

**Design of Semantic Bridge: A Generalized Web Service Providing Programmable Access to Distributed Heterogeneous Data**

G. M. Prabhu, Dept. of Computer Science, Iowa State University, Ames, IA [prabhu@cs.iastate.edu](mailto:prabhu@cs.iastate.edu)  
Prabhakar Balakrishnan, Indusasuka Software Limited, Chennai, India, [prabha@ayyanar.com](mailto:prabha@ayyanar.com)

**Abstract**

This paper describes the design of a system called Semantic Bridge that provides programmable access to unknown, remote data. Using semantic-based principles with a shared domain model, the most recent Java technologies, and advances in knowledge discovery, we enable data users to write abstract, implementation-independent database programs using a Semantic Programming Interface (SPI). These programs are 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 define the visibility rules for different categories of users, and only that data is accessible to remote user programs. Apart from defining the visibility, they do not have to provide any implementations. Because the SPI is stable over time, Semantic Bridge can be thought of as a Generalized Web Service where remote users are in full control of their processing needs--*user-defined data selection, user-defined operations* on the selected data, and *user-defined input/output* formats are supported.

**Keywords:** Programmable Access, Semantic Bridge, Distributed Data

**1 Introduction**

In today's Internet-enabled world, distributed computing is an all-encompassing concept, consisting of, but not limited to, metadata, application logic, interfaces, performance management, database integrity, and middleware technology. Large enterprises with potentially autonomous sub-organizations face a pressing requirement - integrating a multitude of applications and data sources within an enterprise, and across enterprises [1, 2]. Applications must interact to access data stored in heterogeneous databases. The distributed computing model has distributed data sources on one side and distributed users of data on the other side. In order to be effective and efficient, there are three types of problems that must be addressed:

- (1) The problems which are solvable if the data and the processing logic are provided entirely by each of the data sources for use by the large set of distributed users.  

This is accomplished today by three mechanisms - Remote Procedure Call (RPC), Agents, or Web Services. These mechanisms are limited by their lack of support for user-definable data selection and user-definable operations on the selected data.
- (2) The problems which are solvable only if the data alone is provided by the data sources, but the processing logic is provided by the distributed users.  

Existing solutions such as Federated databases do not scale well and are useful only over a limited set of data sources.
- (3) The problems which are solvable only if the data processing logic provided, partly by the data sources, *and* partly by the distributed users, work cooperatively through effective, intelligent communication in a secure and scalable environment.

The distributed computing environment of interest to us is the set of distributed databases, along with their administrators and programmers, willing to expose at least a part of their data to at least a small

set of distributed users. The users, together with their programmers and developers are also a part of this environment. This breakdown of the environment helps identify the problems created by any solution for the distributed environment. It suggests for example, that databases of databases and databases of users will have to be managed by the solution. Furthermore, the solution should solve problems not only for the users, but also for the various programmers, developers, and administrators.

Given this complex environment, the approaches taken to solve each of the three classes of problems stated above create their own share of problems, complicated by the very nature of distributed computing. Hence it is not incorrect to say that the proposed solutions, in these cases, are very much a part of the problem. It becomes essential, therefore, within the scope of the problem definition, to identify the criteria for evaluating the solutions as well. We introduce scalability, maintainability, extensibility, and security as measures to evaluate different solution approaches. These four criteria are of importance to different extents to different participants in the distributed-computing environment.

Semantic Bridge addresses problems in classes (1) and (2) by advocating user-defined *programs* in place of user-defined queries, thereby combining the advantages of the RPC and Query Interface mechanisms. 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. By focusing only on data retrieval, Semantic Bridge sidesteps many technical and practical problems.

Semantic Bridge provides mechanisms for the following:

- Communications and interaction with each data source as needed;
- Specification of a database program, expressed in terms of a shared domain model, across multiple heterogeneous and autonomous data sources;
- Transformation of such a program into executables at relevant data sources for extracting the needed information; and
- Combination and presentation of the results in terms of a vocabulary known to the user.

Bridging the syntactic and semantic gap among the individual data sources and the user is the key problem we address in the system. The main advantages gained are scalability and maintainability. 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 rest of the paper is organized as follows. Section 2 illustrates our motivation through a common scenario. Section 3 describes existing approaches and the problems faced by them. In Section 4, we describe the Semantic Programming Interface which is the heart of the semantic programming model. Section 5 contains the architecture of Semantic Bridge and Section 6 consists of the conclusions.

## 2 Motivation

Let's consider the following scenario which highlights some of the issues stated in problems described in Section 1.

“Tom is the Materials manager of a manufacturing company that is practicing Just-In-Time (JIT) processing. He has a list of 100 vendors who supply items needed by his manufacturing plant. A substantial portion of Tom's present daily activities includes collecting information on the status of shipments from his vendors.

Even though all his vendors have fully functional computer systems to answer any queries he might have and though most of their systems are accessible through the Internet, Tom's job is

not easy. On some systems, he can only send requests for information. This will have to be seen by his counterpart in the other company, who will then have to pull the requested information out of his computer system and forward that to Tom. On some of the more advanced systems, Tom can directly visit his vendor's site and work through the maze of screens to pull out the different pieces of information he is looking for.

Even if Tom is able to get accurate information, it is not easy for him to design a single application that is able to analyze relevant data from multiple sources and aggregate them into a decision-making report. The reason is that the formats of the data are so different and it is not easy to derive a good understanding of their content.

Because of his company policy to practice JIT, Tom is always under pressure because inventory levels are low. His daily activity of compiling information on what has been shipped and what has been delayed is crucial. Only based on this information can he make alternative plans for emergency purchases. Sometimes he is lucky enough to get the bad news on shipment delays early, but not always. It all depends on whom he contacts first.

Tom and all his vendors decide to embrace E-commerce and use XML for supporting electronic communication between their business processes. He then expects that his vendors will be able to effortlessly deliver the information he wants. But much to his astonishment, Tom finds out that he is *not* the only customer to his vendors. The vendors grumble to him about designing a specific process for his company as they cannot design a system according to the specifications of *each* of their clients."

An abstract view of Tom's problem is as follows. Each organization in a domain of industries has a conflicting set of goals: to allow external agencies to obtain from its system any kind of information they are permitted to see while preventing them from obtaining any information they are not permitted to see. If an organization publishes its data storage schema so that external agencies can write their own programs and execute them, it makes it possible for users to get whatever information is available. But this leads to data security problems for the organization and a different kind of problem for the external agencies. In the Enterprise domain, external agencies need to gather information of the *same* kind from many different sources. If they have to develop a *different* set of custom programs for each source, it will lead to a maintenance nightmare.

The main challenge for B2B (Business-To-Business) communities is: enablement must be done for *each* business process and with *every* trading partner. The present models of programming require a detailed knowledge of the structure of database tables in order to write programs. What is needed is a programming model that provides functional interfaces to all server applications on the network. Such a programming model will empower *all* the developers in a trading community and allow collaborative sharing of information with competitive advantage.

The XML standard has addressed only part of the user's problem. With XML, a user sends a request to a data source in the form of an XML document. An XML document is an instance of a pre-defined Document Type. The type can be specified using a DTD (Document Type Definition) or a W3C-approved schema language. Parsing a document with the knowledge of its schema yields the values of the data elements, and the relationships between them. The resulting information is then mapped, to the input of a predefined process on the server, specifically written to service requests of that type. This process may produce an output XML document, defined using another schema, which is then returned to the user for further processing. XML therefore, is just a neutral, document-based mechanism for exchanging data, and the relationships between the data elements, in a structured way.

The user can *only* invoke pre-defined functions implemented to handle the request on the server side. Therefore the functions have to be prewritten and should be capable of handling the request associated with each DTD. On the user side, the returned XML document has to be processed with its DTD for integration into the user's system. Hence XML comes into play before data processing begins, or after the results of the data processing have been returned to the user. It is not involved in the processing itself. But the user should not be limited to the range of the "HOWs" that are available at

the data source – the user knows the concepts and terms he is interested in and the relationships between them – so the data retrieval system must be able to construct the “HOW” for his request, even though the data source may not have the specified functionality. The ability to specify the partial/complete logic of data gathering processes in an implementation-neutral manner is the primary motivation for our research.

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-side 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 [10]. 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 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. Portions of Java security and a digital certificate based authentication system are used in Semantic Bridge. A key feature of our 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 and the tools for the development and deployment of Shared Model classes and applications are described in the approved patent [11].

### ***A Prototype problem***

A law enforcement agency wants to trace the present whereabouts of a man aged about 40 and a boy aged about 16. They are suspected to have left the country using forged documents. The agency’s support group has access to several hundred distributed databases through RPC and Query mechanisms. Since the databases cannot guess ahead of time in what manner the documents have been forged, they cannot implement the necessary functionality at all data sources. What is required is a computer application to accomplish the following goals:

- 1) Identify from the immigration records all travelers (leaving the country) aged about 40 with a boy aged about 16, within the desired period.
- 2) Verify the travel documents with the issuing authorities, and look for suspicious features, and forgeries.
- 3) For the leads to be pursued, track the port of disembarkation and the immigration/embarkation records in that country/port to verify if they are still in that country/port. If they have left, trace them to their destination.
- 4) At their destination, a search on the hotel records is now initiated to finally track down the suspects.

The application has to gather information from the immigration records, passport issuing records and hotel records of numerous systems with very different data storage schemas. Requests for information may have to be sent to literally thousands of servers. All this can be accomplished easily and effectively using our approach (visit [www.semanticbridge.com](http://www.semanticbridge.com) for details). Furthermore, if the first attempt is not successful, modifications to the algorithms to select the target sites, and modifications to the operations at the selected target sites will significantly alter the results obtained. Because the

process is fully automated, this iterative approach can be continued until the desired objective is achieved.

### 3 Existing Approaches and the Semantic Bridge Approach

A Database Program can be thought of as consisting of the following parts:

- (1) Information Input (constraints placed on format)
- (2) Data Selection using input information
- (3) Data manipulation on Selected data
- (4) Information Output (in certain format)

#### **Abstraction of Data Selection**

Databases are in structured form based on *static* type definitions. For example, properties associated with type <invoice> are stored in the database. During selection, we want to select a subset of <invoice> based on certain values of the properties, and or certain relationships, or a subset of these based on some algorithm. The selected subset can also be thought of as a temporary type. Note that while parameterizing allows certain degree of flexibility in defining a range of temporary types, the algorithm portion cannot easily be parameterized. Also, the algorithm acts as a classifier for the temporary type.

#### **Programmer Requirements**

Programmers (users) require the ability to define these kinds of temporary types in every database program. Thus the selection criteria are not based on static types, and will define a temporary type (in abstract terms) in every program. A fully flexible execution environment, therefore, requires the ability to define temporary dynamic types, and a matching classification algorithm.

#### **Solution Characteristics**

To ensure covering the full range of dynamic type definitions, we need the expressive power of algorithms to specify and define subtypes and also to check whether an entity is a member of that subtype. After the data-selection stage, the processing may also need a new algorithm that needs to be executed on the selected data. Here again, the current approaches, discussed below, are not suited for this purpose.

#### **Current Approaches**

*Remote Procedure Calls (RPCs):* Algorithms for type definition are *predefined* and given a name. At run time, these named procedures are called. Since dynamic types can only be encoded within program logic, RPCs cannot support dynamic type definitions. The RPC mechanism can handle complex functionality but the functionality is predefined and not extendable by remote users. Furthermore, *all* data sources have to agree to implement *all* the functionality thereby complicating the development of Standards [3]. This is one reason why CORBA (Common Object Request Broker Architecture) and EJB (Enterprise Java Beans) have not had the kind of success that was initially anticipated. Agent-based systems use RPC mechanisms at the host. If an agent program executes on a limited number of hosts, such a method will work but if the agent program needs to be executed at thousands of hosts, then this approach does not scale well [4, 5].

*Federated Databases:* The current Federated Database solution designed to handle this does not scale well. The data sources are independent, but special wrapper-mediators are used to access all the databases collectively, to even execute queries across the databases [6, 7, 8]. 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 has a rippling effect in all the servers configured to access that source.

*Query Interface:* The Query Interface mechanism through which mediators and adapters allow remote users to dynamically extend functionality also has limitations [2, 9]. A single query returns a single resultset and if the data to be processed has many-to-many relationships, the number of round trips can grow very large. Even when a task can be accomplished with a single query, the resultset can be huge compared to an RPC mechanism that would return only the final processed result. Security requirements are hard to implement and hard to standardize and different data sources are required to build middleware that adhere to the same standard without standard tools readily available to facilitate this task. 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 possible 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.

*Web Services:* The publish-discover-bind paradigm of Web Services to accomplish this is easier said than done. Once a set of target servers are identified through a set of extended “yellow pages,” the programmer has to then consult a “green pages” to customize a set of interfaces for working with the functionality provided by different target servers. If the interfaces change as service providers evolve their services, the programmer who wants to use the service has to change the interface. Thus, the ‘take it or leave it’ approach to the implementation of Web services is ultimately counterproductive.

*Ontology Inference Layer (OIL):* Ontology-based approaches also use static models with a predefined set of types and relationships [12]. Inferences are derived from this static metadata, and not suited for logically processing algorithmic definitions of temporary types.

*Agent Technologies:* Agents are mobile objects that are hosted by servers on a network. They are capable of migrating autonomously from node to node and performing computations on remote server resources on behalf of some user. In order to support agents, a host server must have an interface that “receives” their program code and provides them with an execution environment and access to its services. The program code of the agent runs in a secure ‘sand-box’ at the host and execution is guaranteed to conform to the prescribed semantics. In this sense, Semantic Bridge can also be thought of as an agent-based solution. However there are some major differences.

- *State Information:* In addition to receiving an agent’s program code, the host must also receive the execution state of an agent. This is because an agent may decide to move to another host to resume computation. Since the execution state must be preserved on migration to a destination node, this causes challenging obstacles in protection of the agent state information. In Semantic Bridge, the ‘agent program’ is used for information processing, it does not need to migrate from host to host, and state information does not need to be saved.
- *Execution Environment:* The host provides an execution environment to support agents. The Servlet container is used to host the client code in Semantic Bridge. The executable code in both systems runs securely in a sand-box. But the agent code has to deal with more challenges because of write permissions, protection from malicious hosts which may require tamper-proof append-only containers, etc., that are not needed in Semantic Bridge.
- *RPC Mechanism:* Agent-based systems interact with the host using Remote Procedure Call mechanisms at the host. For information-gathering applications, these RPCs have to take care of the data selection logic and the transformation logic. Therefore all hosts need to implement all the possible methods that can be used as RPCs. If the agent programs execute in a limited number of hosts, such a method will work but if the agent program needs to be executed at thousands of hosts, then the data selection logic and the data transformation logic will have to be implemented at each and every host. In Semantic Bridge, the user has to focus only on the data selection logic and the Shared Model takes care of the transformation logic. Therefore with respect to scalability of the solution for a large number of hosts, Semantic Bridge offers a better and more practical solution.

Thus these existing approaches are seriously limited by their lack of support for user definable data-selection, and user definable operations on the selected data.

### Semantic Bridge Approach

Semantic Bridge addresses many of the existing problems by advocating user-defined *programs* in place of user-defined queries, thereby combining the advantages of the RPC and Query Interface mechanisms. 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.

By focusing only on data retrieval, Semantic Bridge sidesteps many technical and practical problems. It offers the ability to express complex, new, temporary types, and dynamic algorithms to classify instances belonging to these types. It also allows the complete description of the operations on the selected data. In short, Semantic Bridge offers the extremely powerful feature of providing programmable access to remote data, and not just access to remote methods, which is a limitation of other approaches.

### 4 Semantic Programming Interface

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 are 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 sets up the Semantic Bridge. Semantic Bridge is based on the Classification-Projection model shown in Figure 1. Data selection is accomplished by translating each instance of the Classification-Projection Model to a standard Classification-Projection Model [13]. 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.

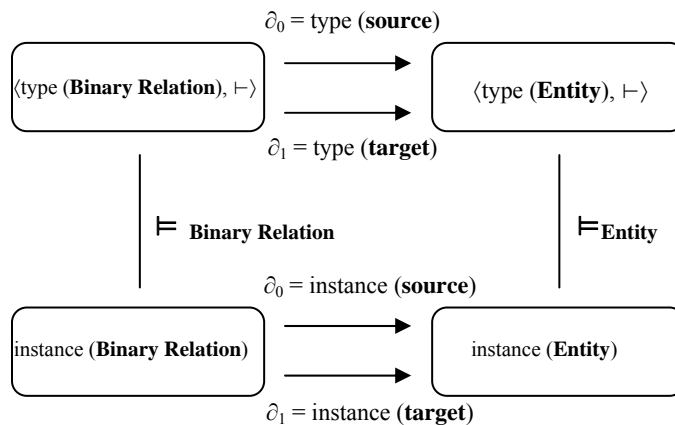


Figure 1. The Classification-Projection Diagram

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.

The fundamental philosophy 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.

### ***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 [11].

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.

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 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.

Thus by using Java with SQL, Java Database Connectivity (JDBC) and the Mapping Interface we can automatically create SQL queries at runtime to gather data requested in the user programs. The advantages of such an approach can be summarized as follows:

- Data sources define the data visibility rules for different categories of users, and make that data accessible to remote user programs.
- Data sources do not have to provide any implementations of functions for data access.
- Remote users are in full control of the processing needed on the data sources.
- Since the same program runs on all data sources without change, reusability of code is enhanced.
- 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 have been used in conjunction with C# to achieve all of the above.

The manner in which Semantic Bridge addresses security, scalability, maintainability, and extensibility is described below.

### **Security**

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. These features are described at [www.semanticbridge.com](http://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. 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.

### **Scalability**

The notion of scalability that is of concern to us is the number of servers that a single client can access. Existing solutions such as Federated databases and wrappers and mediators can access data sources by connecting directly to them (although such a direct connection is unadvisable from a database security point of view). But in these solutions, the executions are at the local machine, giving rise to a large number of possible round trips. In Semantic Bridge, the executions can all occur in parallel at multiple target servers because the executable programs to retrieve data are generated on the remote data source's server, thus making the approach very scalable.

### **Maintainability**

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. However, in a Federated database approach, a change in one data source has a rippling effect in all the servers configured to access that source. This becomes a major maintenance problem when the number of data sources increases because for every data source, every instance of the Federated database server should be reconfigured. Thus existing solutions such as Federated databases are useful for integration only over a limited set of data sources.

### ***Extensibility***

Our notion of extensibility is to permit users to write their own programs to meet their needs. Users should not be limited to the functionality provided by other data sources. This is also possible in Federated databases but they do not provide the scalability and maintainability that is possible through Semantic Bridge.

## **5 Architecture of Semantic Bridge**

The schematic of the overall architecture of Semantic Bridge is shown in Figure 2. It contains the components in the client machine and the external server that runs the client's code. Figure 2 describes the process by which:

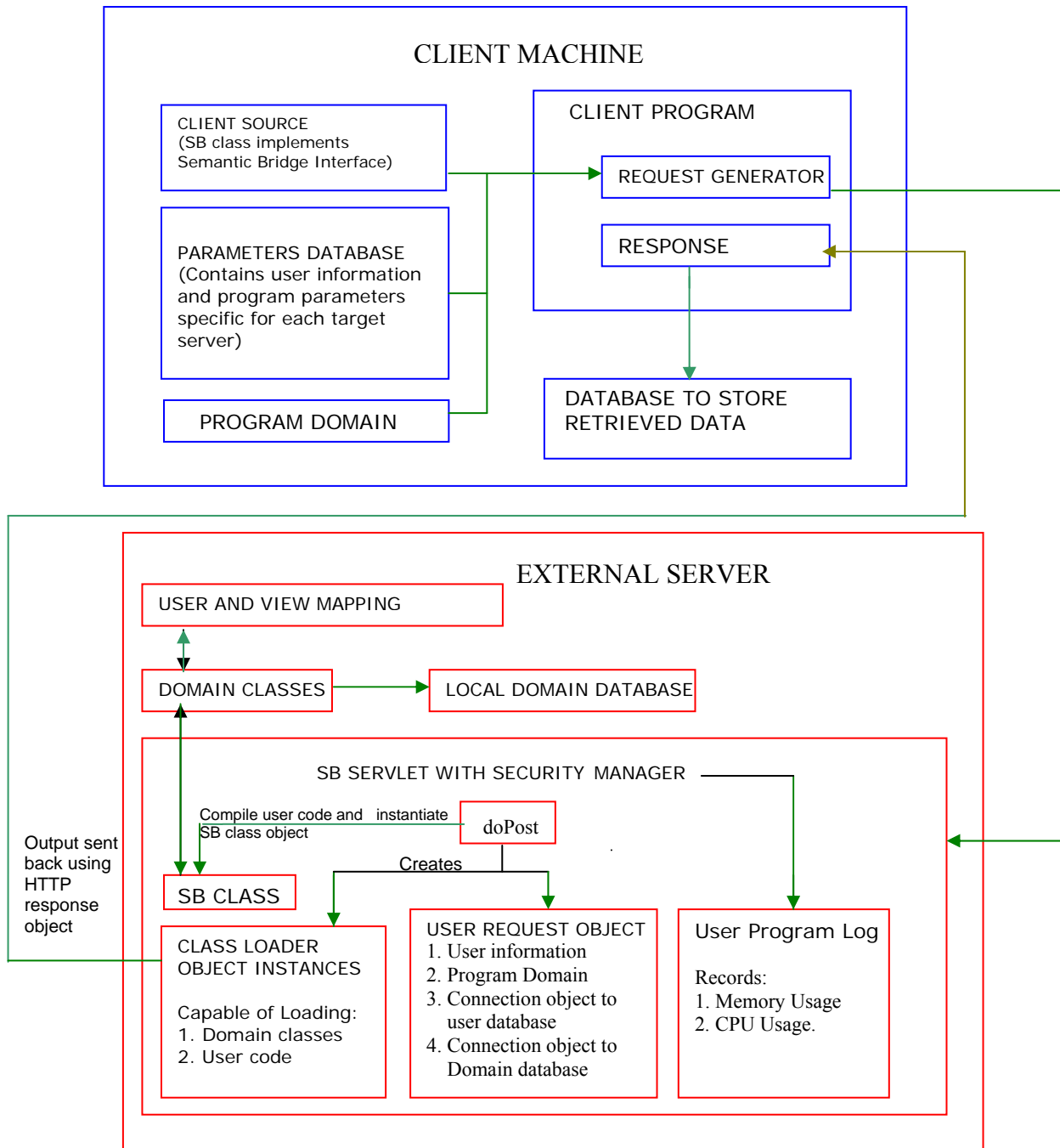
- (1) a user program is sent from a client machine to an external server,
- (2) the user program is compiled and executed on the external server, and
- (3) the results of the execution of the user program on the external server are returned to the client machine.

On the client machine, the Client Program Manager is the controlling element of the new class of application programs written using the Semantic Bridge Programming System. The Client Program Manager manages the connection with external servers, packages the requests, and transfers the requests from the client machine to the external servers.

The Request Generator of the Client Program Manager packages requests from the client machine and sends them to one or more external servers. As described above, the requests include a user source program written using the Semantic Programming Interface, which contains the program that the client wants executed on the remote database, as well as an identification of the program domain for which the source program is written. The external server uses the program domain to permit the incoming source program to access the corresponding domain classes. If a request is to be sent to multiple external servers, a parameters database is preferably used to store certain site-specific information in a database (such as, for example, userID, password, user account number, digital certificates etc.) so that the Client Program Manager can build custom request objects for each external server. The design of the parameters database is straightforward and is not elaborated here.

Requests are preferably sent by the Request Generator using the HTTP/HTTPS protocol. Protocols other than HTTP can also be used. For example, requests may be sent using a mail protocol and the responses can also be received using the mail protocol. Requests sent by the Request Generator are sent to one or more external servers where they are received by the external servers and sent to the Semantic Bridge Servlet (SB Servlet), running in the presence of a Security Manager. The "doPost" method of the SB Servlet is subsequently activated.

The doPost method of the SB Servlet performs the following functions. The user sending the request is identified and authenticated. A special temporary directory is created to store the user class file (e.g., the user source program). The domain for which the client code has been written is identified from the program domain and a class loader capable of loading classes in that domain and the code in the user class file is instantiated. Two database connection objects are assigned to the request from two independent pools of available connections. One database connection is for internal operations – to fetch user specific database access permissions and information to build the "views" appropriate for the current user for the domain database.



**Figure 2. Overall Architecture of Semantic Bridge**

These are the “views” that will be used by the domain classes referenced in the user code. The second connection is used to access the actual data. The newly instantiated class loader is used to create a separate name space and a “user request” object is instantiated in that name space to store the references to the connection objects and user identification in static variables. The access to this object will be controlled through the class loader mechanism – the domain classes will be able to access this object while the user code will not. More details are available at [www.semanticbridge.com](http://www.semanticbridge.com).

## 6 Conclusions

Semantic Bridge is most useful in situations where the information stored in multiple data sources needs to be analyzed in the same way. It significantly reduces the time and skill needed to integrate data sources so that users are freed from the complexities of interacting with heterogeneous, independently developed data sources and can seamlessly gather and process data from multiple sources and in multiple formats. This is made possible by replacing the denotations in the abstract program by the appropriate denotations for the denoted entities at the target system, before runtime. In this respect, Semantic Bridge can be thought of as a Generalized Web Service, with the SPI providing a stable interface over time.

## References

- [1] Collins, S., Navathe, S., and Leo M., "XML Schema mappings for heterogeneous database access," *Information and Software Technology*, Vol 44, Issue 4, pages 251-257, 2002.
- [2] Domenig, R. and Klaus D., "A query based approach for integrating heterogeneous data sources," *Proceedings of the Ninth International Conference on Information and Knowledge Management*, pp. 453-460, 2000.
- [3] Garcia-Molina, H., Ullman, J., and Widom, J., *Database System Implementation*, Prentice Hall, 2001.
- [4] Oliviera, I., Belo, O., and Cunha, J., "Agents working on the integration of heterogeneous information sources in distributed healthcare environments," *Lecture Notes in Computer Science*, Vol 1952, pp. 136-145, 2001.
- [5] Shaw, N., Mian, A., and Yadav, S., "A comprehensive agent-based architecture for intelligent information retrieval in a distributed heterogeneous environment," *Decision Support Systems*, Vol 32, Issue 4, pages 401-415, 2002.
- [6] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., and Widom, J., "The TSIMMIS approach to mediation : Data models and languages," *Journal of Intelligent Information Systems*, Volume 8, No. 2, pp. 117-132, 1997.
- [7] Knoblock, C., Minton, S., Ambite, J., Ashish, N., Mulsea, I., Philpot, A., and Tejada, S., "The Adriane approach to web-based information integration," *International Journal on Cooperative Information Systems*, Volume 10, No. 1, pp. 145-169, 2001.
- [8] Wiederhold, G., and Genesereth, M., "The conceptual basis for mediation services," *IEEE Expert*, Vol. 12, No. 5, pp. 38-47, 1997.
- [9] Kossman, D., "The state of the art in distributed query processing," *ACM Computing Surveys*, Vol 32, Issue 4, pp. 422-469, 2000.
- [10] Tripathi A., "Design of the Ajanta System for Mobile Agent Programming," [www.cs.umn.edu/Ajanta](http://www.cs.umn.edu/Ajanta)
- [11] Balakrishnan, P., "Data Storage Schema Independent Programming For Data Retrieval Using Semantic Bridge," U.S. Patent No. 6,711,579, Issued March 2004.
- [12] The Ontology Interchange Language, [www.ontoknowledge.org/oil](http://www.ontoknowledge.org/oil)
- [13] TheClassification-Projection Model, <http://www.ontologos.org/Simple%20OML/..%5CSimple%20OML%5CFormal%20Model.htm>