SOPHIA: Providing basic knowledge services with a common DBMS

Neil F. Abernethy
Stanford Medical Informatics
Stanford University, Stanford, CA 94305
nfa@smi.stanford.edu

Russ B. Altman*
Stanford Medical Informatics
Stanford University, Stanford, CA 94305
rba@smi.stanford.edu, * Corresponding Author

Abstract

Knowledge base systems have not captured the wide audience enjoyed by database management systems (DBMS). The capabilities of the most high-end knowledge representation and delivery systems are extensive, but may not be pragmatic or accessible to a large community of potential users. We present SOPHIA, a frame-based knowledge architecture which is built upon a relational DBMS (RDBMS) to provide basic frame access capabilities. SOPHIA stores frames in a table with frame/slot/value fields, and does not map individual knowledge classes to separate tables. It complements existing ontology tools because it allows the output of these systems to be compiled and delivered from a more modest DBMS. The KB query/update functions of Sophia are written in SQL and are based loosely upon the OKBC core specification. These functions can be accessed by clients through a URL mechanism or ODBC calls. SOPHIA is being used to deliver a knowledge base back-end for two molecular biology software applications.

Proceedings of the 5th KRDB Workshop
Seattle, WA, 31-May-1998
(A. Borgida, V. Chaudri, M. Staudt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-10/

Introduction

As computer applications in scientific domains such as molecular biology have grown more complex, they have begun to shift to object-oriented methods for representing complex data types. Even disregarding distributed sources of data, the structure of individual datasets is often too complex to represent effectively in a relational database. Straightforward relational representations can leave out important dependencies of interest, and can effectively fit the data to the capabilities of the database structure instead of fitting the structure to the data.

Knowledge bases representation systems (KBRSs) and Object-oriented Database systems (OODBMS) address this problem by more closely modeling the entities in the system of interest and the interactions between them. These systems allow the use of object hierarchies and dependencies to describe the organization of information. They typically provide benefits over RDBMS in the richness of available queries over these more complex data types.

Unfortunately, KB and OODBMS technologies do not yet have a broad base of established users. Object oriented methods require not only that a researcher specify the properties of entities, but also that they map them onto programming language and database structures. Numerous tools such as LOOM, Classic, and Ontolingua have arisen to allow the generation of domain ontologies independent of their programmatic usage.

Nevertheless, there has been a lack of standard interfaces and tools for the design and use of these systems. Evolving standards for KB systems include KIF (the self-descriptive Knowledge Interchange Format language) [Genesereth et al, 1992], OKBC (a generic knowledge base access/update protocol) [Chaudhri et al, 1998], KRSS (a description logic standard) [Patel-Schnieder et al, 1993], KQML (an agent-based knowledge sharing language) [Finin et al, 1994], and numerous SQL extensions, including OOSQL [Steenhagen et al, 1994]. However, most of these have yet to gain wide acceptance or usage. The vast majority of ongoing development in scientific and industrial applications still utilize complex SQL queries on relational databases as their primary method of information access. Furthermore, the exponential growth of the World Wide Web (WWW) has increased the investment in and immediate utility of such applications.

Database and Knowledge Base technologies need not be mutually exclusive. Several systems have mapped a knowledge model into a database. The HYWIBAS system [Norrie et al, 1994] maps knowledge base frames first onto an object model, which is in turn mapped onto a relational database structure. The PERK database [Karp et al, 97] serves as a back end for the OCELOT FRS (and others under the OKBC model), using a non-KB-specific mapping. PARKA uses a database model and multiple tables (for different types of objects) to deliver very large databases with a persistent backend [Spector et al, 1992].

The gap between these technologies can be bridged most easily for programmers and users new to knowledge bases by making use of prevailing tools and experience as the backbone for new systems. By using knowledge-level queries based on SQL and RDBMS, dependencies on the underlying information storage system can be gradually removed.

The primary problems that SOPHIA addresses are:

(1) How can we deliver basic knowledge services widely without asking users to purchase and learn an expensive or special-purpose OODBMS or KBMS? and

(2) Can a KB be represented with sufficient expressive power and performance in a DBMS that it can be served using off-the-shelf software?

Methods

SOPHIA is based on an ontology model similar to the Ontolingua system [Farquhar et al, 1996]. It represents all frames in a hierarchical classification tree, and distinguishes *concept* frames from leaf *instance* frames. Class slots are distinguished as *own slots* (associated with a class, but not its instances) or *template slots* (associated with instances, default values inherited). Slots on instances are simply attached to the frame (similar to own slots). Facets modify frame-slots pairs, used for typing, cardinality and other restrictions.

In order to store frames efficiently, we have used a RDBMS to store all information which constitutes the frame. The commonly available database Access97™ from Microsoft was used, although the database design is general, and is portable to other RDBMS (see discussion). This choice also maintains portability of supporting code from the database (Visual Basic for Applications, VBA) to the web server (Visual Basic Script, VBScript) or stand-alone applications (Visual Basic).

Our design employs a simple set of indexed tables to store frame and class information. Performance is improved and advanced options are supported with additional tables for a cached hierarchy, user information, and user-interface specifications.

The largest and most important table contains all instance frame definitions. We use a many-to-many structure to store a set of tuples describing the slots and values which compose each frame. Thus, each record in the table contains a frame, slot, and value. The full definition for a frame requires several records, one for each component slot and value. Multiple values are represented with multiple records. This table resembles the primary *frames* table of the PERK database [Karp et al, 1997]. Our design differs from PERK in that it does not place *Instance-Of* relationships (classifying instances) into a separate table. We also store additional information in each such record representing data ownership, access permissions, and expiration date. These additional properties are essentially privileged facets that must contain a single value for each frame-slot-value triplet.

Two additional tables contain the full class frame definitions. The first holds all of the class frames, and differs from the frames table by distinguishing between template- and own-slots. This separates the description of the class from default properties inherited by its instances. The second class table contains a computed ordering of all class-subclass relationships as determined by the classification hierarchy. This table aids in the processing of several queries involving class membership and inheritance, since subclass instances are grouped with associated superclasses.

The database makes use of other tables to contain information on facets, users, privileges and user-interface options. These can be sacrificed for simpler, read-only ontologies.

With these tables in place, SQL queries can then retrieve frames from the KB based on program- or user-specified conditions. The supported queries are modeled loosely after a subset of the core OKBC functions. The selection of queries supported was based primarily on the immediate needs of RiboWeb project, which the KB supports on the back end [Chen et al, 1997]. Generally, these queries allow the user to request information on frames, classes, and slots. For example, the **get-class-instances** query will return a list of all frames which are instances of a given class or any of its subclasses (example given below).

The SQL queries themselves are executed through a scripting language across a WWW server. Virtually any scripting language can be used (in conjunction with a compatible database connection protocol of choice such as ODBC). This allows complex operations or queries beyond the capabilities of SQL (including the more complex inferencing) to be processed. These scripts also serve to format the output for client applications.

The basic interface to the KB, an HTML-based browser, serves many functions. Its most basic function is to display frame references with links back to the frame in the KB, similar to the Ontolingua system. However, to enable more complex data types to be displayed, and references to other resources to be represented, values on each slot (relation) are displayed in a specification peculiar to that slot.

For example, the value of a slot with a *Range* of *String* would be displayed without a hyperlink. Alternately, if a slot's *Range* were *Image-Ref*, then the value of that slot would be presented as an HTML image tag, causing the value "http://www.stanford.edu/my-image.gif" to be loaded as an image in the browser window. In a similar fashion, values in the KB can be references to external WWW databases, or can be piped into Java applets. We have used this facility to link to the online biological databases (Ribosomal Database Project - RDP [Maidak, 1996] and NCBI's Medline database [http://www.ncbi.nlm.nih.gov/PubMed/]). In this way, the SOPHIA contains a representation about its external environment, potentially including other online knowledge bases and databases.

Results

The SOPHIA KB server is used as a back end for several knowledge based systems in our lab, serving both the client applications and the stand-alone web-based KB browser. The client applications are RiboWeb [Chen, 1997], and its spin-offs MHCWeb and Oweb [Hon et al, 1998] These systems deliver structured scientific data to scientists using standard ontologies to organize the critical experimental concepts in the scientific domains. They then allow standard analyses to be performed on the data, such as testing for consistency of the data or retrieving relevant subsets of the data. The systems store the organizing ontologies and data instances in SOPHIA .

In some cases, we found it helpful to develop the class structure of the knowledge bases in Ontolingua or Protege, and then compile the knowledge base into SOPHIA for delivery. The shared frame-based data model allows for easy translation of ontologies and data instances between these three systems (as long as system-specific features such as axiomatic reasoning in are not used). In other cases, we have built the ontology and data instances directly in SOPHIA. For this purpose, we have built a prototype KB Editor for acquiring the structure and content of a knowledge base.

The packaged-SQL queries can be accessed directly through ODBC libraries or processed for further output. Developers may write their own queries using this approach, allowing direct access through SQL. To provide transparency of the database, we have written server-side wrapper scripts to return output in a simple ASCII frame|slot||value|||… syntax. Any program which can parse this output can retrieve frames from the KB.

In the case of a client application written in Perl, the Perl API provides access to the library of functions, including: ***get-class-instances, get-class-instances-direct, get-class-all-supers, get-class-direct-supers, get-slots-of-frames, get-slots-of-frames-with-query, create-class, create-slot,*** and ***create-individual***. The API serves the purposes of passing queries to SOPHIA, retrieving data over HTTP, and parsing the resulting frame information.

For example, one commonly used Perl API function is the function that gets all instances of a particular class (including instances of its subclasses), called here on the class *Journal-Article*, one of the concepts in our molecular biology system:

get-slots-all-instances-of-class ($class,@slot_list)

> where the variable *$class="Journal-Article"* and the array *@slot_list=("Journal-Name", "Publication-Year")*

This Perl program uses URL to pass the following data to the server:

> *http://SOPHIA.stanford.edu/get-slots-all-instances-of-class.asp?class=Journal-Article&slots=Journal-Name&slots=Publication-Year*

The corresponding server-side SQL is then executed:

```
Select Distinct [fsv].[frame], slot, value
from [fsv] where [frame] in
(SELECT [fsv].frame
FROM [fsv] INNER JOIN [superclasses] ON
[fsv].[value2] = [superclasses].class
WHERE (([fsv].[slot]='instance-of') AND
([superclasses].[superclass]='{insert class
name}')))
and slot in ({insert slot list})
```

And returns the string:

> *Jmb-Stern-200-291|Journal-Name||J Mol Biol|||Jmb-Stern-200-291|Publication-Year||1988|||Jmb-Svensson-200-301|Journal-Name||J Mol Biol|||Jmb-Svensson-200-301|Publication-Year||1988|||Nar-Atmadja-13-6919|Journal-Name||Nucleic Acids Res|||Nar-Atmadja-13-6919|Publication-Year||1985|||…*

The server-side SQL queries use Internet Database Connectivity (IDC) or Active Server Pages (ASP) to connect to an ODBC datasource (in this case Access97). IDC is simply a template for SQL output, similar to approaches in other databases. ASP can be configured to use several scripting languages, such as VBScript (based on Visual Basic), JavaScript, or even Perl. The same code, altered to generate HTML, displays formatted results within WWW browsers. The same design could be followed in a stand-alone server application based on another language and protocol with HTTP and database support (such as Java and JDBC).

In our current design, processing of the queries always takes place after the SQL is executed. Some processing can be performed by SQL alone, but this approach limits our ability to use computed values in post-processing. Generality of the queries carries a cost in execution speed, but can be overcome by optimizing queries for specific database systems.

The first step in post-processing of queries is to regulate the security of the transaction, and return only those values which are viewable by the *user*, *group*, and *world*. Next, frames are checked to make sure that they are current. This allows cached values to be filtered out or old data to be reviewed by the system manager. The order of these tests can be reconfigured to prune records first on fields likely to fail, saving further computation.

The last step in post-processing is formatting query results for output. This can be either the frame|slot||value||| packaging or a variety of other specialized purposes, including HTML (for the KB browser), MIME file types (Protégé's *.pont file type [Musen et al, 1993], VRML files for 3-D visualization [Bell, 1994]). Exporting to other MIME file types is a function that would be performed by a specialized query, as distinct from the Range specification described above, which formats individual frame values.

The efficiency of the system is dependent largely on the context of its use. In comparison to memory-resident systems, a database back-end offers many advantages. For instance, frames do not have to be loaded from a flat file in order to be queried, eliminating the start-up delay [Karp et al, 1997], Futhermore, virtual memory and caching performed automatically by the DBMS result in speedier performance for common queries. Performance hits are taken, however, for processing queries for output and for network transport. Access97 is sufficient for a limited number of users, with few concurrent queries to the knowledge base. However, a larger applications would likely benefit from industrial-scale databases which are optimized to process multiple concurrent queries.

Figure 1 shows a plot of three typical queries run against the KB. Queries were tested on the Ribosome-2 knowledge base, which contains approximately 15,000 frames, and on 5x and 15x multiples of this knowledge base. Queries involving various classes and instances are shown. The queries in this figure were run with the network API, and the times are consequently larger than for local queries. A simpler query such as *get-class-instances* can return over 1000 frames/second when run locally.

SOPHIA has been constructed to maintain maximum portability to other platforms and databases, and hence minimizes the use of stored queries (views) and database-specific procedures which would further optimize performance.

Figure 1. This plot shows the average times for three different queries to retrieve frames from SOPHIA using the network API. The queries returned 8673, 505, and 1310 records respectively (with a typical frame consisting of 10 records). Typical queries return 20-100 frames per second for a KB with under 200,000 frames (In this example with KBs with 15,000-225,000 frames, the range is 7-79 frames per second).

Discussion

SOPHIA is limited in its power: axiomatic inference must be coded into the structure of the SQL queries or achieved by processing the output of groups of SQL queries. Inference is supported in three ways. Class/subclass subsumption is optional for *subclass-of* and *instance-of* relations in queries (we have also tested this with other relations such as *part-of*). Inheritance and some constraint-enforcement are performed when instances are captured (and this code will be reused to support this inheritance in all queries in future versions of SOPHIA). Finally, relations with an inverse are automatically inverted.

Despite these limitations, SOPHIA provides a highly accessible platform for delivering basic KB services via a

DBMS, while shielding the developer from the underlying relational model. It is portable to other database systems because we have used very few Access-specific features, and so KB functionality could be provided by other relational systems. We have demonstrated this with a partial conversion of the KB to the SYBASE SQL Anywhere database, which was then used as a new ODBC datasource for the SOPHIA code. The SQL used to query the database also remains portable to alternate programming languages.

Ideally, SOPHIA would be a complete OKBC server at some compliance level. Although a network model is not dictated by OKBC (thereby making compliance easier), runtime interoperability of KB servers with different models can complicate client applications by requiring client stubs for each network model. Were SOPHIA made OKBC compliant, however, reconfiguration of a client to use SOPHIA stubs would entail only changes in the code for network connection, while frame-access functions remained virtually identical.

SOPHIA could be made to conform to the OKBC network model in two ways. The first would be to move SOPHIA's VBScript routines into a Visual Basic server application that followed the OKBC network model. This would entail writing and debugging code to talk over OKBC's network protocol or wrapping existing code for VB, since the available libraries are provided for C++, Java, and LISP. The second would be to rewrite the SOPHIA code in one of the supported languages.

Full support of OKBC has not been a primary goal because the protocol has remained under active development. The OWeb family of systems have been developed in our lab over the span of a year and a half, during which time OKBC and its predecessor GFP have undergone many changes. We anticipate that OKBC will stabilize, and have therefore modeled our approach after this protocol. Client applications built on SOPHIA will be designed around frame-based information and knowledge queries. They should therefore be more easily portable to OKBC servers (including possible future versions of SOPHIA) than applications built around other information sources.

In summary, we have stressed simplicity and limitation of features in order to provide a graceful evolutionary pathway for application programmers who are familiar with DBMS (who want to "look under the hood" of SOPHIA), but recognize the value of object oriented or semi-structured data in providing information services. To that end, the use of simple SQL and server side scripting serves as a springboard for building queries across structured data. We focus on a URL mechanism for access to the system, which (despite inefficiencies) is also well understood. Finally, we package data using a simple ascii syntax which is accessible to virtually all end application programming languages. These somewhat severe restrictions provide quite general utility. In fact, SOPHIA is serving the Ribosome-2 knowledge base [Altman et al 1997], originally developed in Ontolingua, with approximately 15,000 frames (a total of 130,000 frame-slot-value records in the database). It is also serving MHCWEB a KB of immunologic scientific literature and data, with approximately 4500 frames [Hon et al, 1998]. We have found the following hardware requirements to be suitable: Windows NT 4.0 Workstation, with a 586 processor, 32 MB RAM and 100MB hard disk (for the swap file). At a minimum, a 486 computer with Windows95 and 16 MB RAM could be used as a web server. We intend to develop the system further to support limited axioms, implement more closely the core OKBC functions, and release the KB Browser to be a functional ontology editor.

Acknowledgments

This work is supported by NIH LM-05652, LM-06422, NSF DBI-9600637 and a grant from IBM. We thank James Rice for useful discussions and code fragments.

References

Altman, R. B., Abernethy, N. F., & Chen, R. O., Standardized Representations of the Literature: Combining Diverse Sources of Ribosomal Data. Fifth International Conference on Intelligent Systems for Molecular Biology, Halkidiki, Greece, 15-24. AAAI Press, 1997.

Chen, R. O., Felciano, R., & Altman, R. B. RiboWeb: Linking Structural Computations to a Knowledge Base of Published Experimental Data. Fifth International Conference on Intelligent Systems for Molecular Biology, Halkidiki, Greece, 84-87. AAAI Press, 1997.

Bell, G., Parisi, A., Pesce, M.. The Virtual Reality Modeling Language Version 1.0 Specification, http://www.vrml.org/VRML1.0/vrml10c.html, VRML Consortium, Inc., 1995

Borgida, A. Description Logics for Querying Databases, Proceedings of the International Workshop on Description Logics - DL-94, Bonn, Germany, 1994.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. P. Open Knowledge Base Connectivity 2.0. Knowledge Systems Laboratory, KSL-98-06, January 1998.

Farquhar, A., Fikes, R., and Rice, J., The Ontolingua Server: A Tool for Collaborative Ontology Construction.

Knowledge Systems Laboratory, KSL-96-26, September 1996.

Finin, T., McKay, D., Fritzson, R., and McEntire, R. The KQML information and knowledge exchange protocol. In Third International Conference on Information and Knowledge Management, 1994.

Genesereth, M., and Fikes, R. 1992. Knowledge interchange format, version 3.0 reference manual. Technical report, Computer Science Department, Stanford University, 1992.

Grosso, W., Gennari, J., Fergerson, R., & Musen, M., When Knowledge Models Collide (How it Happens and What to Do). Eleventh Workshop on Knowledge Acquisition, Modeling and Management, http://ksi.cpsc.ucalgary.ca/KAW98S/grosso/, 1997.

Hon, L., Abernethy, N.F., Brusic, V., Chai, J., and Altman, R.B., MHCWeb: Converting a WWW database into a knowledge-based collaborative environment, Stanford Medical Informatics (SMI) Technical Report SMI-98-0724, March, 1998.

Karp, P. D., Chaudhri, V. K., and Paley, S. M., "A Collaborative Environment for Authoring Large Knowledge Bases, http://www.ai.sri.com/cgi-bin/pubs/list_document_object.pl?doc_uri=/pubs/papers/Karp9704:Collaborative/, April 1997

Maidak, B.L., Olsen, G.J., Larsen, N., Overbeek, R., McCaughey, M.J., and Woese, C.R. The Ribosomal Database Project (RDP). Nucleic Acids Research, v. 24: p. 82-85, 1996.

Musen, M. A., Tu, S. W., Eriksson, H., Gennari, J. H., & Puerta, A. R., PROTEGE-II: An Environment for Reusable Problem-Solving Methods and Domain Ontologies. International Joint Conference on Artificial Intelligence, Chambery, Savoie, France, 1993.

Norrie M.C., Reimer, U., Lippuner, P., Rys, M., Schek, H.-J. , Frames, Objects and Relations: Three Semantic Levels for Knowledge Base Systems, in Knowledge Representation Meets Databases (KRDB '94), 1994.

Patel-Schneider, P., Swartout, B., Description Logic Specification from the KRSS Effort, http://krr.irst.itc.it:1024/scripts/retrieve/dl/dl98/comparison/krss-final.ps.gz, January 1992.

L. Spector, B. Andersen, J. Hendler, B. Kettler, E. Schwartzman, C. Woods, M. Evett. Knowledge Representation in PARKA - Part 2: Experiments Analysis, and Enhancements, Institute for Systems Research Technical Report, TR-92-10, http://www.isr.umd.edu/TechReports/ISR/1992/TR_92-10/TR_92-10.phtml, 1992.

Steenhagen, H. J., Apers P. M. G., Blanken H. M. , Rolf A. de By: From Nested-Loop to Join Queries in OODB. VLDB 1994, 618-629, 1994.