Hermes: Natural Language Access to a Medical Database

Carlos B. Rivera and Nick Cercone

Technical Report CS-98-03
March, 1998

© Carlos B. Rivera
Department of Computer Science
University of Regina
Regina, Saskatchewan, CANADA
S4S 0A2

ISSN 0828-3494
ISBN 0-7731-0361-9

Contents

1 Introduction **3**

2 Overview **3**
2.1 WWW version 4
2.2 UNIX version 5

3 Modules **6**
3.1 Control modules 6
3.2 Preprocessor module 7
3.3 Database access module 8

4 Parser module **10**
4.1 HPSG grammar and lexicon 10
4.2 AVM 10
4.3 Unification and Subsumption 11
4.4 Rules and Principles 11
4.5 Semantic extractor 13

5 Future enhancements **15**

1 Introduction

This document describes a NLIDB (natural language interface to databases) for medical databases. The interface discussed was designed and implemented at the University of Regina based partially on earlier work done on SystemX at Simon Fraser University [CHJ+90, CHM+94, MC91, VPC93]. This project was jointly funded by the IRIS 2 HMI-2 project and the IRIS Centres of Excellence. The new interface is named Hermes, in recognition of his ability as a messenger and his staff, the cadecus. The interface strives to improve the communication between the user and the database, similarly to Hermes between mortals and gods. The cadecus is widely used as a symbol for the medical profession.

Hermes uses an HPSG based parser to translate the users ad-hoc natural language (NL) queries into an intermediate form. This intermediate form is then translated into SQL that can then be run on the medical database designed by Weidong Yu [Yu96]. HPSG is a combined semantic and syntactic formalism that handles sentences, phrases and words. The grammar and lexicon used in the parser was originally developed by Shinta Mayasari [May95, May96a, May96b] and has undergone extensive modifications [Riv98].

Hermes' goal is to allow a user to access the database from a high level without knowing either the database schema or a formal database query language. This high level view would model what a hospital administrator would want. The following queries are samples of what Hermes handles:

- Give me the name of the head of radiology.
- List patients seen by head of radiology.
- Office and home phone number for Dr. Jane Aebig.
- Total number of pediatricians.
- Average age of all patients seen by female surgeons.
- Dr. Aebig's title.

The database Hermes connects to was modelled after the databases maintained by SaskHealth and the Regina Health district and reflect what a healthcare facility would have. The tables covered by Hermes are doctor and patient demographic information, patient weight and height data, patient allergies, insurance coverage information and visit information. The database is further described in Section 3.3.

The remainder of this document is divided into four sections. Section 2 gives an overview of the two version of Hermes presently implemented. Section 3 gives detailed descriptions of the control, preprocessor and database access modules. Section 4 describes the HPSG parser and semantic extractor that handles converting from NL to SQL. Section 5 gives planned future enhancements for Hermes. This paper concludes with two appendices that show a Hermes execution trace and gives sample NL queries the system handles.

2 Overview

There are two versions of Hermes presently implemented, an SGI UNIX version (Hermes:UNIX), the user interacts with through menus, and a World Wide Web (WWW) version (Hermes:WWW), the user interacts with through forms on webpages. Both versions of Hermes are composed of four

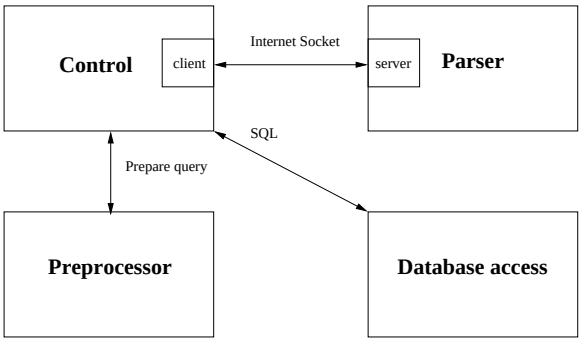

Figure 1: Hermes design

modules: control, preprocessor, parser and database access. These modules and their connections are shown in Figure 1.

The control module handles user input and output and calls routines to access the other three modules of Hermes as needed. The preprocessor changes the user entered queries from natural language to an intermediate format required by the parser. The parser module converts from the intermediate format to SQL executable by the database access module. The parser module is a server running on the fastest available computer, since parsing can be a very computationally intensive operation. The control program has a client stub built into it that connects to this server. The control program passes the intermediate format and receives back all valid SQL strings for the query. The database access module takes an SQL string and accesses Weidong's medical database. This database is an Oracle database running on Zeus, an SGI Challenge L parallel computer.

2.1 WWW version

The WWW version of Hermes is accessible from any browser that handles form (e.g. Lynx, Netscape, Explorer) at http://www.cas.uregina.ca/~systemx/Hermes. Figures 2, 3 and 4 shows the three pages comprising WWW Hermes (not shown is the page generated when there is an error).

Figure 5 shows the programs and flow of information for Hermes:WWW. Creating Hermes:WWW involved integrating four programming languages: C, Prolog, Perl and HTML. Using the form shown in Figure 2 the user enters their NL query. This NL query is passed to a Perl script to be extracted from the format outputted by the WWW form. This script replaces any apostrophes with spaces and removes all periods from the query. The modified query is then passed to a C program which functions as the control module for the WWW version, calling the preprocessor and parser. The output from the parser is checked to see if any errors occurred. If there are were any errors, these are returned to the Perl script and the C program exits. Otherwise, the SQL queries returned by the parser are sorted and duplicates removed. The remaining SQL queries are returned to the Perl script. The script checks if there were error messages and if so creates a web page to display the error messages and a link to error message help. If there were no errors returned, the script will generate a web page showing the total number of parses and number of unique parses. The page also has a radio-button (one-of-many) form showing the SQL queries and prompting the user to select one to run. Figure 3 shows this page for the NL query, *phone numbers for Jane Aebig*. There are two SQL queries shown because the parser was unable to decide whether the user wanted the patient or doctor demographic table to be queried for *Jane Aebig*. The selected SQL

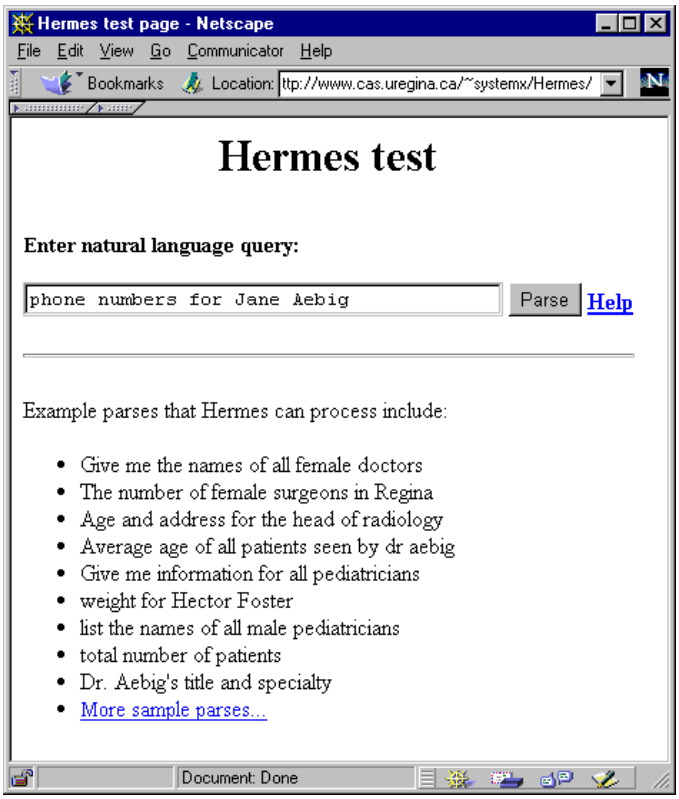

Figure 2: Hermes WWW enter NL page

query is passed to another Perl script which calls a stand-alone version of the database module to run the query. The query results are displayed on an additional page, as shown in Figure 4. The results can be save to a file using cut and paste from the WWW browser window.

2.2 UNIX version

The Hermes:UNIX offers more functionality than the WWW version. The beginning user can enter NL queries and the experienced user can enter SQL queries. The database results can be saved to a file directly from Hermes. There is extensive on-line help available. The user interacts with this version through on-screen menus. The main and help menus are shown in Figure 6. Appendix A shows a sample run for the UNIX version of Hermes.

The normal order of a query in Hermes:UNIX would be:

- user enters a NL or SQL query
- if query is NL then it is passed to preprocessor then to parser
- resultant SQL strings are displayed for the user, if more then one SQL string is generated the user is prompted to select one
- user can either choose to refine the query by entering a new NL or SQL query or run the resultant SQL

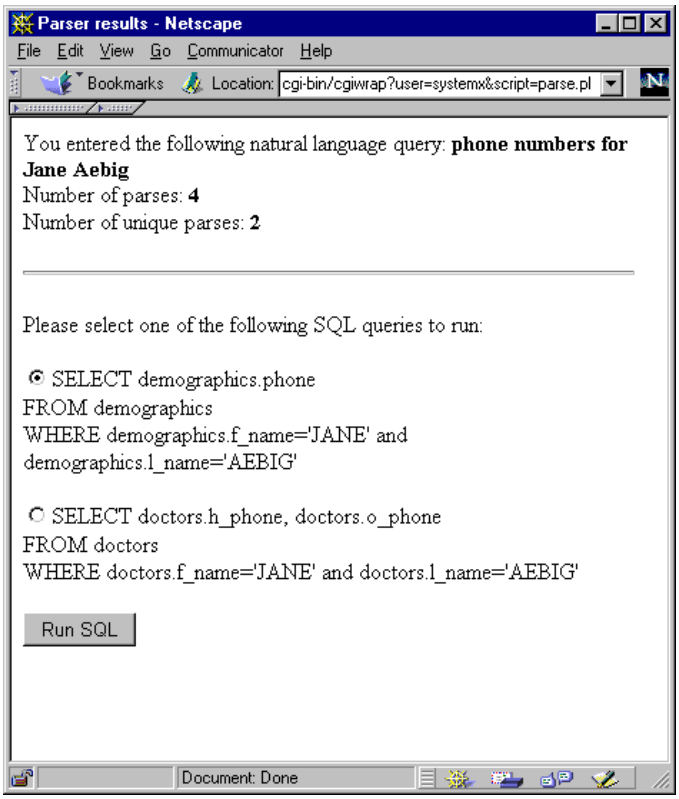

Figure 3: Hermes WWW choose SQL page

3 Modules

The control, preprocessor and database access modules are described in this section. The parser module is the most complicated module in Hermes and is covered in Section 4.

3.1 Control modules

There are two versions of the control module, one for Hermes:WWW and one for Hermes:UNIX. The two version share common code (the client stub for the parser server) but need to be compiled separately. The control module in Hermes:WWW is comprised of the two Perl scripts and C program shown in Figure 5. The interaction and function of these programs were described in Section 2.1. The control module in Hermes:UNIX is a C program that handles displaying the main and help menus, accepting user input in NL or SQL and for menu choices, sorting and removing duplicate SQL strings returned by parser, printing the unique SQL strings and handling user selection of same.

The control module starts off by printing the main menu and waiting for selection from the user. If the user selects and enters an SQL query, then the current SQL query is set to this. The user can then either display or run this query directly by choosing *View SQL* or *Run SQL* respectively from the main menu. If the user selects and enters an NL query, this is passed to the preprocessor. The preprocessor will return the format required by the parser. This is then passed to the parser. The parser returns an SQL for every parse it generates. In most cases even though there is more

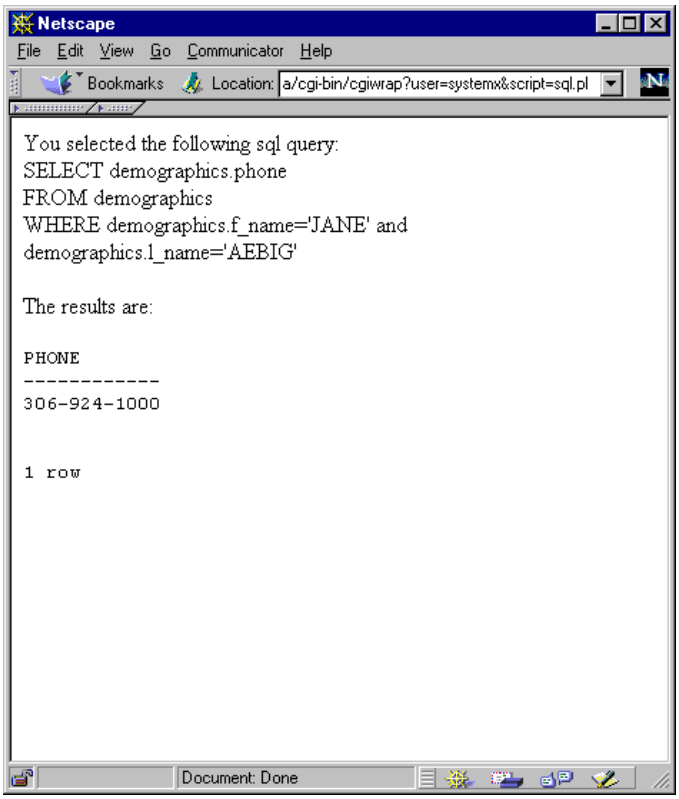

Figure 4: Hermes WWW database results page

than one parse the semantic content is the same. Therefore, the control program sorts the SQL queries and removes any duplicates. The program will report the number of parses and unique SQL queries. If there is only one SQL query then the current SQL query is set to this, otherwise all unique SQL queries are displayed and the user is prompted to select one to set the current SQL query to. After this the user can choose to view or run this SQL query. If the user chooses to run this query then the current SQL query is sent to the database access module.

3.2 Preprocessor module

The two versions of Hermes use the same preprocessor module. The preprocessor module handles converting natural language queries into the format expected by the parser. The parser is Prolog based and therefore expects a Prolog list of atoms. This means that the query must be enclosed in square brackets with each word in all small characters and separated by commas. As well, a period must appear after the closing bracket. Figure 7 shows an unconverted and converted phrase. The preprocessor also removes all whitespace. The parser has a single word, *s*, to represent the possessive *'s*, therefore apostrophes in the input query are replaced by commas. The parser can not handle periods, so the periods after an abbreviation such as *Mr* or *Dr* or at the end of a query are removed when converting from the input query to the intermediate form. The preprocessor module is written in C, although the first Perl script called in Hermes:WWW removes periods and changing apostrophe to spaces to facilitate passing the NL query to the C program.

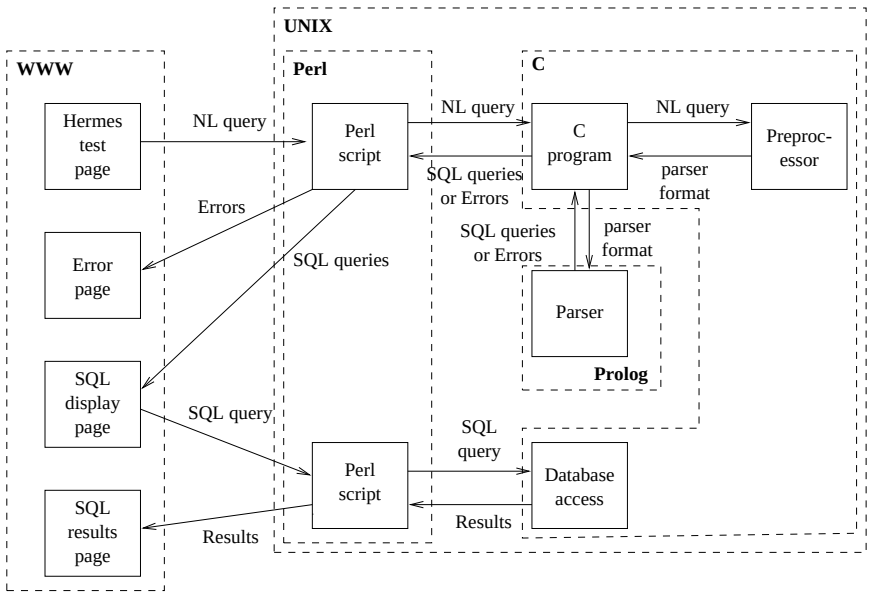

Figure 5: WWW Hermes design

3.3 Database access module

The two version of Hermes use the same database access module with minor modifications for Hermes:WWW. The Hermes:WWW version has a simple main() function wrapped around it, so that it can be run as a stand-alone program and has the saving to a file functionality disabled. The database access module is called when the user wants to execute their database query. The database module was written by Weidong Yu and handles an SQL query stored in a character string. The database is fully detailed in [Yu96]. This module normally prompts the user before running the actual database access to see if the results should be saved to a file or displayed on the screen. The database module handles the creation and filling of the file if that is what the users selects.

Hermes can not access every table in the database. Instead a subset of the database judged most interesting is used. Hermes can access the following tables (which are described in Weidong's manual):

ALLERGIES This table contains allergies for patients (e.g. milk allergy, penicillin allergy).

DOCTORS This table contains demographic information for doctors (e.g. first and last name, speciality, age).

DEMOGRAPHICS This table contains demographic information for patients (e.g. address, marital status, gender).

INSURANCES This table contains insurance information for patients (e.g. insurance number and type).

VISIT This table describes visit made by patients to doctors (e.g. referring doctor, attending doctor, admittance and discharge dates).

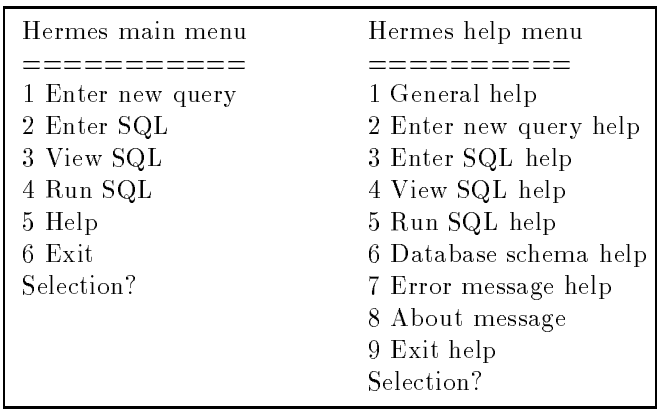

Figure 6: Hermes menus

Original phrase: Give me the office phone number and address of the head of radiology.
Converted phrase: [give, me, the, office, phone, number, and, address, of, the, head, of, radiology].

Figure 7: Preprocessed query

WEIGHT_HEIGHT This table contains weight and height information for patients (e.g. weight, height, date).

Hermes allows allows non-join and join queries to be executed on these tables. Non-join queries access only one table, for example:

- Last names for female radiologists.
- Names for patients in Regina.
- Dr. Aebig's specialty.
- Phone numbers for female patients.

Join queries allow up to three tables to be joined, for example:

- Patients seen by Dr. Affordable.
- Ages of patients seen by head of radiology.
- Addresses for all patients seen by female radiologists.

These queries can include the aggregate functions: AVERAGE and COUNT. The Hermes parser limits the number of columns selected with aggregate functions, but does not check to make sure a selected column is numeric. The following are sample queries involving aggregate functions:

- Average age of patients seen by Dr. Mary Aebig.
- Total number of surgeons in Regina.

4 Parser module

The parser module is the most complicated module in Hermes. Its purpose is to convert from a natural language phrase entered by the user to an executable SQL query. The parser module uses the Prolog programming language and a basic understanding of Prolog is required for the following section. The following are good Prolog references [Bra90, CM84, SS86].

To create the parser module, Sicstus Prolog 2.1 #9 is loaded with the Attribute Logic Engine (ALE) Prolog program. ALE "is an integrated phrase structure parsing and definite clause logic programming system in which terms are typed feature structures" [CP94]. ALE allows creation and manipulation of types, which are used in the Attribute Value Matrices (AVM), used in HPSG to store syntactic and semantic knowledge about words and phrases [PS87, PS94]. The ALE `compile_gram/1` predicate is used to compile the grammar and lexicon described in [Riv98]. The grammar contains the ALE rules that form a HPSG-based parser plus semantic rules hard coded for the health database. The lexicon contain approximately 300 words, half of which have associated semantic content, covering our health care database. A Prolog program, the semantic extractor, is loaded to translate from the output from the ALE `rec/4` predicate to an SQL query. The resulting parser module listens at an Internet socket for incoming phrases. The output back to the client is an SQL query for each parse. Figure 8 shows the parser configuration.

Figure 8: Parser configuration

4.1 HPSG grammar and lexicon

This section gives an overview of the HPSG grammar and lexicon used in Hermes to parse NL queries. A preceding techreport on Hermes [Riv98] has a more complete coverage of this topic. There are good tutorial on the HPSG formalism available on the World Wide Web [Man96, Mat97, Sag95]. The main parts of the HPSG formalism are: the central data structure, Attribute Value Matrices (AVM); the method used to join words into phrases, unification and subsumption; and the rules and principles used to control what words are joined.

4.2 AVM

The data structure central to the HPSG formalism is the AVM. An AVM is a feature structure that represents a *sign*. A sign is a lexical entry (word), phrase or a sentence. Figure 9 shows the AVM resulting from parsing *Office phone number for the head of radiology*. The words in capitals are *attribute* (also called *features*), followed by their *sorts*. Sorts label the attribute values, for example, there are several sorts a HEAD can take, such as *noun*, *verb* or *adjective*. The value for an attribute is the attribute or list of attributes indented below its name. Values are either simple (or atomic), like *ophone*, which is the value of the attribute HD, or complex. Complex values include lists, such as *columns_list* or *synsem_list*; sets, which are not implemented in the Hermes grammar; or another AVM, for example the value for MOD in an AVM for an adjective is an AVM representing common nouns. The AVM shown in Figure 9 is the first of six distinct parses generated by the parser. Each parse contains the same semantic information but has differing syntactic information.

The LOC attribute of an AVM takes two attributes CAT, which contains the syntactic information for the head of sign, and CONT, which contains the semantic information for the sign, as values. The CAT attribute takes four attributes as values: HEAD, part of speech information for word or head of phrase; MARKING, used in conjunctions (e.g. *and* and *or*) and complements (e.g. *for* and *that*); SPR, a list of signs that correspond to specifiers the word or phrase can take (e.g. determiners for common nouns) and SUBCAT, a list of AVMs the word or phrase can take as objects (e.g. a transitive verb would take a noun or prepositional phrase).

The CONT attribute takes the following attributes as values: ACTION, QUERY, REF and SEM_MARK. The ACTION attribute contains the database action required (e.g. `sum`, `count`, `average`) and has a default value of `db_action`. The QUERY attribute controls the columns selected, the where conditions and the tables selected for the query. The QUERY attribute takes a list of *query* objects. These objects each represent a single table, therefore having more than a single object in the QUERY list means that a table join is required. The *query* objects are composed of the TABLE and COLUMNS attributes. The TABLE attribute sorts representing the tables in the database that Hermes can access (e.g. *doctors_rel*, *visit_rel*). The COLUMN attribute is a list of objects that controls what where conditions are set for the final query. The HD value of each element in this list has a sort that selects the columns the user wants displayed (e.g. *ophone* in Figure 9). The objects in the COLUMNS attribute are a list of attributes corresponding to the columns that can be restricted for the current table. The where conditions are set when the value for one of these attributes is not `dval`. For example from the figure, there are two where conditions set: `DEPT eq ARG1 radiology` and `TITLE eq ARG1 director`. The REF attribute is used to control what phrases prepositions can take and SEM_MARK attribute is used in selecting what type of nouns can be used in nominal compounds (e.g. *office phone number*).

4.3 Unification and Subsumption

Unification is the joining of two structures (partial or whole) into a single structure that contains no more or less information than the original two structures. Subsumption is the ordering of values from general to more specific. The HPSG formalism insists on a there being an order to the attributes and values. This means that some values and attribute are more general or specific than others. For example, the value for NUM in Figure 9 is set to *num* the most general value. This value would subsume a value of *sing*, since *sing* is more specific. Unification creates the most general description that is compatible with the two inputs. Unification subsumes one value with another when creating the resultant AVM. Unification fails when neither of two values for the same attribute in the input AVM, subsumes the others (e.g. *sing* and *plur* for NUM attribute).

4.4 Rules and Principles

The great strength of HPSG is the limited number of schemas (rules) and principles used to join words into phrases and sentences. These rules and principles control how signs are joined to create larger signs. There are six schemas outlined in the HPSG formalism to handle different English grammatical structures: Head-Subject, Head-Complement, Head-Subject-Complement, Head-Marker, Head-Adjunct and Head-Filler. Hermes has all of these schemas except the Head-Marker and Head-Filler schemas, as well as two additional schemas to handle conjuncts and specifiers (determiners).

Schema 1 (Head-Subject Schema) This schema licences saturated phrases with a phrasal head daughter and one other daughter that is a complement. An example would be the sentence *She*

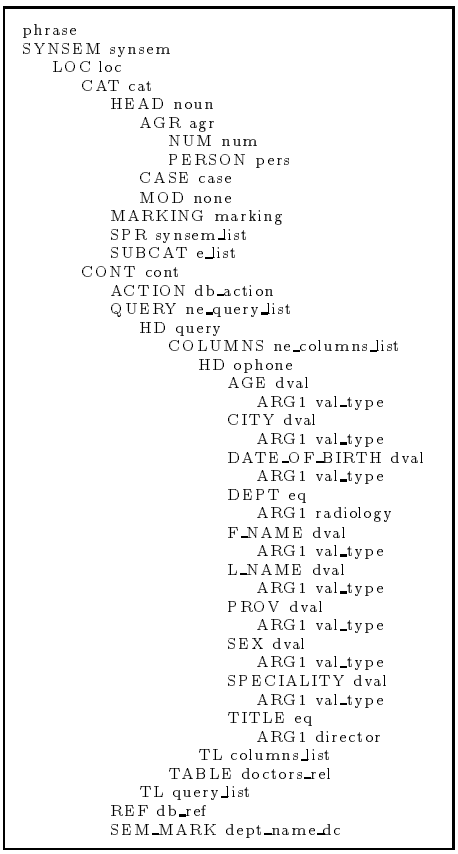

Figure 9: Example AVM

walks. The verb *walks* is the head daughter and *She* is the other daughter that complements *walks*.

Schema 2 (Head-Complement Schema) This schema licenses phrases that have a lexical head daughter (a word rather than a phrase) and zero or more complement daughters. These daughters have satisfied all their subcategorization requirements except the subject. The phrase *is a doctor* is licensed by this rule. The verb *is* is the lexical head daughter and the noun phrase *a doctor* the complement daughter.

Schema 3 (Head-Subject-Complement Schema) This schema licenses saturated phrases with 'subject-auxiliary inversion' clauses. This schema requires verbs to have an additional HEAD subattribute INV which takes the value + or -. A + means the verb allows inverted structures (e.g. *Can Kim go?*). Additionally the head daughter in this schema must be a lexical sign (a word).

Schema 5 (Head-Adjunct Schema) This schema licences phrases with an adjunct daughter and head daughter it selects. Adjuncts in HPSG have to be of sort *substantive* since *functional* sorts are handled by Schema 4. The MOD value of the adjunct is structure shared with the SYNSEM value of the head daughter. This schema licences modifying words (adjectives, adverbs) and the words they are modifying (nouns, verbs). An example phrase would be *first name*, where *first* is the adjunct and *name* the head it modifies.

Specifier-Head Schema This schema licences phrases with a specifier daughter and the head daughter it selects. The specifier daughter has to have a HEAD sort of *det(erminer)*. An example would be the phrase *the doctor*.

Conjunct Schema This schema licences phrases consisting of two daughters separated by a conjunction. The two daughters have to have the same HEAD, SUBCAT and SPR values. An example would be the phrase *doctors and surgeons*.

The HPSG formalism outlines many universal principles covering not only English grammatical and semantic rules but also includes principles that cover other languages. The HPSG grammar defined in Hermes has the following universal principles that handle most English grammatical structures.

Head feature principle The HEAD value of any headed phrase is structure-shared with the HEAD value of the head daughter.

Subcat principle The SUBCAT value of any headed phrase is the concatenation of the list values of the SYNSEM values of the complement-daughters.

Specifier principle In a headed phrase, where the daughter has a HEAD value of sort *functional*, the SPEC value of that value must be structure-shared with the phrases SYNSEM value.

Marking principle The MARKING value of any headed phrase is structure-shared with the MARKING value of daughter if any.

The semantic principle used in Hermes handles unifying the CONT value of phrase or words. This principle was adapted to handle Weidong's medical database, but could be easily modified to handle any relational database. The semantic principle works by individually combining the ACTION, SEM_MARK, REF and QUERY attributes for the words or phrases being joined. The ACTION attribute of the semantic head and complement daughters must be unifiable or this principle fails. This means trying to join the words *average* and *sum* will fail, but *average* and *weight* will not since the ACTION attribute for *weight* is undefined. The SEM_MARK attribute is inherited from the semantic head daughter except for prepositional phrases where it is obtained by unifying the semantic head and complement daughters. The REF attribute is inherited from the semantic head daughter. Unifying the QUERY attribute for words or phrases involves more work because there are several cases that must be covered. First, the TABLE and COLUMNS attribute for the two signs are unified. This will only succeed if the TABLE and COLUMNS values are the same for both signs (or if one is undefined). If the unification fails then the TABLE attributes for the two sign are unified (again only if they have the same value or one is undefined) and the COLUMNS list of the mother is formed by concatenating the COLUMNS list of one sign to the COLUMNS list of the other sign. If the TABLE unification fails this means that the TABLE value of the two signs are different and a table join is required. The TABLE values of the two signs are compared to make sure they have a common attribute (e.g. the DEA# attribute in the doctor table and the REF_PHYSICAN attribute in the visit table). If they do have a common attribute than the QUERY value of the mother is the concatenation of the QUERY list for one sign and the QUERY list of the other sign.

4.5 Semantic extractor

The semantic extractor is a Prolog program consisting of approximately 700 lines of commented code and 38 predicates. The semantic extractor handles converting from the intermediate format

returned by the HPSG parser to an executable SQL string. The intermediate form for the query *Office phone number for the head of radiology* is shown in Figure 10. This form uses nested structures to convey the same information shown in Figure 9.

The semantic extractor first extracts the CONT value from the rest of the AVM (the CAT value is not presently used in converting from the intermediate form to SQL so it is discarded). The QUERY and ACTION values are then extracted from the CONT value (the REF and SEM_MARK values are not used in converting from the intermediate form to SQL so they are also discarded).

phrase(_1100-synsem(_1095-loc(_1089-cat(_1081-noun(_1074-agr(_1068-num,_1065-pers),_1062-case,_1059-none),_1056-marking,_1053-synsem_list,_1050-e_list),_1047-cont(_1039-db_action,_1036-ne_query_list(_1030-query(_1024-ne_columns_list(_1018-ophone(_1001-dval(_996-val_type),_993-dval(_988-val_type),_985-dval(_980-val_type),_977-eq(_972-radiology),_969-dval(_964-val_type),_961-dval(_956-val_type),_953-dval(_948-val_type),_945-dval(_940-val_type),_937-dval(_932-val_type),_929-dval(_924-val_type),_921-dval(_916-val_type),_913-dval(_908-val_type),_905-eq(_900-director)),_897-columns_list),_894-doctors_rel),_891-query_list),_888-db_ref,_885-dept_name_dc))))

Figure 10: Intermediate form

The list composing the QUERY value is decomposed to find the number of tables that are being accessed. While extracting information from this list a structure is created to store information about the current query. This structure is a list of structures, with each structure containing query information for a single database table. Therefore, a join is required if there is more than one element in this list. The label for the structures in the list is the table being accessed (e.g. doctor or demographics) and each structure takes two arguments. The first argument is a list containing the selected columns and the second argument is a list containing the where conditions. An example structure for the phrase *names of female surgeons* is:

```
[doctor([doctor.f_name, doctor.l_name], [specialty(eq, surgeon), sex(eq, female)])]
```

and for the phrase *female patients seen by Dr. Aebig*:

```
[demographics([],[demographics.sex='F']),doctors([],[doctors.l_name='AEBIG'])
,visit([],[])]
```

After the number of tables being queried is found the The TABLE and COLUMNS values for each table are extracted. The table name is converted from the values in the grammar to the table name in the database (e.g. *doctors_rel* to *doctors*). The parser can return an AVM where there is no TABLE value specified (e.g. for the word *name*). The semantic extractor presently generates an error message when there is no TABLE value specified.

The COLUMNS value is a list with each element containing the selected columns and where conditions for the current table. Each element is extracted separately and its values added to the structure for this table. A query with multiple selected columns from the same table (e.g. *weight and height*) will have two elements in its COLUMNS list, one with a HD value of *weight* and one with a HD value of *height*. The selected columns are concatenated with a period to the end of the converted table name and inserted into the first list argument for the current table (see `[doctor.f_name, doctor.l_name]` above). Depending which rule was used to join the signs the where conditions can be in either or both elements. These where conditions are extracted and converted to the structures shown in the second list argument for the current table (see `[specialty(eq, surgeon), sex(eq, female)]` above).

After the structure has been completely built from the passed AVM it is passed to the predicate that handles printing the SQL string. First, a check is made to see if there are no selected column in any of the passed tables (e.g. in an under-specified query such as *female doctors*). If there are no selected columns then default column are assigned. If the ACTION attribute has a value assigned a check is made that only one or none (as needed) columns are selected. If the wrong number of columns is selected an error message is printed. The table name, selected columns and where conditions for all tables are combined into a single list for each. The list is sorted to facilitate finding the unique SQL strings in the control programs. The selected columns and tables are then printed. Printing the where conditions involves slightly more work since there can be no where conditions to print (e.g. *last name of all doctors*) and a join condition must be added for all tables in the table list that have a common column (e.g. `visit.referring_doctor = doctors.dea#`).

5 Future enhancements

The following enhancements are planned for Hermes:

Continued expansion of the lexicon The existing Hermes lexicon contains approximately 300 words. These words are mostly specific to the medical domain but do not yet give good coverage of all the information contained in the medical database.

Different lexicons and databases Ability to connect to different lexicons and databases (there will need to be a different lexicon for each database, since the lexicon contains database specific information). Major portions of the lexicon will be re-usable though since words have no specific database connotations (e.g. *is*, *a*, pronouns).

Save features Adding the ability to "can" (save queries) in Hermes:UNIX would greatly enhance the usability of Hermes.

References

[Bra90] I. Bratko. *Prolog Programming for Artificial Intelligence.* Addison-Wesley, second edition, 1990.

[CHJ+90] N. Cercone, G. Hall, S. Joseph, M. Kao, W. Luk, P. McFetridge, and G. McCalla. Natural Language Interfaces: Introducing SystemX. In T. Oren, editor, *Advances in Artificial Intelligence in Software Engineering*, pages 169–250. JAI Press, Greenwich, Conn., 1990.

[CHM+94] N. Cercone, J. Han, P. McFetridge, F. Popowich, Y. Cai, D. Fass, C. Groeneboer, G. Hall, and Y. Huang. SystemX and DBLearn: easily getting more from your Relational Database. *Integrated Computer-Aided Engineering*, 1(4):311–339, 1994.

[CM84] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, Berlin, 2nd edition, 1984.

[CP94] Bob Carpenter and Gerald Penn. *ALE: The Attribute Logic Engine User's Guide. Version 2.0.1*, December 1994.

[Man96] Suresh Manandhar. *A Course on HPSG grammars and typed feature formalisms.* Language Technology Group, Human Communication Research Centre, University of Edinburgh, http://www.ltg.hcrc.ed.ac.uk/projects/ledtools/grammar_writing/, 1996.

[Mat97] Colin Matheson. *HPSG Grammars in ALE.* University of Edinburgh, http://www.ltg.hcrc.ed.ac.uk/projects/ledtools/ale-hpsg/, 1997.

[May95] Shinta Mayasari. An HPSG Lexicon for a Physical Activity Database. Technical Report CS-95-09, Computer Science Department, University of Regina, September 1995.

[May96a] Shinta Mayasari. *A Natural Language Interface to a Physical Activity Database: Design and Its Implementation.* University of Regina, Summer 1996.

[May96b] Shinta Mayasari. *An Overview of System Y.* University of Regina, Fall 1996. This paper is incomplete.

[MC91] P. McFetridge and N. Cercone. Installing an HPSG Parser in a Modular Natural Language Interface. In *Computational Intelligence III*, pages 169–178. North Holland, Amsterdam, 1991.

[PS87] Carl Pollard and Ivan A. Sag. Information-Based Syntax and Semantics, Volume 1: Fundamentals. CSLI Lecture Notes 13, Standford: Center for the Study of Language and Information, 1987.

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* The University of Chicago Press, 1994.

[Riv98] Carlos B. Rivera. Hermes: Grammar and Lexicon. Technical report, University of Regina, March 1998.

[Sag95] Ivan Sag. *Materials for teaching HPSG.* Stanford University, http://hpsg.stanford.edu/hpsg/lecture-materials/lecture-materials.html, 1995.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. The MIT Press, Cambridge, Massachusetts, 2nd edition, 1986.

[VPC93] C. Vogel, F. Popowich, and N. Cercone. Logic Based Inheritance Reasoning. In Morris Sloman, editor, *Prospects for Artificial Intelligence*. IOS Press, 1993.

[Yu96] Weidong Yu. *Document of Project: Natural Language Interface to Healthcare Database (database part)*. University of Regina, September 1996.

Appendix A - Sample run

The following is a sample run of Hermes:

```
zeus[1]% hermes

Hermes 1.0; March 1998
Copyright (C) 1997, 1998 Carlos Rivera and Nick Cercone
All rights reserved.
Please report any problems with this software to systemx@cas.uregina.ca.

                Hermes main menu
                ===============
         1  Enter new query
         2  Enter SQL
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 1

 Enter new query: phone number for the head of radiology

Parsing...
Done parsing!

There were 3 parses returned by the parser of which 1 are unique SQL strings.
The unique parse(s) are:
1)
SELECT  doctors.h_phone, doctors.o_phone
FROM    doctors
WHERE   doctors.dept='RADIOLOGY' and doctors.title='DIRECTOR'

Setting the SQL query equal to the SQL returned by the parser!

                Hermes main menu
                ===============
         1  Enter new query
         2  Enter SQL
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 4

         +-------------------------------------------------------+
         |               OUTPUT OF ANSWERS                       |
         |   Press <ret> key:   output to screen.                |
         |   Input a file name: output to the assigned file.     |
         +-------------------------------------------------------+

Your choose:

H_PHONE       O_PHONE
------------  ------------
```

```
306-757-1001  306-584-4001


1 row processed.

                Hermes main menu
                ================
         1  Enter new query
         2  Enter SQL
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 1

 Enter new query: phone numbers for Jane Aebig

Parsing...
Done parsing!

There were 4 parses returned by the parser of which 2 are unique SQL strings.
The unique parse(s) are:
1)
SELECT  demographics.phone
FROM    demographics
WHERE   demographics.f_name='JANE' and demographics.l_name='AEBIG'

2)
SELECT  doctors.h_phone, doctors.o_phone
FROM    doctors
WHERE   doctors.f_name='JANE' and doctors.l_name='AEBIG'

Please select the SQL query you want to run: 2

                Hermes main menu
                ================
         1  Enter new query
         2  Enter SQL
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 3

 Current SQL string is:
SELECT  doctors.h_phone, doctors.o_phone
FROM    doctors
WHERE   doctors.f_name='JANE' and doctors.l_name='AEBIG'


                Hermes main menu
                ================
         1  Enter new query
         2  Enter SQL
```

```
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 4

         +-----------------------------------------------------+
         |               OUTPUT OF ANSWERS                     |
         |   Press <ret> key:   output to screen.              |
         |   Input a file name: output to the assigned file.   |
         +-----------------------------------------------------+

Your choose:

H_PHONE       O_PHONE
------------  ------------


0 rows processed.

                 Hermes main menu
                 ===============
         1  Enter new query
         2  Enter SQL
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 1

 Enter new query: phone numbers for Jane Aebig

Parsing...
Done parsing!

There were 4 parses returned by the parser of which 2 are unique SQL strings.
The unique parse(s) are:
1)
SELECT  demographics.phone
FROM    demographics
WHERE   demographics.f_name='JANE' and demographics.l_name='AEBIG'

2)
SELECT  doctors.h_phone, doctors.o_phone
FROM    doctors
WHERE   doctors.f_name='JANE' and doctors.l_name='AEBIG'

Please select the SQL query you want to run: 1

                 Hermes main menu
                 ===============
         1  Enter new query
         2  Enter SQL
         3  View SQL
```

```
        4  Run SQL
        5  Help
        6  Exit
        Selection? 4

        +-----------------------------------------------------+
        |               OUTPUT OF ANSWERS                     |
        |   Press <ret> key:   output to screen.              |
        |   Input a file name: output to the assigned file.   |
        +-----------------------------------------------------+

Your choose:

PHONE
------------
306-924-1000


1 row processed.

                Hermes main menu
                ================
        1  Enter new query
        2  Enter SQL
        3  View SQL
        4  Run SQL
        5  Help
        6  Exit
        Selection? 5

                Hermes help menu
                ================
        1  General help
        2  Enter new query help
        3  Enter SQL help
        4  View SQL help
        5  Run SQL help
        6  Database schema help
        7  Error message help
        8  About message
        9  Exit help
        Selection? 1

General help
============
 Welcome to the Hermes natural language (NL) interface to medical database.
Hermes allows a novice or experienced database user to quickly access information
from a medical database.  Hermes supports ad hoc queries through NL.  These
natural language queries are translated into SQL that can be used to directly
access the database.  An experienced user can enter an SQL query directly into
Hermes.

 Hermes uses a client-server architecture to do the NL parsing and database
retrieval.  This means that there will be a slight pause while the clients
```

```
connect to the parser or database server.

 The following steps outline how to use Hermes to access the medical database:
1) Enter a new query using NL by selecting the New query menu option OR enter
an SQL query by selecting the Enter SQL query option.
2) If you used NL then after connecting to the parser and converting the
passed NL query to SQL, Hermes will display the unique SQL queries returned.
The user should select the SQL query that corresponds to their search OR select
a query then use New query to refine your NL query.
3) You can select View SQL to see the SQL query that will be run.
4) After entering or selecting the SQL query that matches what you want to do
use Run SQL to run the SQL query.

 The rest of this help menu is divided into separate help files covering different
parts of Hermes.  The menu options available on Hermes' main menu are covered in
the help files: Enter new query help, Enter SQL help, View SQL help and Run SQL
help.  An overview of the medical database that Hermes connects to is covered in
Database schema help.  Database schema help has separate help files for the six
tables that can be accessed: doctors, demographics, allergies, insurances,
weight_height and visit.  The error messages that can be generated are covered in
Error message help.  The final option on the help menu will print out an About
message for Hermes.


                Hermes help menu
                ================
         1  General help
         2  Enter new query help
         3  Enter SQL help
         4  View SQL help
         5  Run SQL help
         6  Database schema help
         7  Error message help
         8  About message
         9  Exit help
         Selection? 9

                Hermes main menu
                ================
         1  Enter new query
         2  Enter SQL
         3  View SQL
         4  Run SQL
         5  Help
         6  Exit
         Selection? 6
zeus[2]% exit
```

Appendix B - Sample parses

The existing system handles (but is not limited) to the following sample parses.

```
Give me the names of all female doctors.
Give me the names of all surgeons.
Give me all pediatricians.
Give me information for all pediatricians.
Give me the <first, last> name of all pediatricians.
Give me the phone number for Dr. Aebig.
Give me the phone number for the head of radiology.
The number of female surgeons in Regina.
Name and addresses of all female surgeons.
Home phone <of, for> head of radiology.
Give me Dr. Aebig's age.
Head of radiology's phone number.
Dr. Aebig's title.
list the name of all family physicians
birth date of all male pediatricians
address and phone numbers of all female surgeons
dr Aebig s specialty
<gender, sex> of dr Aebig
specialty of dr Aebig
phone number and addresses for female surgeons
phone number and address of all doctors
phone number and address of the head of radiology
age and address for the head of radiology
name and address for head of radiology
first name and phone number for the head of radiology
total number of doctors
Give me the doctor's average age.
Average age of female surgeons.
average age of all patients <seen> %% note different semantic meaning
average age of all patients seen by dr aebig
average age of practicing surgeons
average age of patients
count the number of female gynecologists
patients seen by dr Aebig
patients examined by dr Aebig
```