The Sybase Architecture for Extensible Data Management

Steve Olson Richard Plederer Phil Shaw David Yach
{*olson, pleder, phil.shaw, yach*}@*sybase.com*

1 Overview

Enterprise integration—unification of all of an organization's information resources, from the mainframe to the desktop, into a seamless, coordinated, easily accessed corporate asset that appears and works as one virtual system—has been an elusive goal for many large organizations. To help achieve this goal, and to help customers solve business problems and achieve competitive advantage, Sybase has introduced a comprehensive strategy for enterprise computing called the Adaptive Component Architecture (ACA). This architecture is based on the requirements of today's business computing environment, which can be summarized as follows:

- Use of industry-standard components
- Rapid application development
- Delivery of data in the right form, to the right place, at the right time
- Minimized complexity for both end users and developers

1.1 Adaptive Component Architecture

Based on open component logic, comprehensive development tools, and optimized data stores, Sybase's Adaptive Component Architecture[7] (ACA) provides a multi-tiered framework designed to manage and deploy components across the distributed computing environment. ACA features an Application Server tier and a Database tier. The Application Server tier supports component-centric computing (Jaguar Component Transaction Server) and page-centric computing (PowerDynamo). Sybase Adaptive Server forms the strategic Database tier.

The Adaptive Server brings existing Sybase DBMS products – SQL Server, Sybase IQ, and SQL Anywhere – into this unified architecture. Adaptive Server features optimized data stores for delivering predictable high-performance management of traditional and complex data within many different types of applications. It also offers a single programming and query interface across all the data stores, using Transact-SQL and standard components, including JavaBeans, running in the server. It also supports specialty data stores, which share a unified programming and operational model. Incorporated into the Adaptive Server is a component integration layer, which enables distributed access to each of these data stores, including multi database distributed transactions.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

1.2 Evolution of Sybase Data Access Middleware

In the early 1990s, Sybase management and engineering came to the conclusion that the Sybase RDBMS was not going to displace existing DBMS systems in customer accounts. Consequently, a strategic decision was made to adopt a policy of *coexistence*, rather than *displacement*. This decision led to a series of interoperability products, beginning with the Sybase Open Server. The Open Server allowed Sybase customers to write their own gateways that would inter-operate with the Sybase RDBMS and with Sybase Open Client application programming interfaces (API's). This toolkit was also the basis for the first Sybase gateway, called the Net Gateway, which enabled easy access to CICS transactions on MVS and IMS, and to DB2 using dynamic SQL.

Later, a series of additional gateways were introduced, all based on Sybase Open Server, which provided application and database interoperability with a variety of non-Sybase RDBMS, including Oracle, Ingres, Informix, Rdb and RMS files on VMS systems

While these gateways solved the primary problem of obtaining access to non-Sybase database systems using Sybase APIs, not all databases are created equal. An application needing access to Oracle could use Sybase APIs to access an Oracle gateway, but needed to use Oracle's PL/SQL syntax in order to do so. Additionally, if an application needed access to two or more database systems, it needed to use the specific SQL syntax associated with each, and it needed to manage SQL joins between two or more systems.

The OmniSQL Server product was conceived and resulted in Sybase's initial Multi-DBMS (MDBMS)[6]. This product enabled applications to interface with a single server even though access to many separate database systems was necessary. The database specific access mechanisms were hidden from the application by the OmniSQL Server. To an application, OmniSQL Server appeared to have the same look and feel as the Sybase SQL Server. The resulting *location transparency* was a key feature of the first release of OmniSQL Server. Additionally, the semantics of Sybase Transact-SQL were enforced. This *functional compensation* ensured that the behavior of the application could remain consistent, regardless of the nature of the data source(s) involved in various queries and transactions initiated by the application.

1.3 Sybase Adaptive Server

Sybase Adaptive Server (formerly known as Sybase SQL Server) ships in two variants: Adaptive Server Enterprise and Adaptive Server Anywhere. Both variants contain the Component Integration Services[8] which encapsulate the support for distributed, heterogeneous data management. Figure 1a illustrates the relationship between the Sybase Adaptive Server Enterprise, Component Integration Services, and external Sybase and non-Sybase database systems. Figure 1b illustrates the relationship between Sybase Adaptive Server Anywhere, Component Integration Services, and external Sybase and non-Sybase database Systems: Note the use of standard JDBC and ODBC APIs in case of Adaptive Server Anywhere.

The following sections will take a closer look how Adaptive Server supports distributed query processing, specialty data type support, and Java extensions.

2 Distributed Query Processing using Component Integration Services

Performance is the leading source of concern expressed by most users of distributed systems. Component Integration Services (CIS) addresses many of these concerns by focusing on two separate aspects of distributed query processing:

- Query decomposition – analyzing query syntax and determining the amount of work to be pushed to remote sites for processing
- Query optimization – analyzing query syntax to establish optimal join strategy and join order

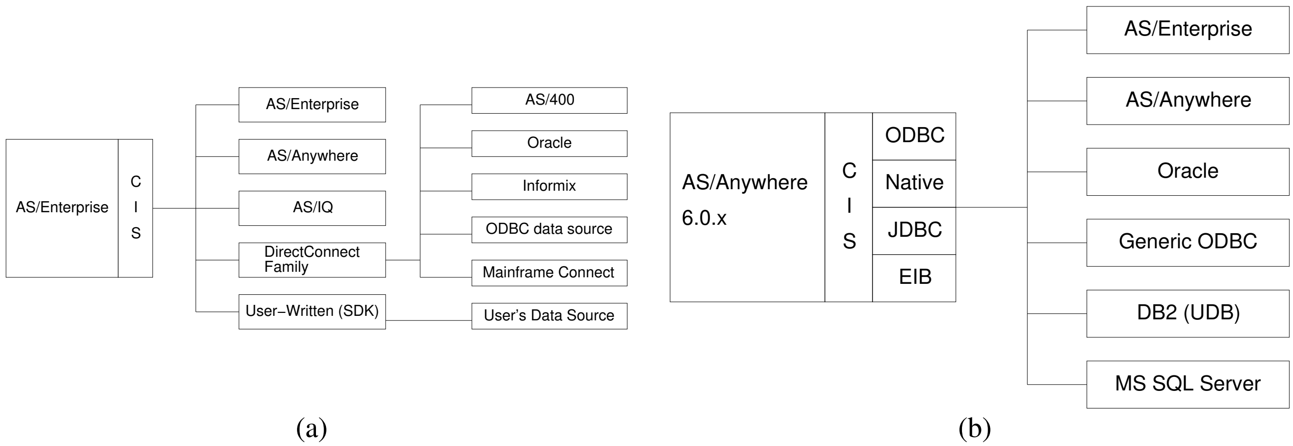

Figure 1: ASE, CIS and Distributed Access

2.1 Query Decomposition

CIS evaluates query trees at two stages of query processing:

- Pre-optimization – before the query optimizer is invoked to analyze various permutations represented within the query
- Post-optimization – after optimization, but before plan generation and execution, to determine if more of the work represented by the statement can be forwarded to a remote location for execution.

2.1.1 Pre-Optimization Query Decomposition

CIS will intercept query processing after the preprocessing stage has completed, but before query optimization is invoked. Preprocessing is necessary in order to perform view resolution. A decision will be made at this stage regarding the level of functional compensation that will be required. There are two questions that must be answered:

- Is every table represented within the SQL statement located on the same remote server?
- If so, is that remote server capable of handling all of the syntax/semantics represented by the statement?

If the answer to both of these questions is yes, then a query plan will be produced which will allow CIS access methods to reconstruct the entire statement and pass it to the remote server. Results will then be routed according to the needs of the statement (back to the client, inserted into another table, assigned to a local variable). This capability is referred to as *quickpass* mode. If the answer to either if these questions is no, then the query trees are passed to the query optimization phase.

2.1.2 Post-Optimization Query Decomposition

In many cases, *quickpass* mode cannot be selected because more than one remote server is involved in the query, or the needs of functional compensation dictate that some of the work must be performed locally, so as to preserve the expected semantics of the query.

Nevertheless, CIS will attempt to re-analyze portions of the query tree to determine if branches can be used to generate sub-select statements that can be forwarded to a remote server. This is particularly useful in the case of **union** operators - one side of a union may contain a query referencing DB2 tables, for example, while another side may contain a query referencing Oracle tables. In this case, the entire select on each side of the union can be reconstructed and sent to the affected database, and the results merged locally by CIS, according to the needs of the union operator (**union** or **union all**).

2.2 Query Optimization

The primary roles of the query optimizer when distributed joins are implicit in the query syntax are to determine access cost for each remote table and to determine join strategy.

2.2.1 Remote Table Distribution Statistics

Without proper cost information for remote table access, it is not possible to select a correct join strategy. Consequently, a means of obtaining distribution statistics for remote tables is needed. This is done in the Adaptive Server (and OmniConnect) through the **update statistics** command. If the object of the command is a local object representing a remote table, the data associated with remote indexes is requested, and processed locally as if the data was derived from local index pages. The resulting distribution histogram[9] and row count information is stored in local system catalogs for later use by the query optimizer.

2.2.2 Query Optimizer Inefficiencies

The primary problem facing the distributed query optimizer is that of calculating accurately the cost of accessing objects across a network. This is an especially important problem to solve when a query involves a combination of local and remote tables, or when the query involves tables residing on two or more separate database systems. If the costing is wrong, then it is possible that the tables that require network access will be positioned improperly in the resulting query plan, resulting in unacceptably high amounts of cpu and elapsed time to resolve the query. For example, in this query of local table L and remote table R:

```
SELECT * FROM L, R WHERE L.i = R.i
```

Local table L could end up being the outer table and remote table R could end up being the inner table of a nested loop join strategy, accessed with a remote query, as follows:

```
for each row of L:
SELECT ⟨ column_list ⟩ FROM R WHERE i = ?
```

As the number of rows in table L increases, the worse the results become. The cost of remote access must be taken into account, so that in this example, the remote table could be positioned as the outer-most table of the nested-loop join strategy, thereby eliminating the network cost for each member of the inner table.

Secondly, during post-optimization processing, peephole optimization can combine the queries to remote tables on the same server into a single sub-join. For this to occur, those remote tables must be adjacent in the plan and be directly connected. For all this to line up properly requires sufficient intelligence in the optimizer to recognize the location of remote objects, and to recognize that join clauses are possible to be pushed to remote locations.

These issues have been addressed in the Adaptive Server by the following changes to the query optimizer:

- A remote access cost formula to more accurately estimate remote query overhead has been implemented.
- Transitive closure has been introduced for joins to ensure remote tables located on the same server are connected at the first order when possible.
- The cost of access overhead for adjacent (in the query plan) remote tables located on the same server has been re-evaluated to reflect that the join between the tables will be processed remotely.

2.2.3 Evaluating the Cost of Network Overhead

Efficient execution of distributed queries requires that the remote tables be accessed once during the processing of a query (or even better, accessed as a group, once, via a remote sub-join). The largest cost is network access, not disk access. The existing cost formula takes into account estimates of logical and physical disk reads, and physical writes in the case of inserts. Access to remote tables also requires an estimate of the network i/o.

The following cost formula has been introduced into the query optimizer to calculate the cost of remote access:

```
(3 + floor((rows-1) / 50)) * WEXCHANGES
```

An exchange is one elemental network interaction: send a message to a remote server, get a reply. For a remote **select**, there is one exchange to open the cursor, one to return up to 50 rows, and another to close the cursor. If more than 50 rows are returned, another exchange is required for each additional set of 50 rows. We set the constant WEXCHANGES at 100 (nominally milliseconds) relative to WLOGICAL at 2, WPHYSICAL at 18. The estimated cost for disk access (logical i/o's, physical i/o's), although calculated as if the remote tables are local, is retained as an estimate of the cost of this work on the remote server.

With this formula in place, the estimated cost of accessing a remote table repeatedly via a parameterized **select** statement skyrockets, as it should. The cost of scanning it repeatedly is astronomical! But, the cost of placing that table as the outermost table in the plan and accessing it once by efficiently streaming its results over the network becomes very attractive.

2.2.4 Peephole Optimization and Remote Sub-joins

In many cases, reformatting is avoided by peephole optimization performed on the query plan. The query plan is evaluated to locate connected remote tables that reside on the same remote location and are adjacent in the plan. These tables are gathered into a single remote query, which performs a sub-join remotely. Only the result of the remote sub-join is retrieved over the network.

In order for this peephole optimization to be activated, the remote tables need to be adjacent in the plan and be connected at the first order. More modifications to the optimizer were required to make this fall out reliably.

First, transitive closure for joins was added to query processing. This algorithm finds all specified relationships between the tables by walking the tree in search of join clauses. Then, all implied but unspecified relationships are derived and added to the query tree as (redundant) join clauses. Now all relationships between tables are directly specified. For example:

```
SELECT * FROM R1, R2, R3
WHERE R1.i = R2.i AND R1.i = R3.i
```

This implies that R2.i = R3.i. Transitive closure for joins adds this join clause to the tree giving the query:

```
SELECT * FROM R1, R2, R3
WHERE R1.i = R2.i AND R1.i = R3.i AND R2.i = R3.i
```

Remote query costs are much lower when tables residing on the same server can be accessed in a remote sub-join. During join costing, adjustments are made to the cost of remote access when costing a remote table that is ordered immediately before another remote table on the same remote server. If these tables are connected and both tables are being evaluated with the nested iteration access method, then the cost estimate is modified as follows.

For each table pair in the current work plan permutation that represent remote tables at the same location, and are directly connected:

- Change the remote access cost of the inner table to 0.

- Calculate the remote access cost of the outer table as the result of the sub-join of the two tables.

In other words, estimate the number of rows returned from the remote sub-join of the two tables and cost that instead of costing individual remote sub-queries. This algorithm cascades over successive remote tables in the work plan permutation.

The net effect is to favor a query execution plan that groups remote tables that are located on the same remote server so they can be combined into a single remote sub–join. Also, regardless of whether there is a single remote table or a group, remote tables are highly favored as the outer table of the join, reformatted to a work table, or selected for a merge-join and, hence, accessed only once during the query.

3 Specialty Data Stores

The high-technology press and analyst community has generated considerable speculation regarding support for extensibility in relational database management systems and real-world requirements for user-defined data types. Discussions with customers reveal that they actually need a solution for a small, well-defined set of problems. This section provides a background on Specialty Data Stores and Sybase's strategy for supporting them in the Adaptive Server.

3.1 Requirement for Special Data Types

Increasing demand for access to more information available within an enterprise is driving requirements for the RDBMS to support other data types in addition to alpha-numeric. This demand can be met with a small number of specific data types that can dramatically improve the utility of client/server applications for the end user, and which can simplify the development of such applications.

The emergence of the Internet has also increased the focus on these specialty data stores. Although static in content, the information on most Web sites today includes a rich set of text and image data. The evolution of client/server applications to the Internet enhances the need for RDBMS to support these rich data types.

The specific data types that have gained mind-share for their potential are:

- Text – in the sense of "full-text" search and retrieval
- Time Series – essentially a compact array with intrinsic awareness of time concepts
- Geospatial – spatial semantics specific to geographical data

There are also specializations of these types, such as a fingerprint comparison application of the image content-type, but these tend to be vertical-market specific. Furthermore, there is a potential for application vendors to build their own types to facilitate their application design and development.

A primary support issue for these data types is that they must be integrated into the RDBMS. From the application and systems management perspectives, they should seem as if they are native features. Indeed, in some cases, the RDBMS can be extended to provide these capabilities natively. In other cases, in terms of performance and integrity, it will be most effective to support existing best-of-breed technologies and provide integration points into the server. A single request of the server should be able to access both native and federated services.

3.2 Text

Requirements for text come from the need to locate documents that pertain to transactions. Users need to search documents in their native formats (ASCII, HTML, XML, Word, WordPerfect, etc.) using a variety of powerful techniques, including "topical" searches. For instance, a trader might want to see documents relating to microprocessor developments in a search for "personal computers."

Applications that search text need to find documents wherever they are located. The RDBMS should be able to search documents stored in the database. However, it is often necessary or more efficient to leave documents outside the database and index them along with documents stored inside the DBMS.

The Sybase solution for full-text search requirements involves integrating software from Verity, Inc., into a separate search engine which is accessed by Component Integration Services when text search operations are requested by the client application. The search engine has knowledge of the Adaptive Server data store, and uses its data to construct text search metadata. This metadata (data about data) is used to allow the text search engine to rapidly scan through text indices to resolve a full-text search query.

3.3 Time Series

Several vertical market segments process data that has an intrinsic time attribute. Stock prices are associated with a ticker value, a day, an opening of the market, a closing of the market, etc. Each of these events gives different meaning to the price. A series of prices occur at regular intervals, such as a series of daily closing prices.

Simple storage and retrieval of such a series might be done in a standard table with two columns, but the time column is actually redundant. All that is really necessary to associate a given price with its relevant date is the starting date of the series and the offset into the list of prices. A time series data type is structured according to this notion of time.

Furthermore, operations on the series require the semantics of frequency conversions. For example, to produce a series of weekly closing prices, a conversion must be done on the series to locate the Friday values. To produce a five-day moving average, a function must process the series in rolling groups of five market days to get the resultant series.

Storage and indexing efficiencies must be considered for time series. Many series are sparsely populated, and most dedicated time series engines do not store the missing values. Nevertheless, the series can be quickly indexed via the location of values in the series – a concept foreign to relational table storage mechanisms.

The Sybase solution for handling time-series data involves a partnership with Fame, Inc., which has provided a Specialty Data Store that can be accessed by Component Integration Services in order to provide support for this data type to client applications.

3.4 Geospatial

Government bodies responsible for managing our public physical assets such as roads, utility infrastructures, emergency services, and natural resources have used mapping technologies from Geographic Information Systems (GIS). These tools process data accessed by latitude and longitude, and sometimes elevation. This data is typically called geospatial data.

Geospatial data has traditionally been processed in the application or client software. The data servers and DBMSs have had no understanding of the semantics, or indexing methods for geospatial data. However, there is a clear trend toward bringing the benefits of client/server and multi-tier computing to GIS. Providing data types, predicates, and indexes specific to geospatial data is an emerging DBMS requirement.

The Sybase solution for handling geospatial data types involves a partnership with Vision International, Inc., which provides a Spatial Query Server. Spatial Query Server (SQS) is an Open Server application that allows an application to include geographic constructs in SQL statements. SQS adds three enhancements to the Adaptive Server: spatial data types (e.g., point, line, polygon), spatial operators (e.g., inside, intersect), and two-dimensional spatial indexing. The SQS decomposed each query it receives, changes the enhanced SQL language, called Spatial SQL, into Sybase Transact-SQL, then sends the query to the Adaptive Server. The SQS then post-processes the result set coming back from the Adaptive Server and relays the final results back to the client.

4 Extensibility with Java

Java is quickly being adopted as a *3rd generation object-oriented language*. Furthermore, Java is beginning to play a major role in providing extensibility to Database Servers and Application Servers. The reasons for that are numerous:

- *Ubiquity:* Java is inherently portable.
- *Safe Component Integration:* Java supports strong typing, separation of implementation from interface and, most importantly, automated memory management which obviates the need for pointer manipulation.
- *Ease-of-Deployment:* Java treats code and data uniformly.
- *Reusable Components:* Java can run on any tier and Java objects can easily be exchanged between tiers.

4.1 SQLJ

Enterprise Computing is inherently multi-vendor. For this reason a set of *uniform APIs* is required to ensure interoperability among multi-vendor solutions. These APIs must be available across *all tiers* and support:

- Client Access,
- Scalable Component Servers,
- Warehousing Engines and
- OLTP Database Servers.

The SQLJ standard developed by the SQLJ consortium describes ways for Java to be used with SQL. The SQLJ effort is driven by major industry vendors such as Oracle, Sybase, IBM, JavaSoft, Tandem, Informix and others. SQLJ specifies the syntax and semantics for Java with Embedded SQL, Java Stored Procedures, Java UDFs and Java Data Types. SQLJ is well integrated with the JDBC API. The SQLJ approach to routines and data types is sometimes referred to as *Java Relational*[3]. For a complete description of the SQLJ standard, we defer to the reference material [1]. Sybase has been one of the major contributors to the SQLJ consortium.

4.1.1 JDBC

JDBC[2] provides Java programmers with a Call Level Interface to perform dynamic SQL operations. JDBC 1.0 is now widely supported by database systems and tools. Version 2.0 of JDBC has added features that allow the application and the database to exchange structured data, including Java objects.

4.1.2 Embedded SQL

SQLJ Embedded SQL allows Java programmers to include SQL clauses in their Java programs. The SQLJ Translator is then used to perform design time validation of the SQL statements and replace them with Java code. SQLJ Embedded SQL also specifies a vendor neutral runtime infrastructure and a vendor neutral representation of the preprocessed SQL statements.

4.1.3 Stored Procedures and UDFs

SQLJ Stored Procedures and SQLJ User-Defined Functions allow the programmer to write these routines in Java and install them through JAR files into the database. SQL operations are expressed by either coding Embedded SQL or raw JDBC calls. Client applications use JDBC, ODBC, etc. to invoke these procedures. By contrast, Java UDFs may be invoked from within SQL DML statements.

4.1.4 Data Types

With SQLJ Data Types programmers may use Java Classes to declare the types of columns, procedure parameters, etc. Java represents a very attractive choice for expressing complex types in a database, due to features like object-pass-by-value or the sandbox execution model. SQL DML statements are extended to permit the access to Java methods for relational operations such as project, filter, etc.

4.2 Java and Sybase Adaptive Server

One of the key features of the recently released Sybase database product Adaptive Server Anywhere[4] is the support for SQLJ-compatible Java Stored Procedures, Java Functions and Java Data Types.

4.2.1 Implementation Aspects

In order to ensure efficient integration of database processing with Java computing, Sybase Adaptive Server hosts a Java Virtual Machine (JVM). The JVM is responsible for performing all Java operations such as execution of *static methods* or invocation of *instance methods*.

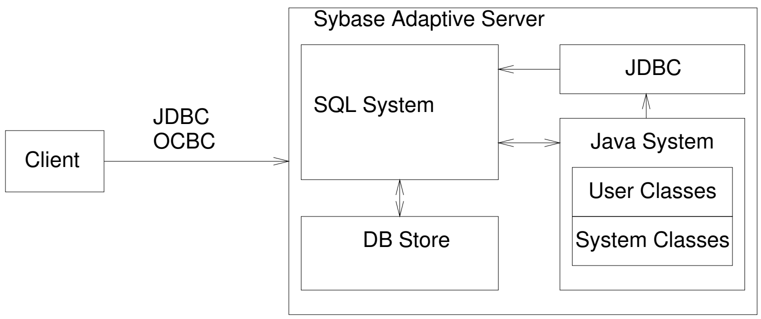

Figure 2: Integrating a Java System with SQL

When a SQL operation involves Java, the SQL System of Adaptive Server makes use of the Java System for operations such as:

- Instantiation of Java objects, either from persistent storage or from client transfer,
- Invocation of instance methods on Java objects,
- Invocation of static methods on Java classes.

Compiled Java code (classes) can be executed within the server environment using SQL statements. The use of a Java Virtual Machine in the database environment ensures that Java in the database fully conforms to Java standards and expectations. Public class and instance methods can be executed. Classes may inherit from other classes. Packages and the Java API are supported. Finally, access to protected, public and private fields is controlled. Every Java class installed in a database becomes available as a data type that can be used as the data type of a column in a table or a variable. An instance of a Java class (a Java object) can be saved as a value in a table. Java objects can be inserted into a table, SQL select statements can be executed against the fields and methods of objects stored in a table, and Java objects can be retrieved from a table. The use of Java in no way adversely or unnaturally alters the use of SQL statements or the way other database schema elements operate. An internal, high-performance JDBC driver executing native functions lets Java objects access SQL data within the database environment.

We expect users to deploy complex Java processing logic into Adaptive Server. An interactive, source-level debugger is provided to assist users with isolating problems in the deployed Java code. Furthermore, high performance access matters to database users. For this reason, attribute-level indexes may be created on columns that are declared as a Java type.

4.2.2 Java Stored Procedures

Sybase Adaptive Server was the first commercial database server to support Stored Procedures based on Sybase Transact-SQL[5]. Transact-SQL (TSQL) enables enterprise customers to host business logic close to the business data, thus improving performance and enabling Client/Server computing.

While Stored Procedures are now a common place feature for most commercial database systems, they have forced business programmers to give up on portability and modern language features in favor of performance. To address this, Sybase approached the SQLJ consortium with an initial proposal for a standard way to use Java as a Stored Procedure language.

Adaptive Server Anywhere Version 6 provides a first implementation of SQLJ-style Stored Procedures. This enables true portability of Stored Procedures across different DBMSs and maintains the capabilities of regular SQL Stored Procedures. SQLJ Stored Procedures are callable from ODBC, from JDBC, from other SQL Stored Procedures and directly from Java.

Java Static Methods Based on the SQLJ standard, any Java *static method* is callable as a stored procedure. Initially only parameter types and result types that are mappable to SQL types will be supported. However, the architecture is extensible to support arbitrary Java types. The body of the static method may use JDBC or SQLJ Embedded SQL to perform SQL statements.

The following Java class **Regions** implements two static methods, **region** and **correctStates**. The **region** method maps a state code to a region number. The **correctStates** method performs an SQL update to correct the state codes.

```
public class Regions {
    // region will be called as a function
    public static int region(String s) throws SQLException {
        if (s == "VT" || s == "NH" ) return 1;
        else if (s == "GA" || s == "AL" ) return 2;
        else return 3;
    }

    // correctStates method will be called as a stored procedure
    public static void correctStates (String oldSpelling,
                                      String newSpelling) throws SQLException {
        Connection c = DriverManager.getConnection("JDBC:DEFAULT:CONNECTION");
        PreparedStatement stmt = c.prepareStatement ("UPDATE emps SET state = ? WHERE state = ?");
        stmt.setString(1, newSpelling); stmt.setString(2, oldSpelling);
        stmt.executeUpdate();
        return;
    }
}
```

Java Stored Procedures may use JDBC or SQLJ Embedded SQL to perform database operations. In this example we are using JDBC, so we require access to a JDBC connection object before we are able to perform any DML operations. In the SQLJ standard, this is accomplished by establishing a connection to a special database URL: *"JDBC:DEFAULT:CONNECTION."* This URL effectively returns a connection to the implicit session context that the Stored Procedure executes in.

The **correctStates** method may then be invoked as a Stored Procedure from JDBC or ODBC. The **region** method may be invoked as a User Defined Function as illustrated below

```
select name, region(state) as region
from employees
where region(state) = 3
```

Any *Java static method* whose parameter types and resultset types are mappable to JDBC types may be called as a Stored Procedure. The architecture, however, is extensible to support arbitrary Java types. The body of the static method may use JDBC or SQLJ Embedded SQL to perform SQL statements.

Installing Java Stored Procedures Java Stored Procedures are installed into Adaptive Server Anywhere by installing a Java Class file or a JAR file. ASA automatically examines each Java Class for *static methods*. These methods are then made available as Java Stored Procedures. The Java reflection mechanism is used to determine the names, methods and signatures of the Stored Procedures.

ResultSet Processing Ordinary Stored Procedures may return result sets that are neither parameters nor function results. SQLJ Stored Procedures model ResultSets as follows. An additional clause in an installation descriptor associated with the Class file specifies that the procedure has result sets. Such a SQLJ procedure may be defined on a Java method with a result set as return value.

4.2.3 Data Types

Adaptive Server Anywhere supports the use of Java as SQL data types for defining columns of SQL tables and views. The advantage to the SQL programmer is that Java provides a simple, robust and ubiquitous type extension mechanism. Java features such as inheritance and encapsulation are immediately available. The advantage to the Java programmer is that the native support of Java objects in a relational SQL database obviates the need for mapping Java objects to scalar types or BLOB data types.

Here is an example of a Java class that we will use as SQL data type:

```
public class Address implements java.io.Serializable {
    public String street;
    public String zip;
    // A constructor with parameters
    public Address (String S, String Z) {
        street = S; zip = Z;
    }
    // Return a string representation of the full address
    public String toString( ) {
        return "Street= " + street + " ZIP= " + zip;
    }
};
```

Here is an example of a subclass derived from the **Address** class:

```
public class Address_2_Line extends Address implements java.io.Serializable {
    public String line2;
    // A constructor with parameters
    public Address_2_Line (String S, String L2, String Z) {
        street = S; line2 = L2; zip = Z;
    }
    // Return a string representation of the full address
    public String toString( ) {
        return "Street= " + street + " Line2= " + line2 + " ZIP= " + zip;
    }
};
```

Column Definitions Once a Java Class has been installed into Adaptive Server, it may be used to declare the type of table columns. The following example creates an **employee** table which contains the **home** column declared as type **Address** and the **mailing** column declared as type **Address_2_line**.

```
createtable employees (
    name          varchar(30),
    home_addr     Address),
    mailing_addrAddress_2_Line);
```

Insert Operations Regular SQL data manipulation operations are used to operate on Java Table Columns. For example, here is how a row is inserted into the **employees** table:

```
insert into employees values (
    'Bob Smith',
    new Address('432 Elm Street', '99782'),
    new Address_2_Line('PO Box 99', 'attn: Bob Smith', '99678'));
```

When the SQL system encounters the **new** keyword, it will internally instruct the Java VM to invoke the constructor method of the target Java class. In the example, it will invoke the constructor method of the **Address** class. This creates a new instance of type **Address**. The SQL system then effectively writes the value of this newly created instance to persistent store.

Retrieval Operations The user may retrieve an entire Java object or may project individual fields. The following example shows a SELECT statement where individual fields of Java objects are returned. Adaptive Server supports regular Java syntax for accessing instance fields and instance methods within a SQL statement.

```
select name, home_addr.zip, home_addr.street, mailing_addr.toString()
from employees
where home_addr.zip <> mailing_addr.zip;
```

When the SQL system in Adaptive Server encounters a reference to a column of type Java, it will read the value of the object instance into the Java VM and maintain an object reference to the instance. Subsequently, when the SQL system needs to access a field or invoke a method on this instance, it will dispatch to the Java system by passing that object reference.

Update Operations Update operations may be performed on individual fields or on the entire Java object. The following shows how the user might modify a field of a Java object.

```
update employees
    set home_addr>>zip = '99783'
    where name = 'Bob Smith'
```

4.2.4 Outlook

Our experience has been that database extensibility based on Java technology is readily accepted by database users. In addition, Java provides a built-in component model, JavaBeans, which facilitates the deployment of reusable components into the database server. Therefore, the natural progression will be for Adaptive Server to natively host scalable and transactional components based on the Enterprise JavaBeans model. Finally, we expect 3rd party vendors to provide a series of added-value components that are written in Java and that can now be safely installed into Adaptive Server.

5 Conclusion

The Sybase Adaptive Server family provides the ability to integrate enterprise information systems through the database extensibility mechanisms of Component Integration Services and Java. CIS provides intelligent distributed query processing capabilities for remote data access, and also provides the enabling technology for Specialty Data Stores. Both of these extend the reach of the Adaptive Server to enterprise-wide, legacy systems as well as to object and non-relational processing systems.

The creative use of Java in the database has provided a means for Sybase to implement support for abstract data types and user-defined functions, as well as to provide an object-relational capability within a database management system that has historically been relational only.

These extensibility mechanisms enable the Sybase Adaptive Server to co-exist in a heterogeneous world, and simplify the task of application development by presenting a uniform and consistent view of enterprise-wide systems. In addition, the extensibility features of the Adaptive Server open the door to whole new classes of applications that can take advantage of the extensibility for use in vertical markets.

Acknowledgements The authors would like to recognize everyone who contributed to the Adaptive Server Enterprise and Adaptive Server Anywhere products. The authors also recognize the efforts of the SQLJ consortium.

References

[1] SQLJ Specifications. Contact `Phil.Shaw@sybase.com`.

[2] JDBC Specification. `http://java.sun.com/products/jdbc`.

[3] Java and Relational Databases: SQLJ. SIGMOD Tutorial, G. Clossman, J. Klein, P. Shaw, R. Pledereder, M. Hapner, B. Becker, ACM SIGMOD 1998 Proceedings.

[4] Sybase Adaptive Server Anywhere V6.0, User Manuals.

[5] Sybase Adaptiver Server Enterprise Transact-SQL User's Guide, User Manual, Document ID 32300-01-1150.

[6] DB Integrator: Open Middleware for Data Access, R. Pledereder, V. Krishnamurthy, M. Gagnon, M. Vadodaria, Digital Technical Journal, 7(1), 1995.

[7] Technology and the Search for Competitive Advantage. `"http://www.sybase.com/aca/whitepaper.html."`

[8] Sybase Component Integration Services User's Guide for Adaptive Server Enterprise and OmniConnect, User Manual, Document ID 32702-01-1150.

[9] Histogram-Based Solutions to Diverse Database Estimation Problems. Y. Ioannidis and V. Poosala, Bulletin of the Technical Committee on Data Engineering, September, 1995, IEEE Computer Society.