A Reusable Platform for Building Sovereign Information Sharing Applications

Rakesh Agrawal, Dmitri Asonov, Priya Baliga,
Linus Liang, Beate Porst, Ramakrishnan Srikant

IBM Almaden Research Center, San Jose, CA 95120

ABSTRACT

Sovereign information sharing allows autonomous organizations to compute queries across databases in such a manner that nothing but the result is revealed. We present the design of a platform that can be used for building sovereign information sharing applications efficiently and quickly. In addition to the core protocols, the platform incorporates components for resource discovery, schema mapping, authentication, and hardware-assisted performance enhancement. We also describe a national security application we have built using this platform that shows the practicality of sovereign information sharing in virtual organizations.

1. INTRODUCTION

Current information integration techniques for virtual organizations require participants to reveal all data necessary to process a query. Consequently, information sharing is inhibited due to security and privacy concerns. The goal of Sovereign Information Sharing (SIS) technology [8, 9] is to enable the computation of queries across autonomous data sources in such a way that no information other than the query results is revealed.

Examples where this technology may be beneficial include:

- **Homeland Security** For national security, it may be necessary to check if any of the airline passengers are on the suspect list of a federal agency [14]. The agency may use SIS to find the names of only those passengers who are on the suspect list, without obtaining information about all passengers from the airline, or revealing the suspect list to the airline.
- **Health Care** In epidemiological research, a medical researcher may want to know whether there is a correlation between a certain DNA sequence and a reaction to a drug, which may require joining DNA information from a genebank with patient records from a hospital. However, hospitals disclosing patient information could violate privacy protection laws, such as HIPAA [3]. Similarly, gene banks may also have privacy constraints on disclosing the information. Using SIS, we can ensure that only statistics about correlations between adverse reactions and DNA patterns is revealed.

We present a SIS platform that can be used for building such applications efficiently and quickly. Section 2 gives an overview of the platform. In Section 3 we describe different approaches for schema mapping. We discuss resource discovery issues in Section 4 and authentication mechanisms in Section 5. In Section 6 we describe the SIS prototype in terms of the homeland security scenario described earlier. We discuss techniques for performance improvements in Section 7 and close with a summary in Section 8. This paper refines and extends the ideas presented in [8].

2. ARCHITECTURE OVERVIEW

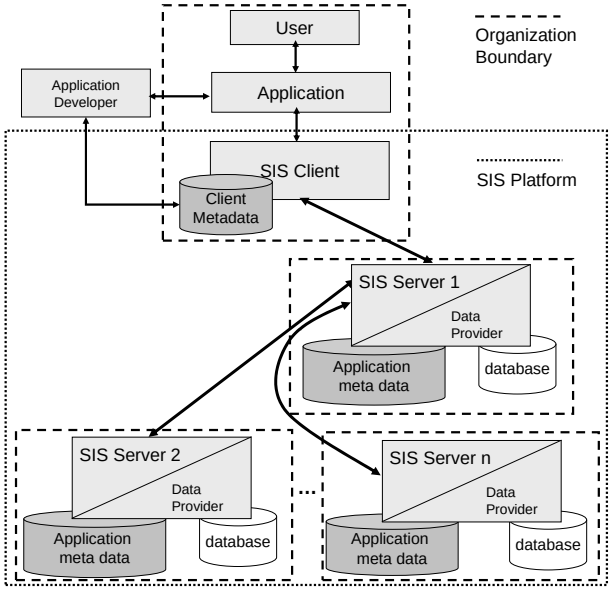

Figure 1: Sovereign Information Sharing architecture.

Each autonomous organization (data provider) participating in sovereign sharing uses the SIS platform to

First Workshop on Databases in Virtual Organizations, June 2004, Paris, France.

offer services for applying operations on their data. Applications are developed by composing these services.

Figure 1 shows the architecture of our SIS platform. It comprises of:

- **SIS Server** The SIS server provides the necessary functionality on the data provider side to enable sovereign information sharing. This includes components for query processing (cryptographic protocols for basic operations), access verification, data encryption, and communication with other data providers. It also binds applications to purposes for a purpose-based access control mechanism [10].

  The server also maintains metadata needed for processing a query issued by an application, including view information to retrieve data from the data provider database, database access information, and context information.

- **SIS Client** The SIS client provides the necessary functionality to map the application schema to data providers' schemas, construct and invoke web service query requests against multiple data providers, and receive web service responses. The SIS client uses the client metadata database to store and retrieve mapping information and data provider access information.

- **SIS Application** An application is a thin layer on top of the SIS client, which invokes the required SIS operations. For example, an application providing epidemiological results for drug reactions may invoke intersection size operations multiple times applying different filter predicates. The application also provides a communication interface to a SIS user and interacts with the SIS client using the SIS client API.

3. SCHEMA MAPPING

The heterogeneity of autonomous organizations makes schema mapping essential for a system providing information sharing. We analyze below the different approaches to schema mapping that can be used for providing a consistent vocabulary for developing applications on our SIS platform.

1. **Federated** Data providers make their metadata information available at some known locations (e.g. by publishing to a UDDI [6]). Any schema mapping and identification of relationships between data of different organizations is the responsibility of the application developer.

   - Pros: This approach requires minimal effort on part of data providers, and it is consistent with the schema publication schemes of federated databases such as DB2 Information Integrator [1].
   - Cons: The schema mapping work is entrusted to the application developer. Identifying relationships can become difficult if the schema names are not self-descriptive or are ambiguous.

2. **Global** This approach assumes that a standard global schema for the domain of interest exists (e.g. RosettaNet [2]), which is used by the data providers to publish the logical view of their metadata.

   - Pros: Application developers can use the standard domain schema to develop client applications and do not have to deal with schema mappings.
   - Cons: A globally accepted domain schema may not exist for all domains. Also, this approach transfers the work of the schema mapping to the data providers.

3. **Application specific** The schema mappings between an application's schema and the data provider's schema is done by the data providers, separately for each application.

   - Pros: Convenient for application developers when there is no global schema available.
   - Cons: The data providers need to map schemas for all applications and they have to apply all changes made later by application developers for a certain application.

We use the federated approach in our implementation because it places minimal burden on data providers and provides them with maximum autonomy and decoupling.

4. RESOURCE DISCOVERY

To build an application on top of the SIS platform, an application developer should be able to find all relevant data providers that offer the required data service, and to infer details about the format and extent of information that the providers are willing to share. We use the UDDI registry for this purpose. UDDI is a public registry that contains cataloged information in an easy-to-search format.

We use two UDDI elements: Business Entities to represent data providers and Business Services to represent services offered by data providers. When a data provider decides to share certain parts of its database, a Business Entity is created and published on the UDDI. The Business Entity contains information such as the name of the organization, its contact information, and a brief description of the organization.

Business Entities provide developers a way to contact data providers, and negotiate access to data. Whether a particular piece of information will be shared or not depends in large part on the purpose for which the data will be used. For example, in an application designed to identify potential candidates for an experimental drug, a gene bank may share the names of those clients who have opted-in for experimental drug testing. However, in an application designed to compute statistics about the correlation between DNA patterns and diseases, the gene bank will use a different protocol [9], thereby ensuring that only statistics and no personally identifiable information is revealed.

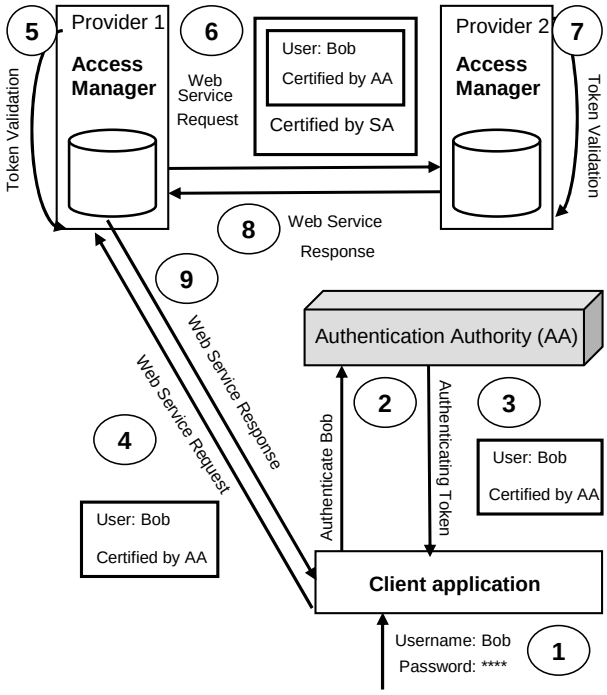

Figure 2: Authentication architecture.

Each Business Entity can contain one or more Business Services. Each service contains an XML description of the schema of the data the provider is willing to share, including table names, column names, and constraints.

A developer wanting to create, say, an epidemiological research application can search the UDDI for all hospitals and all gene banks that are offering data sharing services. She can then look up the XML schema description to ascertain whether the provider is willing to share the pertinent information. Finally, she can contact the providers of interest (using the contact information from the Business Entities) and negotiate the use of their data for the application.

5. AUTHENTICATION

Designing a single sign-on for our architecture was challenging because of two reasons. Firstly, after the initial call to the first data provider, all subsequent calls to other data providers are made by the SIS platform without any user intervention. We therefore need to establish "portable trust" for the user across multiple domains. Secondly, we use web services to perform all communication, but standards for web services security [5] have not yet been widely adopted.

Our solution (see Figure 2) is to establish a central authentication authority (AA) that keeps track of all registered users of the SIS platform. When a user wants to use SIS, she provides her credentials to the application (Step 1), which in turn contacts the authentication authority (Step 2). If the user is valid, the AA returns a digitally signed authentication token (Step 3). The token consists of an assertion of the user's identity, the issuer's identity, the timestamp and validity of the token, and information about the user's authorization group. The token is built in the form of an XML document and digitally signed by the AA to assert authenticity. All communications are encrypted to avoid snooping.

The application includes the authentication token in all subsequent web service requests (Steps 4 and 6). The data providers can verify the user's access privileges based on the attributes of the user specified in the token (Steps 5 and 7). For all web service requests initiated by the SIS platform, a new token is created by embedding the AA's digitally signed token in a master token that is then digitally signed by the sender (Step 6).

6. USAGE SCENARIO

The Transportation Security Administration (TSA) may seek airlines to provide passenger information to test a new counter-terrorism program [14]. In order to comply with TSA's request, airlines would have to share their entire passenger list with the TSA. This would compromise the privacy of their passengers. On the other hand, the SIS platform can enable an airline (AL) to reveal the names of only those passengers who appear on the TSA's suspect list.

We will use the above scenario to illustrate how an application can be built using the SIS platform. This simplified application makes use of the SIS set intersection protocol [9, 12, 13].

6.1 Protocol

Assume tables R and S, belonging to sovereign parties A and B, have one column each. The following protocol will enable A to learn nothing about S but the intersection $R \cap S$ (and $|S|$), while B learns nothing about R (but $|R|$), given a commutative encryption function E:

1. A encrypts table R into $E_A(R)$ and sends it to B.
2. B encrypts table S into $E_B(S)$ and sends it to A.
3. B encrypts table $E_A(R)$ received from A into $E_B(E_A(R))$, and sends pairs of records $\langle E_A(r), E_B(E_A(r))\rangle$ to A, where $r \in R$.
4. A computes $Z_S = E_A(E_B(S)) = E_B(E_A(S))$.
5. For every pair $\langle E_A(r), E_B(E_A(r))\rangle$, A replaces $E_A(r)$ with corresponding r, obtaining a set of pairs $\langle r, E_B(E_A(r))\rangle$.
6. A selects all r, for which $E_B(E_A(r)) \in Z_S$, which equals $R \cap S$.

6.2 Schema Publishing

1. TSA maintains a private UDDI as a metadata repository. This repository is intended to be used by airlines to publish business and schema information to be shared with TSA. Every participating airline creates in this UDDI a Business Entity containing general business information about itself. It also creates a Business Service entry for each service

```
<Sii_DataProvider>
   <Table Name="PassengerList" Schema="AL">
      <Column>
         <Name>PassengerName</Name>
         <Description/>
         <DataType>VARCHAR</DataType>
         <Binary>false</Binary>
      </Column>
      <Column>
         <Name>OriginAirport</Name>
         <Description/>
         <DataType>VARCHAR</DataType>
         <Binary>false</Binary>
      </Column>
   </Table>
</Sii_DataProvider>
```

Figure 3: AL schema information.

```
<Sii_DataProvider>
   <Table Name="SuspectList" Schema="TSA">
      <Column>
         <Name>SuspectName</Name>
         <Description/>
         <DataType>VARCHAR</DataType>
         <Binary>false</Binary>
      </Column>
      <Column>
         <Name>Citizenship</Name>
         <Description/>
         <DataType>VARCHAR</DataType>
         <Binary>false</Binary>
      </Column>
   </Table>
</Sii_DataProvider>
```

Figure 4: TSA schema information.

offered. This entity includes schema information about data the airline is willing to share. The SIS platform provides a graphical interface to publish and retrieve business and schema information.

2. AL publishes schema information about passenger data within a new Business Service called "Passenger Information". Figure 3 shows a section of the schema information published by AL.

3. TSA similarly publishes part of its own schema (see Figure 4).

6.3 Negotiation

In an off-line step, TSA negotiates with AL about the specific use of AL's passenger data. As a result, AL agrees to allow TSA to use its data as follows:

- *PassengerName* will only be used as an intersection column (to learn the common names in the two databases).
- *PassengerName* cannot be used as a filter argument (i.e., in a selection predicate).
- As part of the negotiation, the TSA agrees that the use of the intersection results will be limited to the purpose of identifying suspects.

6.4 Schema Mapping

An application developer on behalf of the TSA looks for the column name and table name in the AL's database that correspond to the *SuspectName* column and *SuspectList* table in the TSA's database. The developer identifies *PassengerName* from *PassengerList* table as the corresponding column and table. The developer then decides to perform an intersection operation between *PassengerName* column from AL's *PassengerList* table and *SuspectName* column from TSA's *SuspectList* table.

6.5 Metadata Storage

1. TSA stores the following metadata in its SIS server metadata database:
   - Application name: *TSA_AL_App*.
   - View information: to access and retrieve *SuspectName* from table *SuspectList*.
   - Column rules [1] for *SuspectName*:
     - will be used as an intersection column;
     - cannot be used in a filter argument;
     - can be returned if in intersection result.
   - Supported operations: Intersection.
   - Access privileges: Application accessible to members of group: *TSA_AL*.
2. AL stores the following metadata in its SIS server metadata database:
   - Application name: *TSA_AL_App*.
   - View information: to access and retrieve *PassengerName* from table *PassengerList*.
   - Column rules for *PassengerName*:
     - will be used as an intersection column;
     - cannot be used in a filter argument;
     - can be returned if in intersection result.
   - Supported operations: Intersection.
   - Access privileges: Application accessible to members of group: *TSA_AL*.

6.6 Application Development

1. The SIS platform includes templates for information integration applications. The application developer creates a new application, *TSA_AL_App*.

[1] Column rules define in which part of an SQL query the column can be used.

Encryption Engine	number of records		
	1,000	5,000	10,000
CPU Intel III 2.0 Ghz	34s	175s	320s
AEP Runner 2000	3.5s	19s	37s

Table 1: Execution times for encryption UDFs.

2. She sets the template values for the Data Providers as *TSA* and *AL*, and sets the SIS operation to be performed to Intersection. She indicates that column *PassengerName* in AL's database corresponds to column *SuspectName* in TSA's database and will be used as the intersection column.

3. Finally, she creates a new user group called *TSA_AL*, and adds the authorized users from TSA. The authorization information is stored in the Authentication Authority's database.

6.7 Execution

For an intersection operation between TSA and AL the protocol performs the following steps to compute the result:

1. TSA encrypts the values of *SuspectName* column of *SuspectList* table using the commutative encryption function.

2. It sends an intersection web-service request to AL, with the encrypted *SuspectList* table as an attachment [8].

3. AL performs its part of the protocol by encrypting the values of *PassengerName* column of *PassengerList* table and double-encrypting *SuspectList* table from TSA. It then sends a web-service response to TSA with both encrypted tables as attachments.

4. TSA performs a double-encryption of the *PassengerList* table from AL. Finally, it uses both double-encrypted tables to perform the intersection operation between *SuspectNames* and *PassengerNames*, and returns the results to the application.

7. PERFORMANCE

Encryption time dominates the performance of the SIS Operations [8].[2] The encryption algorithm we use requires modular exponentiation [9]. We discuss the use of AEP Runner 2000 SSL cards [4] to speed up exponentiation. These cards are commercially available and cost about US $2000.

Our implementation employs a User Defined Function (UDF) [11] to perform encryption. A standard scalar UDF has a scope over only one row from an input table at a time. However, AEP Runner can gain the advertised speeds only if multiple (100-200) threads simultaneously post exponentiation requests to the card's API. Therefore, we used a *table UDF* that can operate on several input rows at a time. It fetches k rows at a time, and creates a separate thread for each row. Once all threads have finished, the encrypted rows are returned to the database and next k rows are fetched.

Our experiments show that AEP Runner can significantly speed up the execution of the UDF performing encryption (Table 1).

An additional feature of the AEP Runner API is that an application does not have to know how many cards are installed in the computer. The AEP scheduler distributes exponentiation requests between the cards automatically. Thus, we can expect a linear speed up with the increasing number of cards.

[2]However, other components may also become computationally expensive if not implemented carefully. For instance, we discovered that parsing database tables into XML web service arguments using Axis SOAP engine [7] requires more time than encryption. To avoid parsing the tables, we attach tables as binary files.

8. SUMMARY

Sovereign information sharing is useful for the creation of any virtual organization where the parties do not completely trust each other, or there are privacy and security concerns about the sharing of information. We described the architecture of the sovereign information sharing platform that we have built, and the rationale behind our design choices. In addition to the core algorithms, our platform also incorporates components for resource discovery, schema mapping, and authentication, thereby enabling rapid developments of SIS applications. We also showed that the use of encryption cards significantly improves performance.

Acknowledgements

This work was done as part of the SII Extreme Blue project, sponsored by Laura Haas, Nelson Mattos, and Dan Shiffman. Our thanks to Deon Glajchen, Tyrone Grandison, Jerry Kiernan, Amit Somani, and the Almaden Extreme Blue Staff, particulary Dave Cheney, for their help and insights.

9. REFERENCES

[1] DB2 Information Integrator. `http://www-306.ibm.com/software/data/integration/`.

[2] RosettaNet. `http://www.rosettanet.org`.

[3] Health insurance portability and accountability act. `http://www.hhs.gov/ocr/hipaa/`, 1996.

[4] SSL Accelerators, Interactive Buyer's Guide. `http://ibg.networkcomputing.com/ibg/Guide?guide_id=4065`, Nov. 2002.

[5] Web Services Security (WS-Security). `http://www-106.ibm.com/developerworks/webservices/library/ws-secure/`, Apr. 2002.

[6] UDDI Spec Technical Committee Specification. `http://uddi.org/pubs/uddi_v3.htm`, Oct. 2003.

[7] Apache Axis SOAP engine. `http://ws.apache.org/axis/`, Apr. 2004.

[8] R. Agrawal, D. Asonov, and R. Srikant. Enabling sovereign information sharing using web services. In *Proceedings of ACM SIGMOD*, June 2004.

[9] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of ACM SIGMOD*, June 2003.

[10] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the 28th VLDB Conference, Hong Kong, China*, Aug. 2002.

[11] D. Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, 1998.

[12] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving data mining. *SIGKDD Explorations*, 4(2):28–34, Aug. 2002.

[13] B. A. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *Proc. of the 1st ACM Conference on Electronic Commerce*, pages 78–86, Denver, Colorado, November 1999.

[14] T. Kontzer. Airlines and hotels face customer concerns arising from anti-terrorism efforts. *InformationWeek,* *http://www.informationweek.com/story/showArticle.jhtml?articleID=18401079*, Mar. 2004.