

Implementation of Semantic Bridge: Providing Programmable Access to Distributed Heterogeneous Data

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

Abstract

A recent paper [1] described the design of a system called Semantic Bridge which provides programmable access to unknown remote data. This paper describes an implementation of that system in the context of a prototype domain for business process integration. With the use of several standard technologies such as XML, SOAP, Java servlets, and Java connector technologies, Semantic Bridge offers users the ability to write a single program that can run without change on all systems. The results demonstrate a proof of concept for this unique and unprecedented solution for distributed computer systems which can be extended to any domain with information sharing needs.

Keywords: Programmable Access, Implementation of Semantic Bridge, Distributed Data

1 Introduction

In today's Internet-enabled world distributed computing applications must interact to access data stored in heterogeneous databases. 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, 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. The manner in which Semantic Bridge addresses the issues of security, scalability, maintainability, and extensibility of Client side applications and Server side applications are described in [1].

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 [2]. 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 [3].

The implementation of Semantic Bridge is described in the context of a prototype domain for business process integration. Our Integrated Development Environment (IDE) allows companies to extensively reuse the abstract solutions developed by others, and weave in their custom solutions where appropriate. This clean separation of the customizable portion from the standard, at the code level rather than at the specification level, makes it easier to ensure compliance with the standards. It also makes the task a lot easier for the programmers. The result is an environment where computers and humans are brought together as peers to achieve common business goals.

Two customizable servlets, the Semantic Bridge Servlet and the Document Servlet, provide full integration capability with existing systems. The SB Servlet is the interface through which users outside the organization interact with the system. The Document Servlet manages the various document exchanges with trading partners. The internal users interact with the Document Servlet to monitor and control the business activities.

The rest of the paper is organized as follows. Section 2 gives a survey of related work and Section 3 describes the implementation of denotational techniques needed for data access. Section 4 gives an overview of the business process integration domain. In Section 5, we describe the implementation of the Semantic Programming Interface for the prototype business process integration domain. Section 6 contains a detailed description of the resources needed for the implementation and Section 7 consists of the conclusions.

2 Survey of Related Work

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 [9, 10]. 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 [11]. 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 [12]. 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. Table 1 compares existing technologies with Semantic Bridge.

3 Denotations for Data Access

Programs written for a given domain can use one of many possible models for that domain. By domain we mean an area of human activity that uses a computer-based information management system. The key elements and concepts in this human activity are collectively referred to as a domain. Examples of domains could be banking, hotels, hospitals, merchandising, etc. By program we mean abstraction of a database program that consists of the following parts:

- Input information (with constraints placed on the format)
- Data selection using input information
- Data manipulation on 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.

The fundamental philosophy behind the solution of 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.

Technology	Status
XML	Standard for Document Exchange Specification oriented XML-parser helps understand request Data processing not done in document API middleware calls predefined functions Functions must exist at each data source Difficult to extend process functionality
RosettaNet	Standards for common business processes Specification oriented Partner Interface Processes (PIPs) Data processing not done Implementations needed at data sources Difficult to extend process functionality
BizTalk	XML-based Document-centric BizTalk server sends & receives messages BizTalk Framework defines BizTags Microsoft is driving force
Web Services	Publish-Discover-Bind paradigm Uses WSDL, SOAP, ebXML, UDDI Precise specification of service needed Functionality extensible, but interface implementations needed for every service
Semantic Bridge	Document based Shared Model programming domain enables code reuse among partners Handles specifications in XML, ebXML, RosettaNet and OAGIS standards Data processing automatically generated Process functionality easily extensible Open, fluid, opportunistic

Table 1. Technology Comparison

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), applies model theory's Classification-Projection diagram to the semantics of the Object Oriented design [11, 13]. 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. 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. In addition, the application would need to have several other capabilities rivaling human intelligence. Maintaining such a system in an environment where the existing systems are evolving, and new systems are being added has its own share of formidable problems.

The Semantic Bridge approach starts with the appreciation for the fact that, for computations to be meaningful, it is necessary that certain relationships exist between entities participating in the computations. Some computations are meaningful only if the terms involved represent the properties of the same entity. For example, the equation,

$$\text{area of rectangle} = \text{length} \times \text{width},$$

is meaningful only if all the properties (e.g., length and width) refer to the same rectangle – real or hypothetical. Thus, the terms such as length, width, etc., have this implicit association and any replacement of these terms must ensure that the relationship (that they are properties of the same rectangle) still holds after their replacement.

Even though the Object Model allows one to infer that a set of properties belongs to a single entity, there are cases wherein the terms in a computation should come from different entities and these entities should bear certain specific relationships. For example, in the equation,

$$\text{total price} = \text{unit price} \times \text{qty sold},$$

the operands on the right hand side are properties of different entities, and if the entities do not bear a specific relationship, the computation will be meaningless.

In the examples above, the entities participating in a computation are directly represented by a variable name. However, sometimes a procedure has to be used to complete the denotation of an entity involved in a computation. For example, to denote that a series of codons specifying a protein are constructed by reading groups of three monomers in sequence in an RNA strand, from the starting element, can easily be expressed with an algorithm. Similarly, users may easily define an algorithm to denote a 40 year-old man traveling with a 12 year-old boy. A system capable of replacing one set of denotations with another should be able to handle such denotative algorithms as input.

How Semantic Bridge addresses these denotation issues

The principles for the design of the Semantic Programming Interface are an extension of those of the present Object Oriented Model. The extension has 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 and the ability of the program to call appropriate implementations at runtime, which allows one to execute the same abstract program on different data sources, without changing the meaning of the program.

An entity can be denoted directly, or indirectly, using its relationship with other entities. For example, the same individual may be referred to as *X* or *fatherOf (Y)* or the *onlySonOf (Z)* etc. The indirect denotations help establish the identity of the entity and relate the entity to other entities. In other words, given the identity of an entity participating in some computation, the identity of some other entity also participating in the same computation can be denoted using the relationship that is necessary to make the computation meaningful. With denotations of this kind, and the special code to identify entities indirectly referenced in this manner, it is possible to guarantee that the semantics will be preserved for all implementations.

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. If the semantics of the relationship is such that it can only return one possible identifier, it is a unique reference. If a relationship can return multiple identifiers, it is a group reference. 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 model 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 [2]. With the help of these, the programmer can easily write code to construct objects having specified properties and relationships. Once the objects are built, the properties of these objects can be accessed and operated upon. This is how complete programs may be constructed, using only the terms in the hypothetical Shared Model (hypothetical because there may possibly be no data source that matches the Shared Model). 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 for the DBA.

Note that the denotations used may have algorithmic components; for example, some ad-hoc denotations such as \$ *xxx* less than the highest bid, or the *nth* highest bid. This is something that cannot be readily captured in the Classification-Projection diagram, and therefore is a shortcoming of the declarative OIL approach [11, 13]. 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.

To implement the Semantic Programming Interface on a data source for a given domain, the Shared Model for that domain should be mapped to the data storage schema used at the data source. This involves a field level mapping that preserves the implied relationships between entities in the Shared Model. For example, the Shared Model might require that an identifier in a certain field represent the *rightOf (x)* relationship with the identifier 'x' in the same row. If the data source is designed to provide the same information indirectly, by storing the *leftOf (x)* for instance, then the mapping will involve providing the query that returns the identity of the entity for which 'x' will be in the *leftOf (x)* field. Therefore, the mapping process involves associating the Shared Model variables with expressions built from database fields (using Structured Query Language (SQL) syntax, or other any suitable syntax, so that these can be used in subsequent queries). The database fields may come from one or more rows from one or more tables. The list of tables and the join conditions should be explicitly specified and the resultant semantic associations between the database fields/rows should match those in the Shared Model.

The implicit relationships that exist in the Shared Model and the implicit relationships that exist in the database are mapped explicitly by the DBA. This 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 program needs to handle the procedural aspects 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. 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 [1, 2].

4 A Prototype Domain: Business Process Integration

A prototype domain was implemented for business process integration. The design allows companies to extensively reuse the abstract solutions developed by others, and weave in their custom solutions where appropriate. This clean separation of the customizable portion from the standard, at the code level rather than at the specification level, makes it easier to ensure compliance with the standards. It also makes the task a lot easier for the programmers. A typical estimate for set up time with complex customization requirements, and the need to integrate disparate systems, is one or two person-months. The Integrated Development Environment (IDE) developed to facilitate this for the user-side developer enables the easy integration of any custom code developed with the code that is part of the standard. The result is an environment where computers and humans are brought together as peers to achieve common business goals.

Semantic Bridge technology, with two customizable servlets, the SB Servlet and the Document Servlet, provides full integration capability with existing systems. The SB Servlet is the interface through which users outside the organization interact with the system. The Document Servlet manages the various document exchanges with trading partners. The internal users interact with the Document Servlet to monitor and control the business activities.

Design Goals of the Business Process Integration System

- 1) Automatic processing of incoming messages, without need for human intervention, for at least 90% of the messages received.
- 2) Pre-defined set of rules for quick, easy implementation, with ability to modify, or add new rules in a framework that allows the rules to be applied in a flexible manner depending on the context.
- 3) Full integration capability with existing systems.
- 4) Easy set up, even for complex requirements with custom rules and possibly requiring integration of disparate systems. Typical estimate of time involved should be one or two person-months.
- 5) The system should be general enough that it can be used without modification to the core code, in a new domain, by just changing the domain classes, the domain specific set of XML files, and the domain specific data tables.
- 6) The system should be easy and inexpensive to set up, so that even small businesses can afford it, and the vision of a worldwide network of millions of computers, automatically managing complex business tasks, can be realized.

Technologies used in the Solution

- 1) Set of XML documents developed by OAGIS¹ (for the B2B domain), ebXML and RosettaNet for Business Process standards.

¹ Open Applications Group Integration Specifications

- 2) The J-Alphabet soup: Java, JDBC, JMS, JNDI, JAXP, JAXB, JAXM, JDOM, and J2EE connector technology to handle the various system integration issues.
- 3) Cryptography and Digital Signature for security, authentication, non-repudiation etc., as specified in the Standards.
- 4) Semantic Bridge with its Shared Model and Semantic Programming Interface.
- 5) Java Servlets for Internet/Intranet access using SOAP, HTTP and HTTPS protocols.
- 6) Data mining techniques (such as Roughset and Fuzzy Logic) for rule processing.
- 7) Comprehensive, customized IDEs to handle all the development and integration issues.
- 8) XLINK, XSL, XPATH, etc., for XML processing, and for enabling humans and computers to work together to achieve business goals.

Implementation Overview

The implemented system has two customizable servlets – the SB Servlet, and the Document Servlet. The IDE provided allows the behavior of these to be customized, in a flexible framework, for handling various standard requests as well as special requests from external users.

The SB Servlet is the interface through which the users outside the organization interact with the system. This also automatically takes care of encryption and authentication issues (digital signature verification). External users can also use the SB Servlet to send custom programs to execute on any data visible to them. Alternatively, they can submit their requests as 'standard' documents used for information exchange in the given domain (for example, many documents have been defined by OAGIS for B2B). The requests always generate messages to internal users, and may also generate new response documents. The Document Servlet maintains relevant information regarding these activities, and can answer queries by the users. The internal users interact with this servlet to get all their messages and the associated documents. The internal users are automatically notified of the actions taken by the system in response to a request. In addition, while processing a request, the notifications could include messages requiring human assistance for certain pre-defined combinations of conditions.

Figure 1 shows the distributed architecture of the prototype system. The JMS server handles the messaging aspects of the communications. The SB servlet handles authentication, authorization, encryption, decryption, and user request processing. It uses an SSL link with the Document servlet to store and retrieve documents.

The code executing on the servlets includes the code developed specifically to meet the needs of the organization. However, this portion of the code is expected to be less than 10% of the code than would otherwise be necessary. External users will also have the capability to execute their custom code on the data exposed to them by their trading partners, in a secure environment defined by their trading partners.

The users within the organization will have the capability to read, review, act on, and even generate documents. Newly generated documents will be stored by the document servlet, and a copy will be forwarded to the SB servlet for encryption and forwarding to the JMS server. The SB servlet will create SOAP envelopes using JAXM, and then pass the envelopes onto the JMS server for delivery.

5 Implementation of Semantic Bridge for Business Process Integration

The User-side Developer IDE

The IDE to set up the two servlets consists of two workspaces. A workspace is nothing but a set of related windows. Based on the choices made in one window, the information/options in the other windows are altered and displayed.

The IDE allows the user to view the effects of her actions immediately. The IDE has been designed to present the information to the user in lists, tables, graphs, or in a browser, or in a source code editor, as appropriate. Altered source code may be readily compiled and tested.

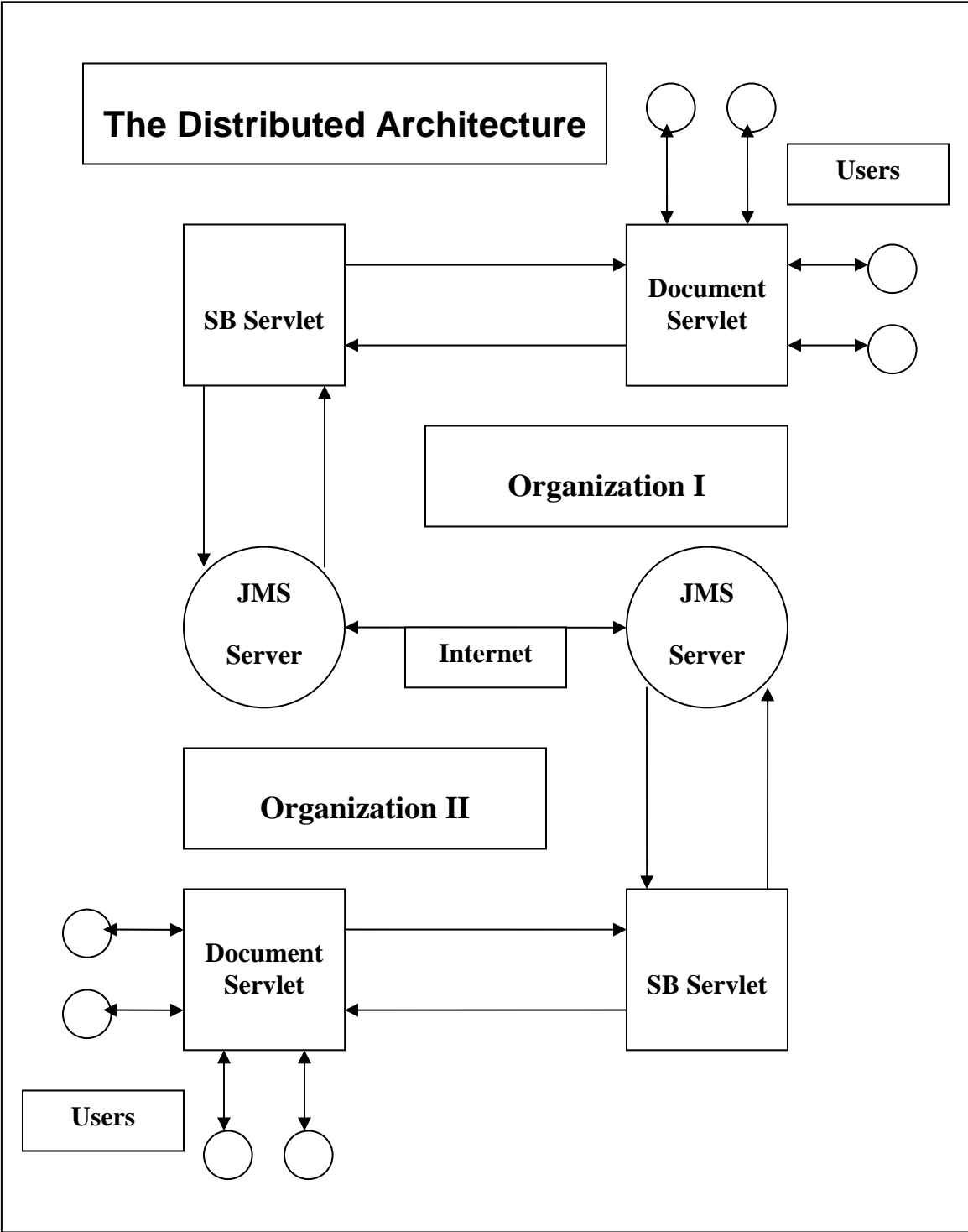


Figure 1. The Distributed Architecture of the Prototype System

SB Mapping Workspace

The SB Mapping workspace helps configure the SB Servlet to receive requests (with custom programs) from external users. This involves mapping the Shared Model classes to the local database tables, and setting the data view permissions for the external users. The mapping of classes is done independent of the permissions given to the users. The mapping should follow the guidelines of Semantic Bridge with respect to units and relationships inherent in the Shared Model.

Visibility permissions are assigned by associating permissions with roles. Special Boolean expressions (with special place-holders to get the user attributes for specific restrictions) are used in SQL syntax to restrict both row-level and field-level visibility. External users are assigned roles associated with the organizations they represent in each role. This allows for the visibility to be restricted by the organizations the users represent, and by the users' roles in the respective organizations. All the data that can be seen by the selected user can be viewed directly in the IDE.

SB XML Workspace

All OAGIS documents use a noun-verb combination to identify the semantics of the processing needed on the document content. The noun indicates the object the service is to be performed on and the verb is the actual service to be performed. The request can be for a product, service, action, or information. The document mechanism could also be used for notification, or for updating some information. All of the above possibilities are collectively referred to by the generic term 'request.' The processing involves the following steps:

- Data gathering – from the current document (and previous, related exchanges), and from the local database.
- Ensuring that the terms of the Trading Partner Agreements (TPA) are met.
- Understanding each of the requests and the terms of the requests, and understanding the ability/willingness to service the request.
- Understanding the value and the range of acceptable variations of the service.
- Making a decision based on the above. The possible decisions are to accept, reject, generate error messages, notify close alternatives, or request human assistance.
- Recording the decision/action taken, and notifying the appropriate human users of the developments.
- Generating response documents.
- Maintaining a complete record of document exchanges.

The above system should integrate easily with any existing MIS system so that it is possible to use all the relevant functionality already available. Also, the messages generated in the existing system should easily and automatically flow into the SB XML system.

The description given so far was for the case of handling incoming requests. In the case of outgoing requests, the logic of the processing is different, and this can also be defined using the IDE. Now our concern is with the process to handle the response to a request initiated by us. For example, a counter offer in response to our offer will have to be evaluated, and decisions may have to be taken. The context for this has to be given by the process that originated the request.

From the analysis of the requirements, it is possible to arrive at a general solution that separates all the functionality that does not vary from system to system, and implement that portion using a common framework. For example, gathering data from the documents, or generating standard response documents, verifying digital signatures, etc., are common tasks, which will have to be implemented in every design. Also, pulling data out of local databases will be necessary. Though the semantics of the kind of data that need to be retrieved from the local database will be fully defined for the request, the code needed to do this varies considerably from system to system. This is where the

power of Semantic Bridge comes into play. The semantics can be expressed using the Shared Model, and the mapping process will take care of the conversion automatically.

Like the access to the data that needs to be programmed only once, there are several aspects of the whole system that need to be programmed only once. The trick to making this work is to allow the user-side developer full control over the portions that need customization, and automatically interlace the custom code with the standard code at the appropriate places. The semantic context in which the custom code will execute has to be clearly defined, and the users of the IDE should be clear on the implications of the entries they make in the IDE.

The fully implemented system will have code developed by four teams of application developers:

- 1) The Semantic Bridge Team
- 2) The Shared Model Development Team
- 3) Team that uses the SB IDE for customizing the mapping/Document interfaces
- 4) The Team that developed the existing/legacy code that will be integrated with the SB solution

The system design is carefully worked out to allow the code developed by Semantic Bridge Team, the Shared Model Team and the existing/legacy code in the local MIS to be readily integrated into a final solution that incorporates the code written into the IDE provided. The IDE is designed to make this integration as easy as possible. The framework also provides for the need to enable the intermixing of various combinations of hardware and software platforms. By incorporating some of the latest developments in distributed computing, in the very framework of the underlying architecture, the systems can be integrated with just a set of appropriate standard drivers.

Although the prototype domain chosen is in business process integration, this approach can be used in any domain that will benefit from a standard for information exchange. In the age of the Internet, it is hard to imagine a worthy domain that will not benefit from Semantic Bridge. In a new domain, our system can be made fully operational by changing the domain classes, the domain-specific set of XML files, and the domain-specific tables corresponding to the standard in that domain. A special Integrated Development Environment (IDE) for standards developers will provide the tools and infrastructure necessary to develop the abstract solutions that will be reused in the final customized implementation for any domain that has recognized the need to have a common ontology and the need to share information.

6 Resource Requirements for Implementation of Semantic Bridge

NetBeans is a customizable Integrated Development Environment (IDE) with built-in support for Java. The Semantic Bridge solution for programmable access to remote data is made possible by the code/configuration information provided by various agents in this widely distributed data environment. The customizable feature of NetBeans really comes in handy here because we can provide the various agents the means to package their input to the solution in manner that enables seamless integration with the input from others.

Installable custom modules is the mechanism NetBeans provides to enhance the features of the IDE. Using this feature three custom modules have been developed to give specific functionality associated with the three principal components of the Semantic Bridge solution. One module (Shared Model Module) helps in the development of the Shared Model, and its associated classes. A second module (Mapping Module) helps in mapping the local data to the Shared Model, and to set the data access permission for remote users. The third module (Application Module) helps in building Semantic Bridge applications that can use the Shared Model classes in multiple domains to access various remote data sources.

The Semantic Bridge implementation also has three types of servlets – Login Servlet, Application Servlet, and Target Servlet. The first two types are deployed within each organization for their internal use. The third type is deployed by each organization to expose their data to remote users. The applications an organization wants to run on remote databases will be deployed on the machine with

the Application Servlet. The Login Servlet manages the access the internal users have to these applications through an Access Control List. The Application Servlet contacts remote Target Servlets on behalf of the users, under the authority given by the organization. The Target Servlets will deny access to remote users, unless a working relationship has already been set up with the organization represented by the remote user.

The Shared Model Module helps with the development of Shared Model classes. It also allows the developer of the model to recommend a role-based data security schema for easy adoption during mapping at various target databases. The schema for the Target Server Search database is also designed using this module. The classes can be packaged for easy deployment on Target Servers, using the code provided in the IDE.

The Application Module can be used to develop custom applications to meet the requirements of a given organization. These applications are meant to be deployed on Applications Servers, so that authorized users may remotely call these through a Browser interface. The request from the users will cause code to be dispatched to remote Target Servers meeting user specified characteristics. The results of the execution at various Targets will be compiled in the Application server according to the specification in the application code, and the final result of the complete execution will be sent to the user's Browser.

The applications are not limited to a single domain. However, the code is expected to follow the framework specified by the Semantic Programming Interface. The IDE is designed to ensure compliance to these specifications. More details about these modules are available at www.semanticbridge.com.

Hardware & Software on the Data Source side:

- A Fast machine with a security enabled Servlet container (such as Tomcat), and the ability to connect to the database using a JDBC driver or a JDBC/ODBC bridge.
- This machine should be accessible through the Internet using a permanent IP address which should be registered with SB's domain server database.
- Program to register IP address
- SB software, and license for some software used in development of SB tools
- Specific Domain classes
- Properly configured Security Policy file
- JDK 1.3 or above

Hardware & Software on the Remote user side:

- Fast machine with Windows 98 or later
- Internet access
- SB IDE for developers, and SB applications for users, and some licenses
- Access to a local database for storing retrieved information, and information on target servers
- Program to update local database of target servers from the SB's domain server database

Resources to be provided by the Client, for Domain Development:

- Domain Expert, Database Expert, Security Expert
 - To identify the domain classes, properties, and relationships of interest to remote users, without complicating the mapping process.
 - To identify the Standard functionality to be included in these classes.
 - To define Standard Units for measurable quantities.
 - To identify the range of possible values for certain properties.
 - To define a set of recommended roles by which the remote users can be classified, and the recommended data visibility rules for each role.
 - To define the structure of the Target Server database for this domain. This should enable searching for different classifications of target servers that could be of interest to remote users.
 - To define security protocols needed.

- To develop the documentation on the Shared Model classes, to be used by the programmers on the remote users side.
- To develop the documentation on the recommended data security aspects.
- To develop test programs to run on systems requesting Registration.
- To develop authentication mechanisms for Registration, and updating Registration information.
- Access to some domain data, using a Servlet, for testing

Resources to be provided by Client, for Trial installation:

- A set of diverse data sources with different schemas (at least five)
- All the hardware and software as noted above for each data source
- DBA's involvement in mapping the local schema to the Standard, using the SB IDE, for each data source
- Registering process for trial installation to be completed
- Client-side user input for sample applications
- Testing of sample applications
- Documentation for trial installation
- Documentation for sample applications

In addition, resources have to be provided by the client for training DBAs and application programmers.

7 Conclusions

The capability provided by Semantic Bridge is revolutionary compared to present database wrapper and federated solutions which provide fixed, static functionality by limiting users to calling named procedures on remote systems. Our implementation of a prototype system for business process integration offers proof of concept for the design described in [1]. However, applications are not limited to a single domain. Understanding the requirements will help determine how SB can be deployed in a given domain. Semantic Bridge is based on the Classification-Projection model. Each vertical domain is built on the basis of a reasonable model. The Shared model that SB builds for each vertical domain is also a reasonable model. The mapping tool provided to the DBA establishes a mapping between two such reasonable models. Depending on the requirements of the Shared Model, we may need to have separate licensing arrangements for each vertical domain. If the use of SB technology is across horizontal domains, then a different model will have to be built that connects all the interacting domains. Thus, by expending a little more effort a whole library of standard and remote-user-defined functions can be executed remotely using Semantic Bridge.

With its use of several standard technologies such as XML, ebXML, RosettaNet, SOAP, OAGIS documents for B2B, Java servlets and Java connector technologies, Semantic Bridge offers users the ability to write a single program that runs *without change* on all systems. What the Internet has done on the hardware side, Semantic Bridge does on the software side. The Internet provides the means to connect islands of locally networked computer systems together. Our simple and elegant solution logically bridges isolated islands of data, making it possible to write applications for the intelligent analysis of shared information in a manner that is not accomplished by today's middleware. A conservative estimate of the reduction in effort needed to set up integration with our software is 90% on the data provider side and 70% on the data user side.

References

[1] Prabhu, G. M., and Balakrishnan, P., "Design of Semantic Bridge: A Generalized Web Service Providing Programmable Access to Distributed Heterogeneous Data," *Journal of Engineering, Computing and Architecture*, **1** (2), 2007.

- [2] Balakrishnan, P., "Data Storage Schema Independent Programming For Data Retrieval Using Semantic Bridge," U.S. Patent No. 6,711,579, Issued March 2004.
- [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*, Vol. **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*, Vol. **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] 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.
- [10] Kossman, D., "The state of the art in distributed query processing," *ACM Computing Surveys*, Vol. **32**, Issue 4, pp. 422-469, 2000.
- [11] The Ontology Interchange Language, www.ontoknowledge.org/oil.
- [12] Tripathi A., "Design of the Ajanta System for Mobile Agent Programming," www.cs.umn.edu/Ajanta
- [13] TheClassification-Projection Model, <http://www.ontologos.org/Simple%20OML/..%5CSimple%20OML%5CFormal%20Model.htm>.