The Architecture for Semantic Data Access to Heterogeneous Information Sources

**Naphtali Rishe, Alexander Vaschillo, Dmitry Vasilevsky,
Artyom Shaposhnikov, Shu-Ching Chen**

**High-performance Database Research Center
School of Computer Science
Florida International University, University Park, Miami, FL 33199
(305)348-1706, Fax (305)348-1705,
{rishen,avasch01,dvasil01,shaposhn,chens}@cs.fiu.edu,
http://hpdrc.cs.fiu.edu**

Abstract[1]

This paper presents the analysis and high-level design of a system that will allow uniform access to heterogeneous data sources such as semantic databases, relational databases, web sites, ASCII files, and others via a common query interface. This interface is based on the Semantic Database Model, which is superior to other data models in expressive power and ease of use, and which allows us to represent a unified semantic schema of all the combined sources — the United Semantic Schema. Interoperability with commonly used tools is supported via ODBC. A particular application of the system to provide an ODBC interface to the WWW as a data source is discussed.

Keywords: heterogeneous databases, semantic databases, interoperability, WWW, data models, query execution.

1. Introduction

Hundreds of different Database Management Systems are used to store data today. Most of them have different data formats that are not interchangeable. In order for applications to interact with different database systems, the ODBC standard was introduced. It allows arbitrary applications to access data from any database without care as to the specific DBMS and data format used to store the data.

We can look at the World Wide Web as a global database containing terabytes of useful information that is distributed all over the world. Such a database contains many different data sources, but the only standard interface to access these data sources is HTML, which is basically a GUI data representation format. While this is fine for presenting data on the user's screen, it is troublesome for a computer program to extract the data and process it.

Some computer-friendly interface should be introduced for computer programs to access the data from this global database, the Internet. The best candidate for such an interface is currently ODBC, since there are many currently existing application programs that "speak" ODBC. If we could hide the Internet behind an ODBC interface, all of those applications would be able to work with the Internet as if it were a local database.

Imagine opening your favorite data processing program, whether it is Microsoft Office, Crystal Reports, or some financial analysis software package, and being able to access and analyze World Wide Web data with it. Would you like Microsoft Excel to draw you a 3D graphic of the current state of your stock portfolio by taking the latest prices from the Internet? Would you like Microsoft Access to find the cheapest ticket to Moscow for you, or build a table of prices depending on the day of week? All of this could be done if only the Internet could speak ODBC — if we had an ODBC driver for the global WWW database.

On the other hand, users already keep their important data in their own databases; some users even have several different databases in use. On many occasions, a user would want to automatically join data from his local databases with data from the Internet. As an example, a user may have a list of employees and the dates and destinations of their business trips for the next month kept in an Oracle database. He could run a query that would find the cheapest airplane tickets for all of them. The software implementing the architecture presented in this paper would run a query on an Oracle database, use its results to make queries to an Internet site for airline reservations and present the user with a table which would have all his original data as well as itinerary data such as the flight number, time, price, and totals for the month. The user would receive all of these results into the ODBC-compatible application program of his choice.

Imagine a more complex query like "How many and which stocks from my portfolio should I sell to generate enough money to buy a ticket to an island in the Pacific Ocean?" Our software would take the information about user's portfolio from wherever he keeps it (a local

[1] This research was supported in part by NASA (under grants NAGW-4080, NAG5-5095, NAS5-97222, and NAG5-6830) and NSF (CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409).

ODBC database or one of the internet services), find the current prices of each stock and trends in their prices from the Internet, choose a stock to sell according to the criteria the user specifies, go to a geographic information Internet site to find all of the islands in the Pacific, join this information with the information from airline reservation sites, and give the user the resulting table in Microsoft Excel (for example) showing him the choices he can afford. All this would be done on-line using currently available WWW information. One just has to write a corresponding query using a favorite query processor.

The High Performance Database Research Center (HPDRC) [6] has implemented a database engine, Sem-ODB, based on the Semantic Binary Model SBM [4]. It has been shown to be very efficient when compared to relational database on a class of applications that utilize the semantic features [7, 9].

Our paper presents the architecture for a system [11] that will provide semantic access to a variety of heterogeneous data sources. A user will view the united database through a semantic schema and access it via the standard interface of the semantic database, which will allow him to run arbitrary queries to the combined data sources. As one of the applications for this system, we will build an ODBC driver for the World Wide Web.

The rest of the paper is organized as follows. Section 2 introduces the architecture of our system. Section 3 describes the data flow within the system. Section 4 shows an algorithm for query execution. Section 5 presents an application of the system.

2. The architecture of the GTOU system

There are many data sources that do not share a common format or common interface. These data sources include semantic databases, relational databases, web sites, ASCII files, and many others. While some of these data sources are structured, some are not. Some do have standard interfaces, some do not; even those that have them often use different standards.

There are many applications that present data to the user. These applications must extract data from the data sources through some interfaces. The most common data retrieval interface supported by Windows applications is now ODBC.

It would appear to be beneficial to have some common interface or several interfaces between applications and data sources as well as between data sources. We introduce a module named GTOU, which will provide a query interface to the application level and which will introduce a standard for this interface. This interface is powerful enough to satisfy the requirements of the application level, is easily computer-readable (without parsing), and allows convenient transformations to be done by a computer. It is semantic in nature, so an application is able to use the semantic features of the underlying database (if they exist), and it is native to Sem-ODB (an implementation of the SBM developed at HPDRC). Implementing an ODBC driver for this interface will provide automatic compatibility with the vast world of Windows ODBC applications.

The same GTOU Module would be used to communicate with data sources via some interface; we propose to use the same interface for all data sources. This interface is native to Sem-ODB so communication with semantic data sources is efficient and does not lose or hide any Semantic features. The interface, however, is independent of the type of data source, which allows us to make all the algorithms within the Module data source independent and application independent. We will try to shift most of the algorithms to this level (inside the Module) in order to implement just one algorithm which can be used for any data source and for any application.

Since the interface between the Module and all the data sources is the same, and not all of the data sources support a common interface, the idea is to create a wrapper around every single data source that will provide a common interface for all data sources. The examples of such wrappers are the WWW Wrapper for WWW data sources and the Semantic Wrapper for relational databases [10]. Once a Wrapper is implemented for a data source, the data source may become part of the grand picture. It can be used by the GTOU Module and thus interoperate with other subsystems. All the algorithms developed inside the module are automatically applicable to this data source.

The GTOU Module must have an internal data structure to represent queries and their results. We have developed a flexible structure named AVDV [8] for this purpose.

The GTOU Module will contain a united schema of all data sources used by the Module and will represent them via a semantic schema, the United Semantic Schema of a universe, where universe is defined as the domain covered by the union of the domains covered by individual data sources.

The algorithms inside the GTOU Module should be able to:

- accept a query from the application level interface,
- do some transformations on the query,
- decide which data sources may be used to obtain the results of a query,
- break up a query into subqueries which can be executed by each data source wrapper (in the case of WWW, for example, the task of determining which query can be executed is far from obvious),
- represent the subqueries in such a way that they can be executed by each data source wrapper,
- understand the order of the subqueries' execution and the possibilities of their parallelization,

- pass the subqueries to the corresponding data source wrappers either in parallel or in the order dictated by the query,
- obtain results from the data sources, including the possibility of accepting partial results in order to optimize the execution by using pipelining,
- feed the obtained results into the other subqueries as required and execute those,
- obtain all final results from all data source wrappers,
- perform the necessary conversions and filtering on the results such as joins between data sources, filtering out extra data returned by data sources that can not do filtering themselves (ex: WWW Wrapper), and
- provide the application that posed the query with an application level interface so it can obtain the results.

The algorithms within the GTOU Module are sufficient to support a distributed/parallel Semantic database. Since the United Semantic Schema may represent several semantic databases, and since we have already described a general algorithm that can execute a query posed to this United Schema by using data from several data sources (in this case semantic databases) and merge the results, the distributed database can be considered a sub-case of our system.

The algorithms within the GTOU Module are sufficient to support a Heterogeneous Database based on a semantic view. We will use the same arguments that we did for the distributed database, considering that the Wrappers for relational data sources and WWW data sources that represent those data sources as semantic databases are implemented. Additional wrappers for data sources not included above can also be implemented as long as the structure of the interface provided by such wrappers is flexible enough; our structure is flexible enough to allow this to happen [8].

The algorithms within the GTOU Module are sufficient to support userviews in semantic databases via the semantic wrapper for the current semantic database (which currently supports only elementary queries). By using the GTOU module, we extend the Semantic interface from elementary queries to complex queries, allow for general algorithms of optimization of these complex queries [2], and implement userviews as virtual categories and relations defined as complex queries on top of the current elementary Semantic queries with the help of the AVDV structure [8].

3. Data Flow in the System

In this section, we will define the data flow in the system. In Figure 1, white rectangles denote modules, gray rectangles denote data structures, solid arrows denote interfaces, and dotted arrows denote modules that are subclasses of other modules. The GTOU Module includes Analyzer and Results Compiler modules. The main execution loop for a query is from Analyzer to Semantic Wrappers to Result Compiler and back to Analyzer. The AVDV structure is used as an intermediate interface structure here. Once the final results of a query are obtained, they are returned to an application via the Results Iterator Module.

The schema compiler is a module that unites several semantic schemas provided by data sources into the United Semantic Schema of the universe. The Query Repository Structure is a knowledge base containing information about the types of queries a data source is able to perform. ODBC can be built on top of SQL server.

The following is a brief definition of every module, data structure and interface with an explanation of their part in the data flow inside the system.

- **ANALYZER** — module (A complex module which takes a query to be executed, analyzes it with the help of the United Semantic Schema of the universe and the repository of queries. Understands how to break a query into sub-queries suitable for execution on data sources. Parallelizes, optimizes, and pipelines the query execution.)
- **SEMANTIC-WRAPPER** — module (A module which translates a query from the language of AVDV structures to the native language of the underlying data source (ODBC, HTTP, ...). Converts results of the query into AVDV structure format. [5])
- **RESULTS-COMPILER** — module (Inserts results of sub query into AVDV structure)
- **APPLICATION** — module (An application could be SQL server, WWW query tool, C++ API, Java API or any other application which is able to pose queries in AVDV structure using United Semantic Schema of the universe and to receive results through the result iterator.)
- **AVDV-STRUCTURE** — data structure (This is a data structure representing a query to the system. The query is built based on the United Semantic Schema of the universe.)
- **SUB-AVDV-STRUCTURE** — data structure (A structure similar to the AVDV structure. Represents a sub query that can be fully executed on one corresponding data source. May contain partial results of earlier executed queries.)
- **DATA-SOURCE** — module (Any external data source. Semantic or relational database, WWW, ASCII file...)
- **SUB-AVDV-RESULTS-STRUCTURE** — data structure (Contains the results of query execution by a data source.)
- **RESULT-ITERATOR** — module (A module which provides an iterative interface to the results of a query.)
- **UNITED-SEMANTIC-SCHEMA-STRUCTURE** — data structure (A semantic schema of the universe. It also contains data that allows finding out where a particular category or relation came from. This

will be used to choose the proper data sources to be used for the query.)

- **SCHEMA-COMPILER** — module (Takes several semantic schemas from different data sources and compiles a united semantic schema of the universe. May be automatic or manual.)
- **QUERY-REPOSITORY-STRUCTURE** — data structure (A repository of all types of queries that a data source is able to execute on its schema. A common database is very likely to be able to execute almost any query. But data sources like WWW can execute only a few types of queries to its schema, so a list of such queries together with translation rules [1, 2, 3] is stored here.)

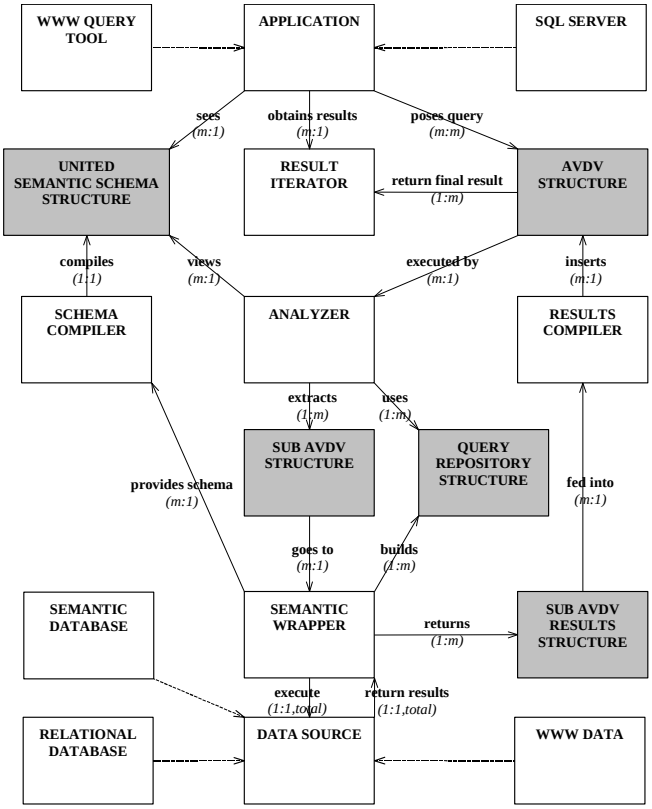

Figure 1. Data flow in the system

- **SQL-SERVER** — one of the modules of type *APPLICATION* (SQL server should be able to translate an SQL statement into an AVDV structure, pass it on to the system, obtain query results from the results iterator, and represent them to the user in relational form.)
- **WWW-QUERY-TOOL** — one of the modules of type *APPLICATION* (A tool which is able to build an AVDV structure through web interface, pass it to the system for execution, obtain results from the results iterator, and pass them to the user usually in the form of HTML or ASCII.)
- **WWW-DATA** — one of the modules of type *DATA-SOURCE* (A WWW data source. It will probably correspond to one HTML entry form (or several correlated forms) and one set of tables returned by a WWW site. Its wrapper should provide the schema compiler with a Semantic schema of the WWW data source (created manually or perhaps automatically), tell the query repository it is only able to execute a few (usually one) types of queries (most probably taking the fields corresponding to the HTML data entry form as input parameters and returning one semantic category corresponding to the output HTML table), translate AVDV structures into HTTP request (or several requests), and translate the HTML (or ASCII) results returned by WWW site into the sub-AVDV results structure (probably using the semantic loader).)
- **SEMANTIC-DATABASE** — one of the modules of type *DATA-SOURCE* (A semantic database containing one data source. AVDV structure is its native interface. Its semantic schema will be passed to the schema compiler to be included in the United Schema and its query repository entry is trivial since it can execute any arbitrary query supported by AVDV structure.)
- **RELATIONAL-DATABASE** — one of the modules of type *DATA-SOURCE* (A relational database. Its wrapper should automatically build a semantic schema corresponding to a relational schema and provide it to the schema compiler, tell the query repository that it can execute an arbitrary query, translate queries from AVDV structure form into ODBC SQL queries and pass them to RDBMS, and obtain results from the ODBC RDBMS and translate them into sub-AVDV results structure form.)
- **execute** — data flow from *SEMANTIC-WRAPPER* to *DATA-SOURCE* (*1:1,total*) (Passes a query in the language native to the data source)
- **return-results** — data flow from *DATA-SOURCE* to *SEMANTIC-WRAPPER* (*1:1,total*) (Returns query results in the form native to the data source (ODBC for relational database, ASCII or HTML for WWW))
- **provides-schema** — data flow from *SEMANTIC-WRAPPER* to *SCHEMA-COMPILER* (*m:1*) (Provides a semantic schema of the data source in order to compile a United Semantic Schema of the world)
- **compiles** — data flow from *SCHEMA-COMPILER* to *UNITED-SEMANTIC-SCHEMA-STRUCTURE* (*1:1*) (A schema compiler generates the United Semantic Schema of the universe.)
- **sees** — data flow from *APPLICATION* to *UNITED-SEMANTIC-SCHEMA-STRUCTURE* (*m:1*) (An application will see the united semantic schema of the universe and will be able to pose queries to it.)
- **poses-query** — data flow from *APPLICATION* to *AVDV-STRUCTURE* (*m:m*) (An application poses a query to the system in terms of AVDV structure.)
- **obtains-results** — data flow from *APPLICATION* to *RESULT-ITERATOR* (*m:1*) (An application will obtain results of the query through an interface of the result iterator.)
- **builds** — data flow from *SEMANTIC-WRAPPER* to *QUERY-REPOSITORY-STRUCTURE*

(*1:m*) (A semantic wrapper must build a repository of all queries it is able to execute on its data source.)

- **uses** — data flow from *ANALYZER* to *QUERY-REPOSITORY-STRUCTURE* (*1:m*) (Analyzer uses the query repositories of different data sources to make sure a data source can execute the query to be passed to it. It must adjust the query to a type that is accepted.)
- **goes-to** — data flow from *SUB-AVDV-STRUCTURE* to *SEMANTIC-WRAPPER* (*m:1*) (A sub query is posed to the data source. The data source guarantees it will be able to answer the query since the query is built according to the repository.)
- **extracts** — data flow from *ANALYZER* to *SUB-AVDV-STRUCTURE* (*1:m*) (Analyzer extracts a sub-query from the big query in such a way that it can be executed by the data source. This sub-query may include some data obtained as a result of some other sub query.)
- **views** — data flow from *ANALYZER* to *UNITED-SEMANTIC-SCHEMA-STRUCTURE* (*m:1*) (Views the United Semantic Schema of the universe to understand the query and distribute it between data sources correctly.)
- **return-final-result** — data flow from *AVDV-STRUCTURE* to *RESULT-ITERATOR* (*1:m*) (When a query is executed and the final results are obtained, they are provided to the application via a result iterator)
- **executed-by** — data flow from *AVDV-STRUCTURE* to *ANALYZER* (*m:1*) (A B query is executed by analyzer. This operation may be performed several times as the B query structure is filled with partial results step by step until the final result of the original query is produced.)
- **returns** — data flow from *SEMANTIC-WRAPPER* to *SUB-AVDV-RESULTS-STRUCTURE* (*1:m*) (Returns a semantic userview filled with data as a result of a sub-query execution.)
- **fed-into** — data flow from *SUB-AVDV-RESULTS-STRUCTURE* to *RESULTS-COMPILER* (*m:1*) (The results of a sub query are fed into the results compiler which will place them into the proper place of the full query, possibly filtering and rearranging the data.)
- **inserts** — data flow from *RESULTS-COMPILER* to *AVDV-STRUCTURE* (*m:1*) (Inserts results of a sub query into the original AVDV structure.)

4. An algorithm for query execution

The AVDV structure, formally defined in [8], is a structure that defines a query for the semantic database and that allows the definition of virtual categories and relations that consist of virtual objects derived from the existing objects and the relationships between them. AVDV is a tuple $\Gamma = (V, P, T)$, where $V = \{a_i\}$ is a set of variables built for existing categories, $P = \{p_i\}$ is a set of predicates restricting those variables, and $T = \{(a_{i_1}, a_{i_2}, .. a_{i_n})_j\}$ is a set of tuples of variables. The tuples in T define virtual categories. Tuples in T are usually tuples of one element. Predicates in P usually represent relations between two variables (Ex: Takes(Student,Course), where Takes is a relation from the database) or comparisons of two concrete values (Ex: Student.age > Professor.age). The goal of query execution is to obtain a semantic database containing the results of the query. This database can then be accessed through the same interface used for Sem-ODB.

The AVDV structure can be graphically represented by a graph where V is the set of vertexes and P is the set of edges. Since in our system an AVDV query is formulated against the United Semantic Schema, the variables in V can belong to categories from different data sources D_1, D_2,..D_n ($v_i \propto D_i$). The algorithm of execution for a query $\Gamma = (V, P, T)$ performs the following actions:

1. Determine which data source D_i supports which variable v_i. This information is taken from the *United Semantic Schema Structure*.
2. Break the graph $\Gamma = (V, P, T)$ into maximal subgraphs
   $\Gamma_i = (V_i, P_i, T_i): \quad V_i \subset V, \forall v \in V_i \Rightarrow v \propto D_i,$
   $P_i \subset P, \forall p(v_1, .. v_n): p \in P_i \Rightarrow v_1, .. v_n \propto D_i.$
3. Execute each Γ_i on the corresponding data source D_i. The independent execution is possible by a Theorem[2] proven in [8]. The same paper shows that this structure can be executed for Sem-ODB. The execution of this structure for relational databases is described in [10]. To execute it for the Web data sources, we can consider all our predicates as one Boolean formula where the predicates are joined with conjunction. Such constraint queries can then be executed with an algorithm similar to [1].
4. Collect the partial results from different data sources into *sub-AVDV results structure* and load them into Sem-ODB.
5. Execute the rest of the predicates as queries within Sem-ODB.

[2] If we can split an AVDV structure $\Gamma = (V, P, T)$ into two AVDV structures $\Gamma_1 = (V_1, P_1, \varnothing)$ and $\Gamma_2 = (V_2, P_2, \varnothing): \quad V = V_1 \bigcup V_2, \quad V_1 \bigcap V_2 = \varnothing,$ $P = P_1 \bigcup P_2 \bigcup Q, \quad P_1 \bigcap P_2 = \varnothing,$ where Q=AreRelated(a,b,R)∈P, a∈V_1, b∈V_2, then Γ_1 and Γ_2 can be executed independently and the final result can be obtained from partial results. This theorem can be easily extended for cases when Q≠AreRelated.

6. At this point the final results are represented in the form of a semantic database and can be accessed through standard interface such as ODBC.

5. ODBC for the World Wide Web

As an application of this architecture, we describe the implementation of an ODBC driver on top of the GTOU Module. This will provide users with ODBC access to the entire united database, which may include some WWW sites as data sources.

This technology is based on our Semantic Object Database Management System, Sem-ODB. The user has to install our Semantic ODBC driver on his computer. A query constructed in his application program will go to our ODBC driver, which will then pass it to the Sem-ODB engine. The Sem-ODB engine can be installed as an Internet server anywhere in the world or on the user's computer if so desired. Sem-ODB will process this query, break it into subqueries, and then send the subqueries to corresponding ODBC databases and/or the Internet. The data retrieved is then processed by our data-cleansing tool and passed to the Sem-ODB loader. The loader extracts the data from the data-source specific format and loads the resulting data into the AVDV result structure. After the data from all sources is loaded, the regular Sem-ODB mechanism will be used to execute the complex query joining the data. The query result is then returned to the ODBC to be passed to the application.

The Sem-ODB database may be used to cache the data retrieved from the Internet or other sources if desired. To do this, we will simply keep the retrieved data for some time. If we do not want to keep even the intermediate query results in the Semantic Database, our Lazy Query technology [9] allows that.

In order to add another data source (ODBC database or WWW site) to our system so that this data source can be accessed together with the others through the same ODBC interface, the following steps should be taken:

- A semantic schema (userview) should be designed for the data source and that schema should be added to the United Semantic Schema of the universe.
- The rules implementing this userview should be programmed if they are not evident [1].
- A control file for the Sem-ODB Loader should be written in a loader's language, which will allow the loader to parse and load data from the output format produced by the data source wrapper (e.g. HTML).

6. Conclusions

In this paper we described a top-level architecture of a system that creates a framework for uniform access to different data sources. It is based on the Semantic Binary Database Model, which has enough expressive power to efficiently represent other data source models and to provide user-friendly query interface to the data. The Sem-ODB database engine implemented by HPDRC handles inter-database relationships and query distribution. An exciting application of this system is providing an ODBC interface to World Wide Web Data. Another important application is a heterogeneous system united under the control of Sem-ODB. An intermediate structure used in this module (AVDV structure [8]) is shown to be flexible enough to support such heterogeneity.

References

[1] C.-C. K. Chang, H. Garcia-Molina, "Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources," Proc. of the ACM SIGMOD International Conference on Management of Data, pp. 335-346 June 1999.

[2] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang, "Optimizing queries across diverse data sources," Proc. of VLDB, pp. 276-285, Aug. 1997.

[3] C. Lee and C.-J. Chen, "Query Optimization in Multidatabases Systems Considering Schema Conflicts," IEEE Trans. on Knowledge and Data Engineering, pp. 941-955, Vol. 9, No. 6, 1997.

[4] N. Rishe, Database Design: the semantic modeling approach, McGraw-Hill, 1992, 528 pp.

[5] M. T. Roth and P. M. Schwarz, "Don't scrap it, wrap it! A wrapper architecture for legacy data sources," Proc. of VLDB, pp. 266-275, August 1997.

[6] N. Rishe, W. Sun, D. Barton, Y. Deng, C. Orji, M. Alexopoulos, L. Loureiro, C. Ordonez, M. Sanchez, A. Shaposhnikov, "Florida International University High Performance Database Research Center," SIGMOD Record, 24, 3, pp. 71-76, 1995.

[7] N. Rishe, A. Vaschillo, D. Vasilevsky, A. Shaposhnikov, S.-C. Chen, "A Benchmarking Technique for DBMS's with Advanced Data Models," Submitted to the ADBIS-DASFAA Symposium on Advances in Databases and Information Systems, Sept. 2000.

[8] N. Rishe, A. Vaschillo, D. Vasilevsky, A. Shaposhnikov, S.-C. Chen. "Query Paradigm for Semantic Databases," Submitted to the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 2000.

[9] A. Shaposhnikov, "Algorithms for Efficient Transaction Management and Consistent Queries in Client-Server Semantic Object-Oriented Parallel Databases," Ph.D. Dissertation, Florida International University.

[10] A. Vaschillo, "A semantic paradigm for intelligent data access," Ph.D. Dissertation, Florida International University.

[11] A. Vaschillo, "The Architecture for GTOU System," FIU SCS Technical Report TR-99-11, 1999.