

HONORS THESIS PROPOSAL

Please complete the thesis proposal form, attach a thorough description of your project according to the set of guidelines appropriate to your project (research or creative/performative), and submit the proposal to the Honors Program office. The proposal is due no later than the end of the tenth week of Semester One.

Name: Kevin Shamm Student Number: 433374

Major(s): Computer Science

Advisor Name: EUGENE WALLINGFORD Department: COMPUTER SCIENCE

Thesis Title: Implementing a Natural Language to Structured Query Language Translator

Approval of Thesis Proposal:

Eugene Wallingford COMPUTER SCIENCE 2010/10/28
Thesis Advisor Department Date

College Reviewer Department Date

Result of College Review: Approved without revisions
 Approved with suggested revisions (see attached)
 Returned for revisions and resubmission* (see attached)

Honors Director Date

*If the designated College Reviewer chooses not to approve the thesis proposal, he or she must attach a paragraph or more of comments to guide the student's revision before a resubmission of the proposal for a second reading.

Title

Implementing a Natural Language to Structured Query Language Translator

Purpose

The purpose of this project is to design and implement a natural language to query language translator. This translator will connect a web page to a SQL database. The web page will accept questions in “natural language” or in this case United States English. Some natural language questions may need to be clarified or restated in order to match the information stored in the databases. The translator will then process the questions looking for data to be read and then pass them to a SQL database. The database looks up the answer to the question and returns it to the web page to be seen by the viewer. All of this will be done using new and advancing technologies to simplify and speed up the development process.

Literature Review

The first attempts at creating systems with natural language access to a database were mostly unsuccessful and very constrained. A system was created to query a military maintenance database for information about particular maintenance requests (Waltz, 1975). Questions had to follow a very specific format, like “Were/Was <maintenance action> performed/done on/for <specific plane>?” About twenty questions could be asked of this system, but each question format had to be hand coded before it was going to be used and the system was unable to recover with grace if it encountered a question it did not understand. Although the system used a constrained form of English which is more user-friendly than a structured programming language, the system was not very flexible. The author noted that this approach to “language parsing” is not really parsing because “[many] words are defined implicitly within each pattern in which they occur in the pre-stored network. For example, the

system does not require subject-verb number agreement” (Waltz, 1975, p. 871) However, the author did build artificial intelligence (AI) into other parts of the system. The system can infer the meaning of certain phrases and ambiguous terms like dates and ranges. This system did not perform well with new types of queries because each query was hard coded in the computer as was the data format.

A second attempt at creating “an intelligent interface that provides natural language access to a large body of data” was investigated by four researchers at the Stanford Research Institute, later known as SRI International (Hendrix, Sacerdoti, Sagalowicz, & Slocum, 1978, p. 105). These researchers clearly knew that the focus of their investigation was to build a system that could extract specific data from databases without employing the services of a technician. Avoiding a technician is important because most of his or her time is spent comprehending the request and creating the necessary extracts and linkages from knowledge of the system and data store. The researchers cited the needs of business executives and other decision makers for current, accurate information as the reason for creating a new system. This reason will return throughout history. Although the researchers wanted to build a cross-discipline system, again their system had a Department of Defense focus. This time, their system extracted data for Navy command and control users. This research paper described the architecture chosen for this problem. The researchers linked multiple computers and computer files to be able to extract data from certain locations within those files. Then, the system of record responded with the answer to the query, returning only the data requested by the query. Fortunately, this system worked well and its pieces were compartmentalized enough so the designers could replace one piece as better technology developed. In addition, Hendrix et al. praised this system because if an incomplete input was given, the system would infer that the user simply wanted an answer to the

previous question with a changed data set or changed data piece. First the user would enter “What is the length of the Constellation?” Then the user would input “Of the Nautilus?” The system would infer that the complete second question was “What is the length of the Nautilus?” (1978, p. 109) The system could also correct minor spelling errors. Unfortunately, the language used for data querying was “Datalanguage,” a language that is no longer the popular language for querying databases. The natural language component of this system also could only accept as input a subset of natural language.

A large part of the ability to recall data from a database is based in part on E.F. Codd’s research into databases. Codd wrote a series of papers in the 1970s about data storage and normalized data. One paper in particular included a discussion about relations (how complex three-or-more dimensional data should be stored in a two-dimensional table) and relational calculus (how the data can be used and aggregated to answer specific questions) (Codd, 1970). This paper also specified what operations all companies’ database implementations should include. All of this is important to the field of natural language processing and database querying because it specified a generalized form for querying that many database systems have chosen to implement. This allows new application developers to create a generalized program that will most likely work with all systems one will encounter. Many of the major database systems including IBM’s DB2 (Chamberlin, 1998) and Microsoft’s SQL Server (IBM Archives: Edgar F. Codd, 2003) cite Codd’s paper when discussing their implementations.

A number of advancements in understanding natural language have made developing the solution to this problem easier and made the solution itself more similar to the linguistic complexity of English. A system named the “User Specialty Languages system” was created in which queries were split by word and identified by parts of speech (Lehmann, 1978, p. 560).

This system was assumed that users of this system would be experts in their field, but not experts in the underlying database or data constructs. This system sent a user query to separate parsers and interpreters. The parser was taken from another project, but the interpreter was custom designed and developed to cover popular questions in three languages. Some concepts were added for the ease of translation and a few more concepts were added because of the specific nature of the system to be developed. This USL system was able to handle time inputs and use them for data comparison. The system also caught plural nouns and used both a formal language interpreter and a world view of the data, as defined as a bit of knowledge about what was around the system, such as what data is available, to create database queries. When the system was developed, it was the most complex and the least restrictive, but other systems would soon come to overtake it in terms of complexity.

A second great advancement in understanding natural language came when researchers worked toward the goal of understanding language in its current state rather than translating it into a set of rules. Again, the researchers built a natural language front end called a “Discourse Module” and connected it to a logic and data back end (de A. Barros & DeRoeck, 1994). The natural language front end is built on top of another front end called SQUIRREL. In this instance, the front end “takes the input sentence, producing syntactic and semantic representations, which it maps into First Order Logic” (de A. Barros & DeRoeck, 1994, p. 120). First order logic is a type of predicate logic where sentences can be represented as facts, objects, and relations. These sentences can either be true, false, or unknown (Schafer, 2010). In this case, the natural language front end was responsible for transforming queries into models understandable by the database. The back end uses the logic-formatted query and domain specific calculus along with Codd’s tuple-relational calculus to create a SQL query which is sent

to the database for a database answer. The front end addition to SQUIRREL was developed in 1994 seems to work well because the creators allow for removal of ambiguity in at least three stages of the data transformation process.

It can also be noted that significant research and feature compilation was done by a research team in the paper “Natural Language Interfaces to Databases – An Introduction” (Androutsopoulos, Ritchie, & Thanisch, 1995). This 50 page paper gave a great general overview of natural language interfaces to databases. The researchers continued to discuss advantages and disadvantages to building such a system and linguistic problems encountered by individuals building similar systems. One important distinction made by the authors is that the users of these systems often expect the systems will be intelligent or will possess some form of artificial intelligence. Unfortunately, this type of database lookup can often be performed through straight English to SQL translation. Many artificial intelligence functions can be performed on a query. Functions are unnecessary and the problem can be solved by simply writing an algorithm that requests additional information from the user if the query is unclear. Similarly to the SQUIRREL system, this system parses a query using English semantic grammar. Grammar categories such as noun, pronoun, verb, and adjective are pulled from the query. Parsing the query in this way will solve some of the primary problems in interpreting English speech.

A recent paper by researchers in Sweden built off of Codd’s calculus, current language parsing techniques, and implemented a design for a natural language interface for geographical data (Minock, Olofsson, & Naaslund, 2008). This paper described how relations between data elements are translated from words like “with” or “in” into union statements between tables in the database. When queries have union, intersection, and other set operations, they can easily be

placed over top of a database to get a response with certain limiting criteria. These researchers also focused on solving two common and complex problems: noun phrases with complex clauses and choosing the right data. Again, by implementing parsing trees, the developers were able to parse a complex clause into simpler yet still difficult parts of speech. The complex problem with choosing the right data was solved by creating a user interface in which users can choose what data they wish to recover. A click and drag interface was created to drill down and allow the users to select particular data values from tables. Users were also allowed to choose from a number of sample queries like “How many ___ are ___” where users of the system filled in the blanks. This made the system much easier to use, but the programming time to convert databases into webpages was complex. Another interesting note is that by this time, other approaches to parsing the geographical database in question had been discovered, developed, and documented in scholarly journals. The researchers found that their method of user interface string selection performed better in tests when compared to methods including string entry where the system filled in blanks based on information it had previously learned (machine learning).

A small number of inference engines are being developed to answer questions that would have been sent to natural language interfaces. One such inference engine is the Wolfram|Alpha computation engine. Wolfram is the company best known for its Mathematica software program for computing, visualizing, and allowing use of special math algorithms. Wolfram took the computing power of Mathematica and the knowledge of the internet and put it behind a familiar search engine interface. The engine can calculate the solution to a number of different types of equations including most that can be drawn on a graphing calculator. Because of the type of display used (computer monitor), the website also draws 2-D and 3-D graphs of any equation entered. However, the developers cite the computational power when applied to data as being

the feature that sets this engine apart. (Gray, 2009) Gray (2009) says that computation matters “Because computation is what turns generic information into specific answers.” A sample query of “undergraduate university of northern iowa iowa state university university of iowa” creates a table with the number of undergraduate students at each state university. Computations can be done for chemical, biological, geographical, dates and times, money and finance, socioeconomic and web page data sets. In short, this website will become every math and science student’s best friend.

A second computational engine that is being developed is set to battle contestants on Jeopardy later this year. A system named “Watson” is being developed at IBM (Thompson, 2010). This question-answering machine is going to be the smartest of its type in the world. Watson will do more than Google’s simple return of a list of websites where the answer is likely to be found. Watson has actually read and stored over 10 million sources and will return a Jeopardy-style question to a Jeopardy answer. IBM was also the company that built the machine that took on Chess champion Gary Kasparov. IBM is building this machine to run on one of its one million dollar super computers. The team to build the machine was given fifteen scientists and three to five years. In just three years they have developed a system that can buzz the buzzer and give an answer in the game of Jeopardy in just six to eight seconds. However, Watson only buzzes in when it feels confident in its answer. Watson builds confidence by solving the problem multiple ways, literally using multiple algorithms. When two or more responses come back with the same answer, the system builds confidence in that answer. These scientists wanted to build a computer that was not limited by the type of data it was to be an expert on. Instead of building a custom natural language interface to a database, IBM built a system that can understand knowledge from all disciplines. Watson has stored information from 10 million

articles and is not connected to the internet when it plays Jeopardy, much like the human contestants (Thompson, 2010). Surprisingly, the computer has read enough cultural sources to do well at Jeopardy. IBM hopes to eventually market this system to law firms that need to sift through piles of legal cases or medical teams that need more experienced and knowledgeable sources.

Central Themes to be Addressed

Businesses across the country are struggling to find ways to get meaning and use out of their historical data. A recent Burton Group report showed that 60% of responders state they do not have the ability to get good analytical data from their data stores and data warehouses (Santos, 2009). In addition, that same report stated business intelligence is critical to virtually all Fortune 500 organizations. Economic pressures and business customers wanting more from their IT investment are driving the desire for more data. However, the aging data stores in most companies limit the ability to get data to specialized and expensive software programs. A number of offerings from companies such as Business Objects and Microsoft exist, but only the largest companies have the budgets and manpower needed to implement these solutions.

Even if a company has the ability to purchase these products, they are often difficult to use. During my two internships, I heard stories from IT customers about the software products they use and the problems they had in making them work. It would be nice to use a program that answered a simple question like “How many contracts will renew in the next 7 days?” No knowledge of database structure or computer programming would be necessary to make this application work.

In order to make data more accessible to those who need it, I plan to build a natural language interface to a database incorporating database schema and sample queries in order to

answer simple natural language questions about the data stored within. I will include a good parser, utilizing my background in English and Spanish linguistics. I also hope to include the ability to extend and add additional database tables as needed. I do not plan to include database updating or machine learning for the database schema, because these two topics are too complex for a semester long research and development project. Database updating in itself must include data dependency checking that is not usually incorporated into database schema. For this reason, the data checking would need to be included in a sub procedure and would again need to be hand coded. Machine learning from database schema is too complex for most databases because databases only include a limited number of data structures and labels given to data may not always be accurate. Also, data fields in databases often are used for identifiers and data the fields were never meant to be used for. In addition, I would like to know whether or not the systems and simple languages described in the literature review can be recreated and improved using current technology. I hope that Java and SQL make the development of such a system easier and faster. In addition, I hope to explore the natural language languages like ResearchCyc, True Knowledge, and Prolog to see if they are useful in solving this problem. Also, because my degree in Computer Science will have an emphasis in Software Engineering, I plan to produce all of the typical artifacts during the course of this project. I will include these and my source code with my final reflection.

Methodology

As with most software development projects, a good deal of planning and securing resources will go into this project. I plan to follow the standard waterfall approach software development life cycle (SDLC). In this approach, developers work through sequential phases waiting to jump into development until they know what needs to be coded.

The first phase in the SDLC is the initiation phase. This is when a project is described, resources are approved, and the project description is also approved. Some businesses call this an Approval to Plan (ATP) or Approval to Initiate (ATI). This Honors Thesis Proposal shall serve as the ATI for this project. Once it is approved, I will take this document and attempt to secure any resources I would need from other entities at the University of Northern Iowa (UNI) and other institutions. I would love to build my project on top of actual UNI databases. Institutional Research gave initial approval to this approach for certain data sets. Now that I know more about the project, I will need to return to them and ask again for specific needs. I also may need to get approval from the Office of the Registrar. If these offices on campus, or others, wish to use the project when I am finished, I will need to identify them as stakeholders and get their input so that my solution will work within the already existing UNI systems or recently purchased systems. Some frameworks for making the problem easier are available at low or no cost to researchers and academic institutions that the average person would not have access to. If this is the case, I plan to work with my advisor to get the necessary faculty and institutional approval to use these frameworks. The benefit of using a framework for language parsing or language translation is that it will save me time and allow me to build other features with the time I saved. This should be completed by the time I return to school in January.

After initiation is complete, I will begin the Analysis or Requirement Gathering phase of the project. After reading literature, speaking with individuals on campus, and attempting to build a system, I will come across needs, desires, requests, and scope boundaries that limit or require pieces of the system to be developed in a particular way. These needs and requests for a particular system feature are better known as requirements. Depending on the final scope of the project and the time I have to build the system, I may create requirements documents for User

Requirements, High-level Business Requirements, Non-Functional Requirements, Business Requirements, and Functional Requirements. This project may also require a simple architecture document to specify the type of hardware, software, and development environment necessary for the system.

Separate from requirements gathering is the Design phase is followed by the Coding phase. Design is when the requirements turn into actions, and the plan in the developers' head goes onto paper. Someone with a background in the architecture of the system puts together a plan for what methods to use and how certain algorithms should be implemented. If the design is done well, a coder just needs to type the design into the integrated development environment and click "Compile."

Testing is an important part of the software development life cycle. In addition to final acceptance testing, tests need to be run on each individual piece of the application and whenever pieces are combined together. Using the education received in Software Testing class in the fall of 2010, I will write test cases that accurately and fully test the important parts of the system. I plan to write unit test plans, integration test plans, and system test plans.

If my plan holds true to build a system that recalls data and answers questions about UNI students, one final system test will be to compare the results from my system against other systems in operation. One example is the Institutional Research website. This website contains facts and links to facts about UNI. The numbers on the site can be determined by asking questions of the database. Assuming the system works properly, the numbers created from a new system will match the numbers created by the original source system. Another source of reference is the UNI Fact Book, and individual pages from the Fact Book can be used to verify the system output.

Depending on the final outcome of the project, the system may be packaged for deployment. This process and the system as a whole will be documented if it is in fact done, but the type of packaging will depend on the receiving system.

All of these steps must occur between Monday, January 10, 2011 and Saturday, April 10, 2011. Below is a bulleted timeline for development:

- Week 1 - January 10, 2011 – January 16, 2011
 - Finish Initiation Phase by January 10, 2011
 - Finish High-level Business Requirements by January 16, 2011
- Week 2 - January 17, 2011 – January 23, 2011
 - Finish User Requirements, Business Requirements, Non-Functional Requirements by January 23, 2011
- Week 3 - January 24, 2011 – January 30, 2011
- Week 4 - January 31, 2011 – February 6, 2011
 - Finish Architecture Documents, Functional Requirements by February 6, 2011
- Week 5 - February 7, 2011 – February 13, 2011
- Week 6 - February 14, 2011 – February 20, 2011
 - Finish High Level Design and Master Test Plan by February 20, 2011
- Week 7 - February 21, 2011 – February 27, 2011
 - Begin Preliminary Development on February 21, 2011
 - Finish Low Level Design and Test Plan by February 27, 2011
- Week 8 - February 28, 2011 – March 6, 2011
 - Begin Main Development on February 28, 2011
- Week 9 - March 7, 2011 – March 13, 2011
- Week 10 - March 14, 2011 – March 20, 2011
 - Spring Break – no work because of mission trip
- Week 11 - March 21, 2011 – March 27, 2011
 - Finish Main Development by March 27, 2011
- Week 12 - March 28, 2011 – April 3, 2011
 - Finish Integration Testing by April 3, 2011

- Week 13 - April 4, 2011 – April 10, 2011
 - Finish System Testing, Regression Testing, Comparison Testing to other systems by presentation day
 - Present at Honors Research Day on April 9, 2011
- Week 14 - April 11, 2011 – April 17, 2011
 - Integration into final environment, user acceptance testing this week
- Week 15 - April 18, 2011 – April 24, 2011
- Week 16 - April 25, 2011 – May 1, 2011
 - Begin final reflection on April 25, 2011
- Week 17 - May 2, 2011 – May 7, 2011
 - Finish final reflection by May 2, 2011
 - Present to Computer Science students (if asked) on May 6, 2011
 - Graduate with Honors on May 7, 2011

Anticipated Results

The final project will be a system that allows natural language interaction with a sample database found in a university environment. It will allow queries similar to those commonly sent to Institutional Research and the Office of the Registrar at UNI or sent from academic departments such as the Computer Science department. The system should be able to recover from some, but not all, confusing queries. Not all queries will be recoverable, because of the scope of the project. The system should also perform basic mathematical calculations on results such as sum, count, and average.

References

- Androutsopoulos, I., Ritchie, G. D., & Thanisch, P. (1995). Natural language interfaces to databases – an introduction. *Journal of Natural Language Engineering* , 1(01), 1:29-81.
- Chamberlin, D. (1998). *A complete guide to DB2 universal database*. San Francisco: Morgan Kauffman Publishers.

- Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM* , 13(6), 377-387.
- de A. Barros, F., & DeRoeck, A. (1994). Resolving Anaphora in a Portable Natural Language Front End to Databases. *Proceedings of the 4th Conference on Applied Natural Language Processing* (pp. 119-124). Stuttgart, Germany: Applied Natural Language Conferences.
- Gray, T. (2009, May 1). The Secret behind the Computational Engine in Wolfram|Alpha [Web log message]. Retrieved from <http://blog.wolframalpha.com/2009/05/01/the-secret-behind-the-computational-engine-in-wolframalpha/>
- Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., & Slocum, J. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems* , 3(2), 105-147.
- IBM Archives: Edgar F. Codd.* (2003, April 23). Retrieved October 23, 2010, from International Business Machines: http://www-03.ibm.com/ibm/history/exhibits/builders/builders_codd.html
- Lehmann, H. (1978, September). Interpretation of Natural Language in an Information System. *IBM Journal of Research and Development* , 560-572.
- Minock, M., Olofsson, P., & Naaslund, A. (2008). Towards Building Robust Natural Language Interfaces to Databases. *Natural Language and Information Systems* (pp. 187-198). London: Springer.
- Santos, J. (2009). *Is Business Intelligence Relevant?* Midvale, Utah: Burton Group.
- Schafer, B. (2010). *Session 22: The Powerpoint Slides [PowerPoint slides]* . Retrieved from <http://www.cs.uni.edu/~schafer/courses/161/sessions/s22/>.
- Thompson, C. (2010, June 20). What is I.B.M's Watson? *The New York Times* , p. MM30.

Waltz, D. (1975). Natural language access to a large data base: an engineering approach.

Proceedings of the 4th international joint conference on Artificial intelligence. I, pp. 868-

872. Tblisi, USSR: Morgan Kaufmann Publishers Inc.