologies is that designers can build whole new sections of an ontology and then send them to other designers to quickly drop them into their database design. For example, a designer can build a whole set of tables devoted to the operation of a confocal microscope. The user can then separate the XML code for these tables from the XML code of the other tables by simply cutting around the tags in the OWL document. Then, any new user can paste the new tables into their database by adding the XML code to their ontology. Ontologies do not read the files in any particular order, so the user could technically add the tables anywhere inside the ontology as long as it is not inside the defining statement of any other objects. The result is that users can make plug-ins for all of their different lab techniques, and then post the small snippets of XML code for other users to add. Instrument manufacturers can easily support the capabilities of their instruments by adding native tables to the existing databases. Users can now create a new standard by simply adopting new sections of the

ontology as they see fit. Ontologies allow designers to feel free from dated technology and organization and can design and share entire new sections without drastic search implications as seen in current static databases. Labs will be able to reach a state such that whenever a new technique is performed or a new piece of equipment is acquired, the corresponding OWL tables will be added to the master database.

Overall, the use of ontologies in database design allows for a much faster turn around and a much more relaxed approach when trying to design a standard that encompasses every point of a field. What used to take years to build the perfectly structured database, can now be shortened to months as database standards can grow. This is especially pertinent to modern biology where more often than not the science is out-pacing the rate IT can grow to accommodate it. OWL ontologies will make the use of database storage a real possibility for a fast paced laboratory. Thus, the major advantages of relational databases that have been developed over the past 20 years can be captured and exploited in a dynamic environment.

4.2 Searching and Entering Data into a Dynamic Database

Noticeable improvements in the ability for users to search and work with were also found when ontologies were integrated. One major source of improvement comes in the simple form of documentation. A normal database comes with next to no inherent documentation. Some database systems come with their own methods for adding information to databases, but there is no standard across all systems. Ontologies correct for this by providing a cross platform, cross server interface where a designer can include all of the information a user would need to correctly and accurately input data into a system. One example already cited earlier is the use of units throughout different properties. Each numerical property can come with its own unit, as well as numerical exponent and other desired parameters. In addition, non-numerical inputs can

also begin to be documented. When a column for *Description* pops up in a database, the user may not know what they are supposed to describe. But, if *Description* has a comment attached to it saying that it is supposed to be used to give a written observation of any abnormalities in a sample, the user then can correctly fill in the information. A standard SQL database cannot store all of that information, but using ontologies, this shortcoming can be compensated for by adding the fields easily and non-destructively as needed. Thus, whenever data are accessed or deposited into the database, the user needs to look at the ontology as well as the database schema.

One specific area that has not been pursued is the field of data input into the ontology based database. In some ways, it would seem that data input has only gotten more complicated because there is an extra layer of abstraction on top of the database. Based on the limited work done with test databases, ontologies create a more tractable environment. Ontologies allow for a developer to quickly create a custom application that will build more complete pages without ever having to look at the specific schema of the database. Custom input pages can easily be generated from the ontology itself, because the ontology holds so much more information than the average database table about what is supposed to be entered. Using ontologies, it is now possible to develop methods such that the instant a table is created, it's corresponding input page and display page are also created, as opposed to making custom pages for each table.

Lastly, and most importantly, users gain the ability to search multiple databases easily and freely. Ontologies allow designers to create dynamic databases that can be based on standards instead of building a custom database that is only fit for the application of the lab. With the implementation of these new standards, users can create queries that extend over a large range of servers, and while some of the servers may have modified the tables slightly and may no longer

contain that specific data, other servers will hold true to the standard and can be searchable quickly and easily. In addition, it is easy for small clusters of people to synchronize their databases together. A database does not need to be a centralized entity, but can operate more like a document for each user, allowing other users to come in and search their data when they plug into the network. The use of ontologies in database management allows for a whole new level of computation to be performed on large amounts of data, while still maintaining the ability for customization to specific applications.

5. Conclusions

The use of ontologies in a modern database system allows for scientists to create and modify their own database, while at the same time providing a platform for other scientists to search those databases with better knowledge of the kinds of data they are reading. Problems like incorrect unit conversions and abstract columns can now be eliminated. The modern SQL database can start to provide the functionality and the customizability of an XML document, while building in the power and search-ability of a relational database, all through the overlay of OWL ontologies.

Acknowledgments

I would also like to thank the individuals in the Dewey lab, including Shiva Ayyadurai, for helping to work through all of the ideas that have been generated.

Appendix A – The Design Of OWLdb

OWLdb represents the major source of innovation in the work that was performed in this project. OWLdb has enabled the user to be able to convert their ontologies into full database schemas without ever having to write a single line of SQL code. In addition, OWLdb has created a process that allows for a user to create a database entirely using GUI's and a

WYSIWYG interface instead of having to go through and write out each table with each respective column, as shown in Figure 5. The major objectives for OWLdb were to first make the program easy to use. Without a simple and straightforward program, the whole purpose of OWLdb would be defeated. Second, OWLdb has to be multi-system. The solution to this problem was simple by just using the Java language from Sun Microsystems. OWLdb should be able to run in any environment that supports Java. Lastly, OWLdb has to be flexible. Not so flexible, that the interface is no longer easy to use, but flexible enough that advanced users can customize the conversion process as well as how they move their ontologies into databases. OWLdb provides the key link to obtaining a fast and easy database development process. Through the rest of this appendix, the function and underlying structure of OWLdb will be described.

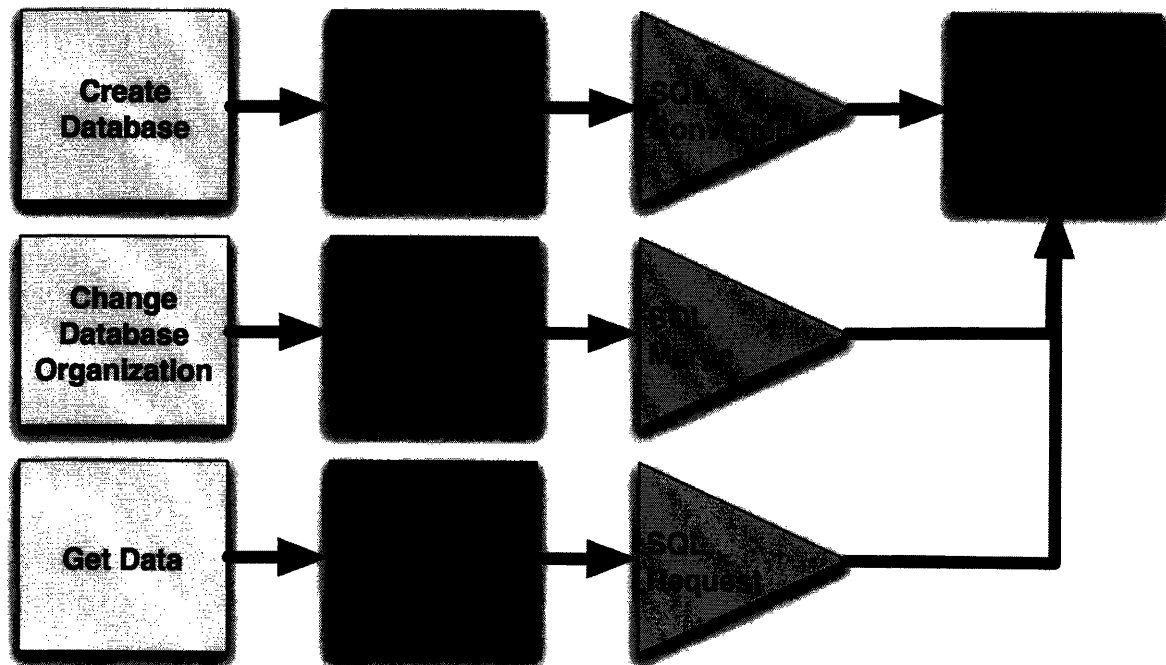


Figure 5 – The figure shows the user environment on the left featuring tasks that a user would want to perform when creating a database. These paths then move through the ontology stage and into OWLdb as the triangles, where they will finally reach the actual database.

The structure of OWLdb is based on four classes that run each of the functions of the interface, as shown in Figure 6. The first action that OWLdb must perform is to take an owl ontology and create objects that can be read and interpreted by the rest of OWLdb. These objects are called owlTables. In order to create an owl object, the software must run through and find all of the super classes of the designated class. Then the process of making an owlTable begins. First the name of the table is inserted as the object name. Then, each of the annotations related to the class itself are added. These annotations will be displayed next to the class name, allowing the user to gain more information about each class individually. Next all of the properties of the class must be included. To do this, the software must find each of the associated properties for the class, and all super classes, as properties are inherited. The properties will define all of the parameters that each class has. Lastly, the annotations for each property must be included. The software must loop back around and check each of the individual properties for relevant annotations. The annotations will be used to show the user information about each property as well as to convert the property into a SQL column.

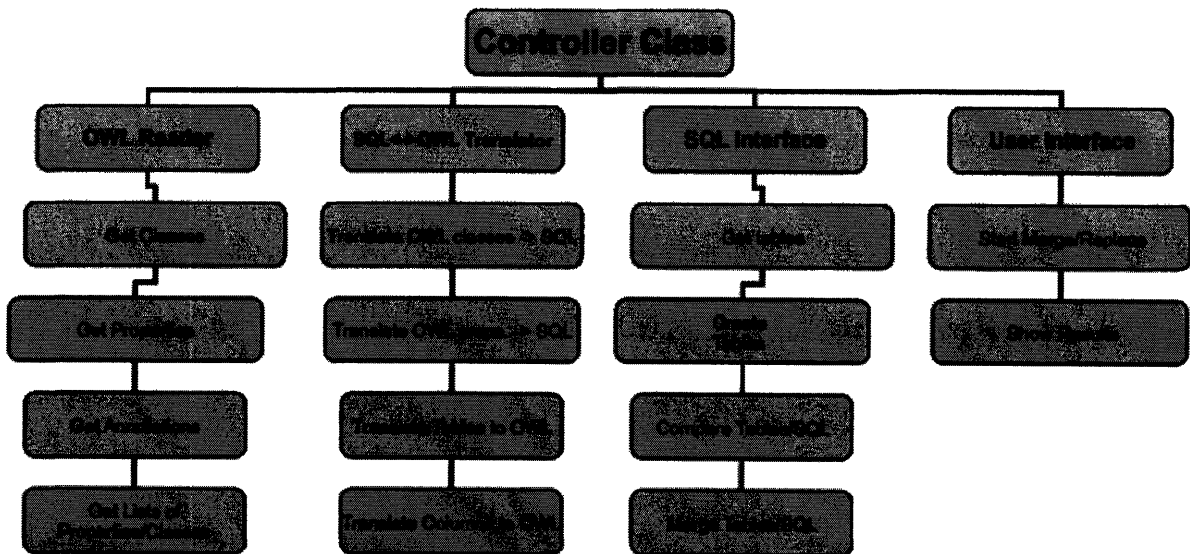


Figure 6 – The figure shows the overall layout of OWLdb, detailing the five major classes and their functionality.

The next step in the process is to convert the owl object into a SQL table object. The purpose of having two separate objects is that the conversion process can be controlled such that the user only performs the process at one instant, and then the results are saved, instead of having to convert the table every time the SQL properties are accessed. In addition, it enables the creation of SQL objects from the database itself. To convert to a SQL object from an OWL object, the name is transferred, and then each of the individual properties are converted into columns. In order to convert each of the individual properties, the size and type of the column must be determined. The column type comes from the range field of the property. The ranges are matched using a list of possible ranges and converting them into their corresponding SQL types as described in the theory section. The size of the field is derived from an annotation. The user sets the annotation, so that any annotation they choose can be the field that sets the size of the SQL column. The only requirement is that the annotation be of an integer type. The last step of the conversion process is to add in any annotations that have pertinent SQL qualifiers

(e.g. unique or key.) Each SQL object also includes two additional features. First, the object can output a SQL string to create the table that is represented by the object. It is then easy to build databases by simply collecting all of the strings from each of the objects. Lastly, each SQL object includes functionality to compare its columns to an input column. The SQL objects created by the database can then be easily compared to the SQL objects created by the ontology, and the underlying differences can be outlined.

The last step in the process is to connect to the relational databases themselves and enable the SQL table objects to create and modify tables. The SQL database manager class first takes care of connecting to the database. The class gathers the connection information from the user interface and then connects to the respective database. To connect to the database, the class pulls the correct JDBC plug-in from its library of plug-ins, and uses it to connect. Currently, only one plug-in is installed, but it should be straightforward to integrate multiple plug-ins at a later point in time. Once the database is connected, the database manager provides three options. First, it can download all of the tables from the database and create SQL table objects out of them so that they can be displayed to the user. The manager can compare each of the downloaded objects to the list of SQL table objects that are derived from the ontology and include colors to designate whether an ontology object is creating, modifying, or deleting an object from the database schema. Lastly, the database manager can modify the database by making select queries that force the database to resemble the loaded ontology. By using only the required queries, the rest of the data in the database is preserved, and as long as users are only creating objects, no data are lost. The ability to merge the ontology is the key behind OWLdb's functionality, and by using all of the described classes; the process is as clean and clear as possible.

The GUI is the last class required to make OWLdb work. The GUI provides the user with the ability to view each of the Owl objects as well as each of the SQL objects individually. The user can then browse through the database and compare how their ontology is being converted into a table. The GUI then provides a pane to upload it to the database. The pane will either show the results when the user simply compares the current ontology to the database, or when the user merges the ontology and actually makes changes to the database. The ability to view differences in color enables a fast method to spot differences and judge whether the changes are required or whether, the ontology should be modified to match the database in order to not lose valuable data. Lastly, there is a pane that allows the user to set up all of the configuration properties of OWLdb including the source for the OWL file and the SQL database connection properties. By providing a clean and simple interface, the GUI provides the final element needed to make a smooth transition from OWL ontology to SQL database.

OWLdb can make the transition from OWL to SQL clean and complete by providing all of the necessary operations, neatly organized into four different classes. In the future, it will be possible to add additional features by connecting in new classes and operations, as well as building on the original classes. OWLdb is a flexible and expandable platform that brings two worlds together by providing a straight forward path for conversion.

References

Dewey, C.F. et al. (2004) A Unique Opportunity in Biological Information Object Standards. *W3C Conference Paper*, 10/27/2004. http://www.w3.org/2004/10/swls/forbes_dewey.ppt

Knublauch, H. Dameron, O. and Musen. M.A. (2004) Weaving the Biomedical Semantic Web with the Protégé OWL Plugin. *First International Workshop on Formal Biomedical Knowledge Representation, Whistler, Canada, 2004*.

Kohler, J. and Shulze-Kramer, S. (2002) The semantic metadatabase (SEMEDA): ontology based integration of federated molecular biological data sources. *In Silico Biology*, 2(3), 219-231.

Lam, YK et al. (2006) Using web ontology language to integrate heterogeneous databases in the neurosciences. *AMIA Annual Symposium Proceedings*, 464-468.

Lee, T.J. et al. (2006) BioWarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7, 170.

Lee, J. and Goodwin, R. (2006) Ontology management for large-scale enterprise systems. *Electronic Commerce Research and Applications*, 5(1), 1.

Michael, H et al. (2005) Deriving an ontology for human gene expression sources from the CYTOMER database on human organs and cell types. *In Silico Biology*, 5(1), 61-66.

Ma, J. (2006) Building an Ontology-driven Database for Clinical Immune Research. *AMIA Annual Symposium Proceedings*, p1018.

McGuinness, van Harmelen. (2004) OWL Web Ontology Language Overview. *W3C Recommendation*.

Noy, N.F. and Musen., M.A. (2003) The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. *International Journal of Human-Computer Studies*.

Perez,C and Conrad, S. (2006) Database to Semantic Web Mapping using RDF Query Language. *Conceptual Modeling - ER 2006, 25th International Conference on Conceptual Modeling, Tucson, AZ, USA, November 2006*.