PROGRAMMABLE ACCESS TO DISTRIBUTED DATA:
DESIGN OF SEMANTIC BRIDGE

G. M. Prabhu
Department of Computer Science
Iowa State University
Ames, IA 50011, USA
prabhu@cs.iastate.edu

Prabhakar Balakrishnan
Chief Technology Officer
Indusasuka Software Limited
Chennai, INDIA

Abstract

Semantic Bridge focuses on providing scalable programmable access to unknown, remote data. Using semantic-based principles with a shared domain model, the most recent Java technologies, and advances in artificial intelligence, we enable data users to write implementation-independent database programs. These abstract programs are then automatically transformed into executables that run within a secure ‘sandbox’ at each data source and return the processed results to the remote user. The data sources need to define the visibility rules for different categories of users and make that data accessible to remote user programs. The data side need not provide any implementations.

Key Words
Programmable Access, Semantic Bridge, Distributed Data

1. Introduction

The computing and communication world has changed dramatically with the advent of the Internet. Mere possibilities for information sharing have become necessities. Advances in digital storage, computing, and communications technologies have resulted in unprecedented opportunities in computer-assisted data-driven applications. Computers interacting directly with other computers to gather information is more advantageous, and the task, even more formidable. For any two computers to interact sensibly they need to have a common vocabulary on the data and processes involved in order to enable any semantic-based communication. This presents several challenges [1]:

(a) Data repositories are large in size, dynamic, and physically distributed. It is neither desirable nor feasible to gather data in a centralized place for analysis. Hence an efficient method is needed to operate across multiple data sources without the need to transmit large amounts of data.

(b) Data sources are heterogeneous in structure (e.g., relational databases, flat files), content (names and types of attributes and relations among attributes used to represent the data), and types of interactions or queries supported. Their implicit ontologies often do not match the needs of their users. In many applications because users often need to examine data in different contexts from different perspectives, there is no single universal ontology that can serve all users, or for that matter, even a single user in every context. Hence methods are needed to extract context-dependent dynamic information based on user-specified ontologies.

(c) Data sources are autonomously owned and operated and the range of operations and types of interactions allowed can be quite diverse. Hence methods are needed to obtain information from these data sources within the operational constraints imposed by the data sources.

Given this, how do we design a framework in which it is possible to develop the kind of applications that can process the information in thousands of databases, even in non-standard ways, and report the results automatically?

The required framework should provide a scalable, maintainable architecture that is extensible, i.e., with support for user-definable data selection, and user-definable operations on the selected data. The notion of scalability that is of concern to us is the number of servers that a single client can access. The notion of maintainability that is of concern to us is that when a change occurs in a data source, it should be sufficient to update just some interface at that data source. It should not be necessary to make changes in all other systems that need to access this data source.

The attempts to integrate distributed data have so far taken two major approaches:

- The Remote Procedure Call mechanism (RPC) can handle complex functionality but the functionality is predefined and not extensible by remote users. Furthermore, all data sources have to agree to implement all the functionality thereby complicating the development of Standards. This is the reason why CORBA (Common Object Request Broker
The data sources are copies of data from several relational or object-oriented database views for integrated access to data from several relational databases. Examples of mediator-based approaches include INDUS [1] and IBM’s Discovery Link [5]. Examples of mediator-based approaches include the TIPMMS project at Stanford University [6], the Ariadne project at the University of Southern California [7], and the NIMBLE system based on research at the University of Washington [8].

To summarize, the approaches taken to address query-based information integration [9] are:

- **Federated databases:** The data sources are independent, but special wrapper-mediators are used to access all the databases collectively, to even execute queries across the databases. Even though this is easy to build, it cannot be used to integrate thousands of databases. It is usually limited to providing a solution within a single Enterprise. It incurs large maintenance costs to deal with schema changes and addition of more data sources. The major maintenance problem arises because when the number of data sources increases, every instance of the Federated Database server should be configured for every data source. A change in one data source will have a rippling effect in all the servers configured to access that source.

- **Data warehouses:** Copies of data from several sources, possibly processed in some way, are stored in a single database called a data warehouse. User updates to the warehouse are generally forbidden but the warehouse is periodically updated. Here again schema changes and the addition of new databases lead to large maintenance costs.

- **Mediators and wrappers:** A mediator is a software component that supports a virtual database, which the user may query as if it were materialized. The mediator stores no data of its own. Rather, it translates a user’s query into one or more queries to its sources, synthesizes the responses from the responses to those queries, and returns the answer to the user [10]. The queries to the data sources are sent to wrappers that are already constructed to provide the necessary functionality of the queries. In the mediator-based approach, data can be accessed from any stand-alone system thereby obviating the need for a Federated approach but the integration is not scalable. The number of round trips needed and the fact that executions are at the single local machine rather than in parallel at multiple target servers makes it less scalable.

Given the nature of the problem, and that of the needed solution, the proposed solution advocates the use of user-defined programs in place of user defined queries, thereby combining the advantages of the RPC and Query Interface mechanisms. The objectives of Semantic Bridge discussed in this paper are to:

- Enable the development of abstract, database schema/implementation-neutral programs to meet the specific needs of remote users.
- Enable the development of abstract, database schema/implementation-neutral programs to service standard, universally accepted semantics such as standard Business Reports, without requiring the data sources to provide the implementation.

The above objectives should be accomplished without compromising either the data security, or the system security of the hosts. The main advantages we gain from our design are scalability and maintainability.

The rest of the paper is organized as follows. Section 2 is the body of the paper consisting of the context, the semantic programming interface, denotations for data access, and the design of the semantic programming interface. In Section 3, we discuss the conclusions and possible advantages of this approach.

2. Body of Paper

Context for Semantic Bridge

Semantic Bridge was designed and developed to support programmable access to distributed heterogeneous data, focusing on data retrieval from a large number (thousands, or more) of data sources, raising the issues of scalability, maintainability, and extensibility of these Client side applications, just the same way such issues are raised in typical Server side applications. Note that Semantic Bridge, by focusing only on data retrieval, sidesteps a lot of technical and practical problems.
The assumption so far has been that Server-side code has to be developed and deployed only by the Server-side programming team, that is, it is not a good idea to run code sent by untrustworthy sources on one’s server. However, if one can design a system to allow client-programmers to develop, deploy, and execute data-accessing programs in a secure 'sandbox' at the remote servers, dynamically, this will lead to an enormous increase in data processing capability. This is similar to the approach taken in Mobile Agents technology [11]. However, the Agents use Services provided by Server-side components to access data. This only addresses the efficiency of execution adequately. Some issues of scalability, and most issues of maintainability and extensibility still remain.

Consider a scenario where a Client machine needs to process data from hundreds or thousands of data sources, selected from millions of data sources. The best way to support this is by permitting client-side code-injection, that is, to allow the Client to send her code to execute on remote data sources, rather than have all the millions of data sources implement the logic of the program that the client wants to execute. But the Servers need to be protected against the possibility that the code sent might be malicious, and so the security issues in running such code on the servers have to be dealt with. Unlike conventional programming situations where the programmer who writes the code also implements the security policy of the organization, Client-side code-injection has to be concerned with malicious code. The person who writes the Client-side code cannot be trusted to implement the security policy of the remote host.

Java security architecture and more recently the .Net security architecture have solved this security problem by implementing a code-source based security. Java security architecture was the first major release that addressed this security issue and portions of Java security are used in Semantic Bridge. However, a secure system should be designed with several layers of security. It is not sufficient to rely only on the language features, or the features of the security architecture. The database layer, the transport layer, application layer all have to contribute to the overall security of the system. A digital certificate based authentication system is used. By restricting the validity of user certificates to very short periods, and using a longer validity for the organizations they are associated with, data-sources can manage the permissions given to remote users to access local data on behalf of the certifying organizations that have developed prior working relationships with the data-source. A key feature of this solution is that neither the organization, nor the users need to have any database access permission, such as user name – password, to connect to the database. These features are described in www.semanticbridge.com. The tools for the development of Shared Model classes, and for their deployment, and for the development of applications, and for the deployment of these applications are also described there.

Semantic Programming Interface

The distinctive feature of Semantic Bridge is to provide a virtual interface, as shown in Figure 1, to enable data users to write not just complex queries but, abstract, implementation-independent database programs. Such a programming model can reduce cost, increase efficiency, and enable a large class of users, both within and outside an organization, to take advantage of available information. We make the assumption that write-access to the data sources is not required by most users and therefore do not address the database update problem.

Programs written for a given domain can use one of many possible models for that domain. Examples of domains could be banking, hotels, hospitals, merchandising, etc. It is usually not a difficult task, with appropriate tools, to map one model of a domain to another for the same domain. It is even easier if only some well-designed interface, such as the Semantic Programming Interface, has to be mapped.

An abstraction of a database program may consist of the following parts:
- Input information (with constraints placed on the format)
- Data selection using input information
- Manipulation of selected data
- Output information (in a certain format)

The Semantic Programming Model allows programs to be written on hypothetical Shared Objects using a Semantic Programming Interface (SPI). The design goals of the Semantic Programming Interface are:
- The ability to construct programs using the interface that is as simple and as powerful as possible.
- The Implementation of the SPI by the database administrator should be as easy as possible.
- The SPI should work correctly for all possible programs and all possible data sources.

User programs written using the SPI are developed on the guarantee that the semantics of all relationships in the model will be ensured in all implementations of the Interface. The implementation of the SPI will set up the Semantic Bridge.

The fundamental philosophy behind Semantic Bridge is that Database administrators (DBAs) should be able to define the rules for the set of Tables, rows and fields in databases that should be accessible to a given user (possibly external), and then the user should be able to retrieve that information and process it any way he/she wants. The user should not have any further constraints in the program development – either in the logic of data
retrieval or the processing of the retrieved data. The data retrieved and processed in this manner can be stored locally along with similar data collected from other sources for further analysis. This way, it is possible to automatically gather and analyze information obtained from various sources.

To enable the remote user to run the same abstract program on different data sources, with possibly different data schema requires that the denotations in the abstract program be replaced by the appropriate denotations for the denoted entities at the target system, before runtime.

The knowledge-based approach to this, using Ontology Interchange Language (OIL) [12], applies model theory’s Classification-Projection diagram [13] shown in Figure 2 to the semantics of the Object Oriented design. In theory, the description of the data sources using OIL can be used to replace the denotations in the abstract program. In a typical program however, all the denotations are not explicit. For example, an object that bears a certain relationship with another object may be necessary in a certain operation. But the programmer usually assumes the responsibility to see that such a relationship exists, and there usually is no explicit way to denote the fact that a relationship should hold. The underlying axiom of the Classification-Projection diagram is: If a relation instance is classified by a relation type, then the source instance of the relation instance is classified by the source type of the relation type and (dually) the target instance of the relation instance is classified by the target type of the relation type. In terms of mapping to object based design and databases, this implies the following:

- A table represents state variables of a type.
- The records in the table represent state information of object instances that have been classified to belong to that type.
- The relationships between types are the projections from one type to another which translate to columns in another table.
- The rows contained in the other table represent the instances of that class and the specific rows corresponding to the related entities are the classified instances based on the projected relation.

Also, the relationships can be implemented differently – for example, ‘left’ in one system may be ‘right’ in another – so that it is possible to derive one from the other, and accordingly the operation to produce the equivalent output may be different. If these issues are not addressed correctly, the semantics of the abstract program will be different from the program that executes at the target site. Furthermore, the fact that an indirect relationship exists will have to be recognized, and processed suitably. This requires the knowledge-based application capable of translating programs across multiple schemas to have a vocabulary that is a superset of all target systems.

Denotations for Data Access

Semantic Bridge works with database views instead of tables and focuses only on the Retrieve operation. Object instances of interest are denoted using special tree-based structures. The denotations provide the user programs sufficient power to express complex semantics for data retrieval. As a result, complex queries needed to discover objects of interest can be constructed without direct access to the database Connection, or Statement objects. The fixed semantics of the query building process considerably simplifies the validation routines to prevent SQL injection type attacks.

Another reason that Semantic Bridge works with views is because the users are not all trustworthy and may send malicious code that may try to access restricted data. Views offer a useful security feature at the database layer. There is also another subtle security flaw that can be prevented through views. Consider the situation that a DBA wants to permit a user access to some rows in a table, but not to some fields in some of these rows. An object-based security to restrict access to the permitted row-field combinations can be defeated by performing multiple queries on that field, until the value is discovered. But, if a view is defined to exclude a value (using Case in the view definition), a search on the view will not reveal additional information.

Since security is such a vital aspect for a solution that permits Client-side code-injection, access to controlled resources can only be through a trusted code intermediary. The protected resources include database-related resources such as Connection object, Statement object, Object storing User information for determining data access permissions, etc., and system-related resources such as opening sockets, loading executable machine code, etc. In the access mechanism of these resources, a lot of them have to be public variables because, for example, Connection, Statement, and User information are common to all domains and need to be accessed from several external classes. However, they should not be accessible from the user program classes. This feature is implemented using two user defined ClassLoaders.

Design of Semantic Programming Interface

The principles for the design of the Semantic Programming Interface are an extension of those of the present Object Oriented Model. The extension will have the means of expressing complex denotations in the domain within the framework of the relationships defined. In all programs, the objects to be used should be explicitly identified, before they can be used in any computation. It is this ability, to explicitly specify the complex denotations, using Interface members, and the ability of the program to call appropriate implementations at runtime, that allows one to execute the same abstract
program on different data sources, without changing the meaning of the program.

The fundamental entities in the Semantic Programming Model are objects, object identifiers, properties, methods, relational references and denotations. Relational references are special methods that return object identifiers. Denotations are based on special syntax used to refer to a single identifier, or a set of identifiers having some special relationships/properties. With the help of these denotations, it is possible to refer to objects indirectly, and with more semantic information. During model development, for a given domain, one should identify the computationally meaningful relationships between the objects. Computationally meaningful groupings should also be identified. These are declared as special methods associated with the objects in the model. These methods return one or more object identifiers. These methods and the parameters passed to them, together, are said to denote the objects of interest in that class.

The programs written using this framework require the programmer to denote the objects of interest by specifying the properties and relationships (using member methods and member properties from the Semantic Programming Interface of the Shared Model classes) that must be satisfied by each of the objects participating in a computation. With the help of these, the programmer can easily write code to construct objects having specified properties and relationships. The terms in these programs will be interpreted to represent the terms in the implementation at the time of program execution, using the implementation for the Mapping Interface provided by the DBA.

Note that the denotations used may have algorithmic components; for example, some ad-hoc denotations such as bidders with bids $xxx less than the nth highest bid. This is something that cannot be readily captured in the Classification-Projection diagram [13], and therefore is a shortcoming of the declarative OIL approach. Programmable access provided by Semantic Bridge allows one to define complex algorithmic denotations (through the power of a programming language) that will be accurately translated at all data sources.

The mapping code is used automatically at runtime to construct the SQL queries to gather the data requested in the user programs. By asking the DBA to reason about implied relationships during mapping, we have essentially partitioned the problem between the user and the DBA – the user programs need to handle the processing logic alone, and the DBA will handle the logic and reasoning that ensures proper mapping. Resolving the differences in terminology, interpreting the relationships, figuring out what is required, and determining the means of getting that information, are all hard tasks for pure knowledge based systems. However, mapping is a simple matter for seasoned programmers. The use of expressions or aggregating sub-queries depending on the structure of the underlying data, are all simple tasks for humans. If the version of SQL supports calls to external functions (in a select clause), DBA’s can even use functions to map the fields. If the computation is complex and function calls are not supported, the DBA may consider adding the fields, or even tables to her database. The point that is emphasized here is that the partitioning of the problem greatly simplifies the solution. If the mapping process also has to be automated, the sophistication of the knowledge base, and the reasoning capabilities required of the program, have to be far greater.

2.1 Figures

![Figure 1. The Semantic Programming Model](image)

![Figure 2. The Classification-Projection Diagram](image)
3. Conclusion

Semantic Bridge is based on the Classification-Projection model. Data selection is accomplished by translating each instance of the Classification-Projection Model to a standard Classification-Projection Model. In addition, the programmer may use the power of a programming language to define algorithms for the classification. Data manipulation is accomplished by Java programs that use the vocabulary of the standard Classification-Projection Model. These Java programs run in a secure ‘sandbox’ and use XML for input/output.

The advantages of this solution approach can be summarized as follows:

- A secure host environment is provided to execute client programs with a flexible mechanism to control the data visible to any given user.
- Data sources do not have to provide any implementations of functions for data access.
- Remote users are in full control of the I/O and processing needed on the data sources.
- Since the same program runs on all data sources without change, reusability of code is enhanced.
- Target identification, communications, and protocols routines can be reused, so that the programmer building the remote user applications can concentrate on the task at hand.
- Applications that require data from remote sources to be used in new unanticipated ways can be easily developed and deployed.
- Java Security architecture is used to limit the range of actions possible by the user programs. Instead, the security architecture of the .Net framework could also have been used in conjunction with C# or other .Net languages to achieve all of the above.

Semantic Bridge offers tremendous opportunities to influence the direction of standards development for the next generation of Internet-based data processing applications. As the Federal Government prepares for the massive reorganization of 22 separate departments with 170,000 employees, Semantic Bridge can play a pivotal role in integrated data management and information fusion applications.

For a preview of the tools and a sample multi-domain application using an Integrated Development Environment, developed to implement the Semantic Bridge solution, please visit www.semanticbridge.com.

4. Acknowledgements

The authors would like to acknowledge the support of Gary Hopwood and Niranjan Sankar, and thank the referees for their valuable comments in considerably improving the presentation of the paper.

References


