INTERNET AND RELATIONAL DATABASES IN A
MULTI-TIER CLIENT/SERVER MODEL

By

Sanhita Sarkar
and
Shyam Sundar Sarkar

IMA Preprint Series # 1439
November 1996

INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS
UNIVERSITY OF MINNESOTA
514 Vincent Hall
206 Church Street S.E.
Minneapolis, Minnesota 55455
1. Introduction

Client/server technology evolved in late 1980s and early 1990s. It introduced many intelligent links from various client sites to server sites over the network. Even though client/server model represents an interesting step forward from earlier techniques, it suffered from some important negative aspects. Client/Server model relied heavily on the vendor that created it because information traveling back and forth between clients and servers was usually transmitted over a proprietary protocol intending to a closed solution. Client side logic was generally platform specific leading towards additional version control and cross-platform maintenance cost. In mid 1990s, there is a great need for open business applications that can be deployed into a continuously evolving matrix of business, partnerships and customers. This demands co-petition, continuous process improvement and platform independent, component oriented software solutions. To meet these demands, a new class of network-centric, multi-tier applications need to be developed supporting open standards. From the inception as an open, information-sharing medium, the internet and intranets have evolved into a full application development platform which transforms into a new paradigm compared to client/server model.

These network-centric applications embrace a collection of cooperating applications and network services. Applications are built by building new services that transparently integrate and customize the services offered by internets and intranets. These platform independent, network centric applications will save cost for developing and deploying applications The key feature of the internet is that all these network centric applications will cooperate and communicate with other services on the internet and intranets. Much of the popularity of the internet and intranets come from the fact that applications can be easily integrated with open protocols and gateways. The initial thrust of network-centric applications is in document management and information publishing. In near future, all kinds of applications will exploit platform independent architectures with open protocols to access data and information from anywhere in the world.
Ranging from desktop productivity applications requiring data from intranet servers, various heterogeneous mission critical applications will exploit the power of internet and intranets.

**Figure 1. Internet and various connectivity**

The relational database has proven its value as a data services solution for the last two decades. There is open database connectivity middleware to connect relational databases from anywhere in a network. In this new network-centric multi-tier model, rdms can be used to play multiple roles for storing and servicing different forms of data e.g. relational data, multi-media data, webpages, java scripts, definitions etc. Rdms should support all types of data for network-centric multi-tier application development model depending on the following technologies. Originally designed to represent text and images, HTML is rapidly developing into a client interface to develop network centric broad range of applications. HTML and Internet/Intranet web pages are now the universal desktop to run all kinds of applications and as a result a storage and retrieval mechanism for HTML web pages into relational databases is becoming an unavoidable task. The need for active web pages, dynamically formatted based on data or other information retrieved from anywhere, is growing rapidly. The value of databases as repositories for HTML pages, images, sounds and other web components is becoming more widely recognized. Also as the first programming language to provide a comprehensive solution to the challenges of the internet and intranets, Java is the most important thing to consider. Java is an object-oriented language which has a much broader scope for serious business application developments over the internet. Database connectivity and data manipulations from Java are becoming serious issues for upcoming internet and intranet centric applications. Java is designed to be platform independent by mapping any Java program into bytecode which can be interpreted anywhere on any platform. Java class definitions and interface definitions need to seamlessly integrate with relational databases. In a distributed environment with heterogeneous machines, applications are provided with distributed messaging system based on Internet Inter-ORB Protocol (IIOP). IIOP is part of CORBA and is open internet protocol for connecting applications across the Internet. IIOP enables developers to selectively expose the interfaces of network-centric applications, providing other developers with the ability to access the functionality as a service available on the internet or intranets.
Figure 2. Application Architecture.

The internet/intranet architecture should be designed to accommodate multiple object-based components, which can be plugged together to form a connection from the application requesting the service to the application providing the service. The service requester and provider can be on platforms running variations of Windows and UNIX. On Windows, the object framework is based on OLE/ActiveX. OLE/ActiveX and CORBA need to talk to each other for interoperability among heterogeneous platforms and software systems.

Figure 3. Web sites getting matured through phases.

2. Various Data Types - Their Sources And Clients

There are clients of different types, namely:

a) **Web Browser** as client will need various types of data. First thing is the "web page" which is written in HTML or VRML. Web pages can carry interfaces with java scripts, CORBA, Multi-media query languages, SQL etc. This client will maintain intrinsic display for HTML, TEXT, GIF, VRML etc. It will also maintain intrinsic cache storage, local file storage for binary or unknown objects. A data management technique for storing and retrieving various different types for this client is needed. HTML, TEXT, GIF and VRML pages as well as Java scripts as data
objects to be stored and retrieved from a database management system is the need currently addressed. All the features offered by a database management system e.g., security, concurrency, recoverability etc. should also be incorporated in managing these different data types in internet.

b) **Web Server** as client will need index retrieval from another web server for hyper-text/graphic objects and also for URL lists. Web server can also fetch interface modules written in CGI or MMQL etc. Native data services will involve intrinsic client objects such as GIF, HTML, TEXT, VRML etc., as well as client-foreign objects such as binary or unknown objects.

c) A **database management** system can also serve the role of a client to other web servers, file systems, applications in progress, another dbms or to an archival storage. A dbms can have stored procedures which can be executed to retrieve various forms of metadata and data objects. Metadata and data can be retrieved by using native protocols like ODBC, SQL, etc. Also java bytecode can be stored as "stored procedures" involving JDBC or messaging to other data object type interfaces. A dbms client can involve interface modules for data blades, Multimedia queries and user defined functions. Such interface modules and definitions can also be retrieved from other sources.

d) **Applications in progress** in internet or intranets can also be clients for data object descriptions and data object services. Data object definitions can be dynamically parsed models like SQL, ODBM etc. or static models like java classes, C++ classes, CORBA, OLE etc. Data object services can involve native instantiations with native model, dynamic instantiations or static models. These models again can be CORBA, OLE, SQL, java/C++ classes etc. Applications in progress can be clients to other dbms, other applications in progress, archival storage, file systems or web servers.

There are four possible **sources** of relational data for the internet and intranets as listed below:

a) **HTML/http route**: A URL is sent back to the web site where the data is requested and SQL scripts will pull out data to display in HTML. Such SQL scripts can be executed via java and ODBC driver for the data source. The data in the forms of table (columns and rows) can be displayed in a part of a web page or in a completely new web page. Web page construction can be tailored to display data with nested or related semantics. HTML pages can be constructed and stored as part of table definition in rdms for displaying different forms of relational data. HTML script can be stored as "stored procedures" in table definitions such that the page construction will be done on the fly for displaying data. Construction of a page can be parameterized based on number of tuples to be displayed, relationship to other data windows etc. In other words, relation type in a relational database can be associated with a HTML script description which is the type description in another form. Conversely, any HTML script should be mapptable to a relational schema representation. As we retrieve data to to view in HTML, data itself can be HTML script. This is important because web page storage in rdms is an open issue so that all good features of rdms can be used in web servers. HTML script describes a polymorphic type as opposed to monomorphic types in relational schema and table definitions. A mapping from polymorphic HTML types to relational schema with foreign-keys and normalized tables is a non-trivial task to be addresses soon.
b) A web server's script can execute a Java application program in the server's back end to message an object for data. The retrieved data can be formatted in HTML output for display. Java is object oriented language, it has classes, methods, constructors, packages, instances etc. A data type can be constructed by creating instance(s) of a class. A message can be sent to a constructor of a class to create instance(s) of a class. The constructor can fetch data from any type of data store (relational, non-relational etc.) and can use that data to initialize the instance variables in instance(s). Any object oriented database where data retrieval and manipulation are done by messaging to objects only and not by explicit query mechanism on a database, will fit in this model for data source on internet. The data accessing and data navigations are encapsulated in classes and well defined interfaces provide the connectivity for the outside client applications. Data objects encapsulated into classes provide many facilities for interoperability over the internet. Interfaces to data objects can be maintained by ORBs and applications running anywhere in the internet can retrieve or manipulate these data objects by talking through these interfaces in ORBs. Since the data objects are encapsulated into classes, construction of an instance of a class can involve messaging to other classes locally or remotely present on the internet or intranets. There are efforts by some vendors to make SQL objects. This implies that relational data access through ODBC will become “open”. Under these implementations, several classes are defined to deal with well defined interfaces for relational database storage and manipulations. A seamless coupling between relational databases and internet will depend on an object solution for relational data, tables, schema and queries.

c) A JDBC interface will go through a package of Java code to translate the information into a
JDBC call. A JDBC call opens a data source, retrieves the info and sends it back for display. JDBC interfaces are the Java based solutions for accessing data from relational dbms. These calls are at the transaction level, table level and query level. JDBC follows the path taken by Microsoft ODBC APIs based on CALL LEVEL Interface defined by the SQL Access Group and X/Open.

The ODBC interfaces allows:

A library of ODBC function calls that allow an application to connect to a DBMS, execute SQL statements and retrieve results;

Support for SQL92 syntax;

A standard set of error codes with support for native error messages;

Standard ways to connect and log on to a DBMS;

A standard representation of data types.

ODBC satisfies the need for solutions that exploit all of the features of a dbms as much as possible. If a dbms does not support a feature, it can notify the calling application. Later on, whenever the enhancement is there for the feature, the application can immediately make use of it. JDBC is based on ODBC but there are some elements of ODBC missing in JDBC as follows:

DBMS feature interrogation is not allowed in JDBC. ODBC allows users to ask a database whether certain features are implemented or not. If implemented, users can take advantage of these features;

ODBC allows a block of records to be fetched by a single call. JDBC supports only fetching of single row at a time;

ODBC provides for backward or forward movement of cursors through a result set. JDBC allows only forward movement which will make list processing more complex;

ODBC supports the idea of binding database columns to program variables which makes data movement between program storage and database buffers simple. JDBC does not allow such binding. The mapping of instance variables to database columns are supposed to be done by the user or developer;

ODBC allows a row of interest to be marked in a result set so that it can be located in future. JDBC does not provide such a feature.

If full power of ODBC is not required by some applications, JDBC interface can be used to gain access to ODBC-enabled databases. ODBC drivers will be required for implicit connectivity to databases from JDBC api. However, this solution requires special client side software to access databases. Installation of native software components on the client machines will be necessary to perform the translation to ODBC. This scenario can be simplified if the translation is done at the server side with another layer sitting somewhere or may be at the server side to listen to all clients for specific database requests. This layer can provide higher level of abstractions for data requests in higher level than JDBC api. Java application classes will like to retrieve and manipulate data objects viewed at a level higher than SQL.
Applications may like to send messages to business objects and deal with business logic rather than low level tables and queries. JDBC-ODBC connectivity problem can be resolved if this higher level view mechanism for data objects can be formalized as a layer sitting between client application and JDBC api.

d) **Object request broker** technology can be used for general applications written in Java to access and manipulate data objects sitting anywhere on the internet. Here an interface is defined to an object and the behaviour on that Java object is executed using a standard ORB implementation. The concept of remote method invocation requires that involved components can be interfaced in terms of remote operations supporting different execution semantics. The interfaces should be described by an interface definition language allowing to abstract from the encapsulated component specifics. Since interfaces can become known during run-time when new managed objects are introduced, the ability for dynamic binding of interfaces is mandatory. The basic concept of CORBA relies on the client-server paradigm. Clients invoke operations at the servers denoted as objects in CORBA's terminology. An object itself can act as a client for other targets whose operations are needed to satisfy the initial request. Unlike other client/server models, CORBA deals with objects for data retrieval and manipulation.

When an object is created, its object-id or object reference is created for further reference. A state of an object is preserved dynamically till the time that object is destroyed. Over the internet or
intranets, data objects and their interfaces will be supported through CORBA. Since objects can be anywhere in a distributed system (internet), any complex levels of object abstraction for data will be possible. Unlike client/server scenario using ODBC for data retrieval and manipulation, CORBA allows users and developers to deal with high bandwidth data with higher levels of abstraction. The basic principle is to provide object wrapper for any kind of data in the internet for CORBA compatibility. For relational dbms, data is usually stored in tables in a particular location on the internet. If there is an object abstraction involving some business logic at some other location, then the state of that object should be maintained or can be made persistent. Making a business object persistent implies that the underlying data retrieved and manipulated after relational operations from rdms (which is located somewhere else), should be stored as materialized views for further manipulation. If that is not done, then everytime a specific business object instance is messaged for manipulating underlying data, a re-execution may be required for the underlying relational operation at the data server. Distributed object system manipulating data from relational databases should incorporate views and their materializations at various levels of abstractions to reduce the re-execution cost and network cost. Construction of multiple levels of views and materialization techniques should be adopted for multi-tier CORBA compatible network objects.

In the following sections, a discussion over functional extensions over relational schema will be made to address the issues mentioned in the previous sections.

3. Polymorphic Types For Web Pages

A web page is a polymorphic type because it can consist of many text sections, images, audio scripts, Java scripts etc. There can also be links to many other web pages from anywhere in a web page. With all the links and heterogeneous mixture of audio, image, text etc. a web page is difficult to be represented in a static type format. Let us consider here the types representable in a relational database and consider the possibilities for extension. A table in a relational dbms contains a set of attributes, each attribute is of a fixed type. An example of a student table format is given below:
Student (Student-Id, SSN#, Name, Age, GPA):

<table>
<thead>
<tr>
<th>Student-Id</th>
<th>SSN#</th>
<th>Name</th>
<th>Age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>123456566</td>
<td>John</td>
<td>30</td>
<td>3.5</td>
</tr>
<tr>
<td>345</td>
<td>342487459</td>
<td>Kay</td>
<td>23</td>
<td>2.6</td>
</tr>
<tr>
<td>445</td>
<td>324233334</td>
<td>Mary</td>
<td>15</td>
<td>4.0</td>
</tr>
</tbody>
</table>

In the above table, Student-Id is the primary key to the table Student (Student-Id, SSN#, Name, Age, GPA). The same table can be represented in a different way by representing attributes in a column-wise fashion. This transposed description with column-wise representation is as follows:

Student (StudentId, PropertyName, PropertyValue):

<table>
<thead>
<tr>
<th>Student-Id</th>
<th>Property-Name</th>
<th>Property-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>SSN#</td>
<td>123456566</td>
</tr>
<tr>
<td>112</td>
<td>Name</td>
<td>John</td>
</tr>
<tr>
<td>112</td>
<td>Age</td>
<td>30</td>
</tr>
<tr>
<td>112</td>
<td>GPA</td>
<td>3.5</td>
</tr>
<tr>
<td>345</td>
<td>SSN#</td>
<td>342487459</td>
</tr>
<tr>
<td>345</td>
<td>Name</td>
<td>Kay</td>
</tr>
<tr>
<td>345</td>
<td>Age</td>
<td>23</td>
</tr>
<tr>
<td>....</td>
<td>........</td>
<td>........</td>
</tr>
<tr>
<td>445</td>
<td>GPA</td>
<td>4.0</td>
</tr>
</tbody>
</table>

In the second table, attribute name is encoded into character strings which are values of the attribute Property-Name. Next column Property-Value contains the values corresponding to Property-Names (this is an union over many different basic types). The primary key Student-Id is left as it is because in a join, Student-Id will be used to identify various pieces of a Student record. In the second table Student-Id, Property-Name can play very well as primary key definition. There is a principal difference between the first table and the second table. The first table is of static type. It is monomorphic. The second table is dynamically typed (in the sense that type names and values can be added or dropped) i.e., polymorphic in nature. Tables in relational databases can be represented in a transposed fashion to address dynamic type problems. We now use this polymorphic representation possibility for storing web pages. One can formally represent a type and its transpose (as illustrated) in the following way:
A type can be transformed into a transpose and vice versa for relational operations etc. A web page can be represented as a polymorphic type as follows:

**PAGE table**

<table>
<thead>
<tr>
<th>PageId</th>
<th>PropertyName</th>
<th>ForeignKeyToVal</th>
<th>PageLocation1</th>
<th>PageLocation2</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>TEXT</td>
<td>124124823</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>123</td>
<td>IMAGE</td>
<td>243664264</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>123</td>
<td>PAGE</td>
<td>230</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>230</td>
<td>TEXT</td>
<td>223424423</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Above table is a representation for web pages where TEXT, IMAGE etc. are the names of other properties. TEXT represents a property name which is a textual type represented in a separate table. Similarly IMAGE is a property name which is an image representation in a separate table. ForeignKeyToVal is a foreign key to point to a record (a large object) in another table. A property name can be PAGE to signify another web page being linked through the first web page. To retrieve the value of the PAGE property name, the foreign key 230 will be used to join with the same PAGE table. There can be URLs to represent remote pages and other attributes of interest. A TEXT table can be represented as:

**TEXT table**

<table>
<thead>
<tr>
<th>TextId</th>
<th>FontSize</th>
<th>Columns</th>
<th>BLOBPointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2234555436</td>
<td>16</td>
<td>4</td>
<td>x55555AAAAA</td>
</tr>
<tr>
<td>*********</td>
<td>********</td>
<td>********</td>
<td>**************</td>
</tr>
</tbody>
</table>
There are separate files for text and image large objects (character strings). Pointers to those objects will be maintained as BLOBPointers in TEXT and IMAGE and other tables. A HTML skeleton can be stored in the main web page record which can be augmented with accurate descriptions of text and image types by taking join and unions as:

\[
\text{[ (PAGE Join TEXT) Union (PAGE Join IMAGE) Union \ldots \ldots ]}
\]

The advantage of dynamic typing in relational storage is to use integrity, concurrency, recovery and other techniques available in relational engine. One can also think about the power of triggers and stored procedures for advanced manipulations over web pages. In the following subsection some of the possibilities for automatic web page maintenance and other object embedding possibilities will be discussed.

**Relational Operations On A web Page or its components** is possible to retrieve and manipulate various components in a complex situation. A portion of a web page can be retrieved, modified and stored back. For example, a text portion of a web page can be retrieved and modified simultaneously by many users. Concurrency control will be in place by rdms. Versioning is also possible for previous incarnations of web pages or their components. It is possible to choose a version of a web page from storage for manipulations. Sharing of components will be possible by foreign key relationships in many different objects. Unlike relational operations we have today in rdms, it is necessary to add an extra operator TRANSPOSE for transposing information as described above for polymorphic types. Joins can be taken before or after TRANSPOSE is applied to records in a table. A SQL query involving TRANSPOSE will look like:

```sql
SELECT HTML_description() FROM
    schema.T1 ,  TRANSPOSE (schema.T2)
WHERE  T1.key = T2.foreignKey AND
       T2.PropertyName = TEXT AND
       T2.ForeignKeyToVal = TEXT. TextId;
```

A view can be defined with TRANSPOSE operator used like any other relational operators used and such a ViewCache can be kept anywhere in a network. ViewCache can be materialized whenever the need arises. View materializations will be discussed later.

**Triggers And Stored Procedures** can add lot of values to web page storage and manipulation mechanism. Triggers can be written and provided to get components from many different locations or to get updates from other locations for the same web pages or the components or to enforce certain integrity checks for various pieces of hypertext information in web pages. Stored procedures can reformat a page based on size constrains adding intelligence to authoring systems.
not available in the market. Stored procedures and triggers can be effectively used to do many interesting things automatically as follows:

Auto formatting for web pages are not present today. HTML documents does not automatically adjust for font size, image size, locations of various pieces for space optimization etc. Intelligence built within a web authoring system will add lot of values to today’s systems. Modifications to web pages can be done freely without worrying about interrelationships across the sizes and embedding of various components. Active web pages using stock information actively from some location or tabular values needing different space in a page, can be implicitly taken care of by stored procedures or triggers.

Web pages of great interest can be brought in from various internet locations and stored in rdms keeping live contact. This will reduce network time delay everytime a page is needed. The information in web pages brought in can be used in other documents (Word documents, Excel) or charts. These pieces of information should be up-to-date for critical value of the document. If web pages are brought in from some other location in the local storage system, triggers can be used to find whether there are any changes in the web pages in the distant locations. This can be done by checking last update time. If there are changes, update logs can be brought in to incorporate into local web storage. A dynamic client/server link can always be maintained for intelligent accessing in the web.

Stored procedures can be used to link to special packages for manipulating an object in the database. For example, a text object retrieved from rdms can be implicitly linked to Microsoft WORD for editing and changes. The linking can be done by a stored procedure which uses OLE automation facilities for object linking. Similarly, a chart can be implicitly linked to EXCEL by stored procedures.

**SQL Windows front end** can be used as a client tool for managing and manipulating hypertext objects. Using SQL windows (e.g. QUEST product from Centura software), one can select a database for documents and images, create a table and enter, sort and edit data or text in the table, query a single table, link two tables in a query and arrange result set of a query etc. It is possible to generate a HTML form (master/detail form) and then enter data in or out of database tables. Pages can be viewed at any time for adjustment of data, text and images. SQL windows should be made to be supported by OLE automation to link to other products like Microsoft WORD, EXCEL etc., which is discussed later.

### 4. Objects - ViewCache And View Materialization

ViewCache is a technique which is under active research in the database community. Views have been used for providing conceptual subsets of the database to different users and as an implementation tool for efficiency, security etc. Views can be used as constructors for objects in an object paradigm sitting on top of relational databases. In particular, for internet applications, one can visualize application components sitting anywhere on a network working on metadata (Views) whereas the real data is sitting at specific locations on the internet. For each view, we can store a collection of indices that point to the records of base relations required to materialize the view. These indices can be primary keys, foreign keys or record pointers. The resulting table
(or index) is very compact and can be loaded and kept in main memory. They can be thought of as a form of an I/O or network cache. For this reason, we shall call them ViewCache. The records or indices in the ViewCache represent interrelationships in base relations and as a result the ViewCache is dealing with Metadata. It is possible to manipulate metadata before materializing a view. Reexecution cost can be very high at the database server. By maintaining ViewCache in different locations of a network, re-execution can be avoided.

**Objects and Multi-level ViewCache** can be made to correspond to each other in a hierarchy of definitions, storage and persistence. Given two ViewCaches, another higher level ViewCache can be constructed by storing keys, foreign keys or pointers from those two ViewCaches. An object A can have two subobjects B and C. The constructors for the objects can be based on ViewCaches in hierarchy. For example, sub-objects can relate to lower level ViewCaches and constructor for object A will relate to second level ViewCache. ViewCaches will be used to instantiate a class (for example a Java class) before applying methods. A method in an object can be applied only after materializing view pointers from base relations. Parts of a method can be applied at the metadata level (keys, foreign keys, pointers, relationships etc.) and the rest of the method can be applied after materialization of the view. Based on the business logic (in a multi-tier client/server model) in applications, it is possible to replicate few more fields from base relations in ViewCache records. The reason for doing so is that a method can be applied to completion just by working on the information in the ViewCache records without further materialization. This will lead to true distributed computing with proper separation of data, metadata and related business logic. Such extensions for relational database implementations will be required to address complex multi-tier solutions.

![Diagram](image)

**Figure 10. Correspondence Between Objects And ViewCaches.**
Objects and Views are distributed over a network as self contained units to collaborate with other units. Views will exist at the client site, at the sites where business logic is performed and at the mobile users, partners locations etc. This implies that the business logic will stay the same but the way the user interacts with their applications will change. An application will be viewed as a combination of several components on the internet. Each component is a self sufficient unit sitting at a certain level of abstraction. For each such component a persistent store will maintain views over elements in other units for other components on the network. Components will talk to each other through CORBA. Each persistent store can be a small footprint database to maintain views and other local data. A SQL engine will serve this purpose at different locations on the web.

Views are not just made of keys, foreign keys, pointers etc. Views can involve attribute values from base relations along with keys and pointers. The attribute values necessary for a particular business logic can be maintained in a view for the objects corresponding to the business logic. This is a notion where a relevant data portion is exposed for distributed executions. For example, if an application handles customer information, contracts, shipping, and billing then components can be created for each of the four activities. This means that application server will be able to handle more users since each one will be trying up fewer resources. Each such component activity can act on some records and some attribute values which can be maintained in the views along with record pointers. The replicated attribute values will prevent application logic to always travel to data servers for data retrieval and manipulations.

Views are updated by sending logs from data servers or other view servers. If views are constructed and maintained at multiple levels of abstraction, update logs will be sent from a level to next higher level for applying the updates. This implies that data servers and view servers will collaborate with each others for consistency and integrity across the whole network. Updates can be deferred updates or of some other form of our choice.
**Flow of functionality** will be from client site to business logic sites and finally to data server sites. Unlike client/server systems we have today where data comes from data server before any logic can be executed, in a multi-tier client/server model incorporating objects and views it is possible to apply business specific functionalities in a top-down fashion and there will be migration of view pointers along with some associated functionalities (to be applied) from one site to another. The view will be refined and manipulated at every site on a network before it is sent to base level data servers for materialization. The cost of large joins and other operations will be avoided. Data and control flow in an application’s logic will be newly implemented.

**Role of Java** will be critical in this top-down flow model. If certain business logic cannot be handled in the exposed data portions in a view at a node, it will be possible to package the Java bytecode corresponding to the business logic along with the view records and be sent to a view server or data server for execution. This will not be possible in a platform specific scenario where data and logic are tied to the platform. Java applets can be used for implementing parts of business logic on partially exposed data values in views or can be sent to other sites containing relevant data portions in views. As the database applications are tied to the locations where the data resides, it is not possible today to distribute logic anywhere in a network because data transfer is expensive. This problem is well taken care of by adopting views with partial data exposition and platform independent Java language.

![Diagram](image)

**Figure 12. Transfer of Java bytecode and View Records.**

The model resulting from above description is very flexible with respect to users and application developers. Any component or view can be modified without affecting others in a complex environment. A particular component can be enhanced to deal with new data or views from other sources without affecting other existing components. In a complex corporate environment, such component oriented approach for intranets and internets will be highly valuable.
4. JVL (Java-View Linking) On Top Of JDBC

Every node in a network will maintain its own relational dbms. Java applets running at each node can access data from local rdms by connecting through JDBC. JDBC enables object based database connectivity. There are number of advantages for object oriented database connectivities. Since objects can be embedded into other objects or one can build a hierarchy of object definitions, higher level object abstractions for data can be defined based on lower level data objects. JDBC interfaces are there to connect to relational dbms. However, higher level abstractions for business objects can be made by defining more complex data objects with more complex built-in logic where JDBC calls are made in the constructor definitions of these more complex objects. By seamlessly calling JDBC api in constructors, any complex business object with complex logic can be implemented in Java. Such objects can be defined and manipulated through JVL. JVL is a set of Java classes to define, create and manipulate complex views for business objects. JVL sits on top of JDBC.

**View materialization** from a node needs view records to be sent to proper data servers. This can be done by implicit communication between two JVLs sitting on top of databases in the nodes. JVL abstraction is a set of higher level objects to seamlessly communicate with remote and local JDBC interfaces for view materialization. Objects in JVL will talk to remote JVL objects through Object Request Brokers. JVL can talk to another JVL in a network for fetching remote data to materialize views. The problem of maintaining JDBC-ODBC bridge everywhere along with JDBC is avoided by keeping JVL. JVL can talk to ODBC through JDBC-ODBC bridge for data access in a server. However, at the client site, only JVL is needed to communicate with remote JVL. JVL will lead to several possibilities for object linking and embedding in different frameworks. JVL can be designed to completely encapsulate all of the complexities and differences of the relational model into object-oriented database classes. JVL can also preserve the power of SQL so that dynamic environments can adjust rapidly on the fly.

![Diagram](image)

**Figure 13. JVL to JVL connectivities via ORBs.**

By applying the power of the object oriented view model and the relational model, JVL can provide a fast, easy to use, class interface which allows one to construct a small Java applet to produce powerful application.

**View creation and deletions** can be done in a local node through JVL. A view creation implies a query to be executed in a data server and then foreign keys, selected attribute values, keys or
record pointers will be brought in to populate ViewCache in the local node. JVL interfaces will allow such creations, deletions or view modifications to be possible. A ViewCache can contain records with specific attribute values brought in from the data server and replicated for business logic applications. Such attributes can be modified, updated etc. which needs an implicit transaction on the data server containing the base records. These transactions, multi-threading, asynchronous executions etc., will all be handled by JVL.

**View updates** by collaborations across network nodes will be done by JVL components. Whenever a ViewCache is maintained at a node, if there are any changes in the base level values or other views on which the current ViewCache depends on, then the update logs from base level data server or other views will be brought in to the local node and the updates will be incrementally applied to the current ViewCache. Incremental updation needs to be done before a view is used in any transaction. Incremental updates of ViewCaches can be made automatic by keeping stored procedures in the local dbms. These stored procedures will interrogate with the base level tables or other related views to check if there were any updates made. In case of any updates, the logs will be brought in to apply the updates. Since ViewCaches can be constructed using any levels of hierarchy, current ViewCache should maintain an update log which should be sent to dependent views if there is a request. VL should provide mechanism to send update logs and apply them to any node in the network.

**Security considerations** should be a superset of the security issues for JDBC implementations. On top of that JVL will enforce security considerations for view objects. In a multi-tier access to database servers, middle tier services will implement special purpose business objects to enforce business logic. For example, customer objects at a node may implement operations for customer invoicing, address change and other transactions. A security enforcement for legal transactions from legal users for such business objects can be done, avoiding direct unrestricted updates to the dbms servers. JVL can be used to enforce such security issues for views. Views can be created with many different purposes. For example, views can be created with just read only access. Operations on a read only view will prevent any unauthorized updates to the database. By enforcing view types, JVL can allow many complex mechanisms of security enforcements. Such security enforcements are not possible by JDBC alone. A view can be created by accessing records from two other views with different access privileges. The resulting view will enforce security which will be different from attribute to attribute. Such security possibilities are not present in dbms or any connectivity solution.

5. **Integration of JVL with OLE**

With OLE automations, one can give users of integrated solutions access to data in different ways. For example, one can start applications and open documents so that users can view information or edit it directly in that application, allow users to send data to other applications, allow users to retrieve data from other applications, control the actions of other applications by invoking commands on the objects that are exposed to OLE automation. OLE automation creates applications that "listen", and programming tools are used to "talk" to these applications, directing them to accomplish a given task. OLE automation allows users to use the best of individual applications or tools and combine it all into a complete solution. For multi-tier
application model, OLE automation will be a powerful tool for data publishing, remote access to analysis data etc. JVL objects can be supported by OLE for view creation and manipulations. Users are free to define a view and create a ViewCache in a local machine. Next OLE automation can be used to link to JVL objects and embed them into other applications like WORD documents, reports etc. The power of defining views for higher levels of data abstractions can be nicely utilized in many ways. Since JVL can talk to other JVLs in a network, the remote accessing will be transparent to an application using JVL classes through OLE automation.

**Live and active** data can be accessed and used by applications using JVL classes through OLE automation. If an application is using view records to draw a graph through OLE automation, then that graph will be upto-date with any updates which can take place in the data servers or other view locations. Since JVL will apply the update logs incrementally from remote locations, the local view records will remain consistent with data servers anywhere. The graph drawn will be modified next time the application is run again if there are any updates. This active connections to remote data and views will be very useful in many domains of applications.

![OLE support for JVL](image)

**Web page management** was discussed in Section 3. Using relational dbms extensions for polymorphic types. It is interesting to point out that view creation and maintenance will add to web page creation and maintenance. Any table definition containing columnwise description for various attributes of a web page is a view for polymorphic types where the operation is a TRANSPOSE operation. JVL can provide facilities for transpose of a type and interpretation of transpose operation with respect to other relational operations in view creations and view maintenance. Editing and intelligent creation of web page can be done by using SQL windows supported by OLE automation. If JVL is also OLE supported, then SQL windows can talk to JVL to retrieve data from local rdmss and display results. Hypertext values can be displayed using respective applications. For example, a text can be edited by WORD, a picture can be edited by MS DRAW, a chart can be drawn by EXCEL etc. All these things can be done switching from JVL to respective applications and then going back to JVL. This integrated solution for web page
maintenance with implicit view definitions (remote or local) will add lot of strength to tools available today.

6. Summary

In this document, internet, intranets and various requirements have been discussed. Web page creation and maintenance has been discussed by comparing static versus dynamic typing in relational databases. Relational dbms needs to address the idea of dynamic typing for storing and manipulating heterogeneous, polymorphic types. A robust solution like relational dbms will be soon required for web page servers and browsers. Next this document elaborated on the ideas of ViewCache creation and maintenance problems. Views are constructors for objects sitting on top of relational databases. Views can be created with multiple levels of abstractions and various security issues can be resolved as view maintenance level. ViewCache maintenance and manipulations will answer the complex database question for multi-tier client/server applications. A new idea of partial exposition of data values in view caches has been discussed. A Java View Layer can be defined and implemented to manipulate views as opposed to base records from a relational data server. JVL will sit on top of JDBC and will talk to remote JVLs through CORBA. JVL will add powerful data abstraction for network centric applications. Java bytecode along with view records can be transported to other network nodes for materialization and computations. This paradigm will add completely new feature to object paradigm and relational dbms integration. JVL and OLE integration will add enormous power to desktop computing solutions and in general to distributed computing. Web page storage, creation and retrieval will be further automated and secured by storing web pages in rdms and integrating JVL, SQL windows and OLE.
Recent IMA Preprints

# | Author/s | Title
--- | --- | ---
1348 | A.V. Fursikov | Certain optimal control problems for Navier-Stokes system with distributed control function
1349 | F. Gesztesy, R. Nowell & W. Pötz | One-dimensional scattering theory for quantum systems with nontrivial spatial asymptotics
1350 | F. Gesztesy & H. Holden | On trace formulas for Schrödinger-type operators
1351 | X. Chen | Global asymptotic limit of solutions of the Cahn-Hilliard equation
1352 | X. Chen | Lorenz equations. Part I: Existence and nonexistence of homoclinic orbits
1353 | X. Chen | Lorenz equations Part II: “Randomly” rotated homoclinic orbits and chaotic trajectories
1354 | X. Chen | Lorenz equations, Part III: Existence of hyperbolic sets
1356 | C. Liu | The Helmholtz equation on Lipschitz domains
1357 | G. Avalos & I. Lasiecka | Exponential stability of a thermoelastic system without mechanical dissipation
1358 | R. Lipton | Heat conduction in fine scale mixtures with interfacial contact resistance
1359 | V. Odisharia & J. Peradze | Solvability of a nonlinear problem of Kirchhoff shell
1360 | P.J. Olver, G. Sapiro & A. Tannenbaum | Affine invariant edge maps and active contours
1361 | R.D. James | Hysteresis in phase transformations
1362 | A. Sei & W. Symes | A note on consistency and adjointness for numerical schemes
1363 | A. Friedman & B. Hu | Head-media interaction in magnetic recording
1364 | A. Friedman & J.J.L. Velázquez | Time-dependent coating flows in a strip. part I: The linearized problem
1365 | X. Ren & M. Winter | Young measures in a nonlocal phase transition problem
1366 | K. Bhattacharya & R.V. Kohn | Elastic energy minimization and the recoverable strains of polycrystalline shape-memory materials
1367 | G.A. Chechkin | Operator pencil and homogenization in the problem of vibration of fluid in a vessel with a fine net on the surface
1368 | M. Carme Calderer & B. Mukherjee | On Poiseuille flow of liquid crystals
1369 | M.A. Pinsky & M.E. Taylor | Pointwise Fourier inversion: A wave equation approach
1370 | D. Brandon & R.C. Rogers | Order parameter models of elastic bars and precursor oscillations
1371 | H.A. Levine & B.D. Sleeman | A system of reaction-diffusion equations arising in the theory of reinforced random walks
1372 | B. Cockburn & P.-A. Gremaud | A priori error estimates for numerical methods for scalar conservation laws. Part II: Flux-splitting monotone schemes on irregular Cartesian grids
1373 | B. Li & M. Luskin | Finite element analysis of microstructure for the cubic to tetragonal transformation
1374 | M. Luskin | On the computation of crystalline microstructure
1375 | J.P. Matos | On gradient young measures supported on a point and a well
1376 | M. Nitsche | Scaling properties of vortex ring formation at a circular tube opening
1377 | J.L. Bona & Y.A. Li | Decay and analyticity of solitary waves
1378 | V. Isakov | On uniqueness in a lateral cauchy problem with multiple characteristics
1379 | M.A. Kouritzin | Averaging for fundamental solutions of parabolic equations
1380 | T. Aktosun, M. Klaus & C. van der Mee | Integral equation methods for the inverse problem with discontinuous wavespeed
1381 | P. Morin & R.D. Spies | Convergent spectral approximations for the thermomechanical processes in shape memory alloys
1382 | D.N. Arnold & X. Liu | Interior estimates for a low order finite element method for the Reissner-Mindlin plate model
1383 | D.N. Arnold & R.S. Falk | Analysis of a linear-linear finite element for the Reissner-Mindlin plate model
1384 | D.N. Arnold, R.S. Falk & R. Winther | Preconditioning in $H(div)$ and applications
1385 | M. Lavrentiev | Nonlinear parabolic problems possessing solutions with unbounded gradients
1386 | O.P. Bruno & P. Laurence | Existence of three-dimensional toroidal MHD equilibria with nonconstant pressure
1387 | O.P. Bruno, F. Reitich, & P.H. Leo | The overall elastic energy of polycrystallin martensitic solids
1388 | M. Fila & H.A. Levine | On critical exponents for a semilinear parabolic system coupled in an equation and a boundary condition
1390 | J.M. Berg & H.G. Kwatny | Unfolding the zero structure of a linear control system
1391 | A. Sei | High order finite-difference approximations of the wave equation with absorbing boundary conditions: A stability analysis
1392 | A.V. Coward & Y.Y. Renardy | Small amplitude oscillatory forcing on two-layer plane channel flow
1393 | V.A. Pliss & G.R. Sell | Approximation dynamics and the stability of invariant sets
J.G. Cao & P. Roblin, A new computational model for heterojunction resonant tunneling diode
C. Liu, Inverse obstacle problem: Local uniqueness for rougher obstacles and the identification of a ball
K.A. Pericak-Spector & S.J. Spector, Dynamic cavitation with shocks in nonlinear elasticity
G. Avalos & I. Lasiecka, Exponential stability of a thermoelastic system without mechanical dissipation II: The case of simply supported boundary conditions
B. Brighi & M. Chipot, Approximation of infima in the calculus of variations
G. Avalos, Concerning the well-posedness of a nonlinearly coupled semilinear wave and beam-like equation
R. Lipton, Variational methods, bounds and size effects for composites with highly conducting interface
B.T. Hayes & P.G. LeFloch, Non-classical shock waves in scalar conservation laws
K.T. Joseph & P.G. LeFloch, Boundary layers in weak solutions to hyperbolic conservation laws
Y. Diao, C. Ernst, & E.J.J. Van Rensburg, Energies of knots
Xiaofeng Ren, Multi-layer local minimum solutions of the bistable equation in an infinite tube
Vlastimil Pták, Krylov sequences and orthogonal polynomials
T. Aktosun, M. Klaus, & C. van der Mee, Factorization of scattering matrices due to apportioning of potentials in one-dimensional Schrödinger-type equations
D.N. Arnold, R.S. Falk, & R. Winther, Preconditioning discrete approximations of the Reissner-Mindlin plate model
M.A. Kouritzin, On exact filters for continuous signals with discrete observations
R. Lipton, The second Stekloff eigenvalue and energy dissipation inequalities for functionals with surface energy
R. Lipton, The second Stekloff eigenvalue of an inclusion and new size effects for composites with imperfect interface
W. Littman & B. Liu, The regularity and singularity of solutions of certain elliptic problems on polygonal domains
C.R. Collins, Spurious oscillations are not fatal in computing microstructures
M.A. Horn, Sharp trace regularity for the solutions of the equations of dynamic elasticity
A. Friedman, B. Hu & Y. Liu, A boundary value problem for the Poisson equation with multi-scale oscillating boundary
P. Baumann, D. Phillips & Q. Tang, Stable nucleation for the Ginzburg-Landau system with an applied magnetic field
J.M. Berg, A strain profile for robust control of microstructure using dynamic recrystallization
P. Klouček, Toward the computational modeling of nonequilibrium thermodynamics of the Martensitic transformations
S. Chawla & S.M. Lenhart, Application of optimal control theory to in situ bioremediation
B. Li & M. Luskin, Nonconforming finite element approximation of crystalline microstructure
H. Kang & J.K. Seo, Inverse conductivity problem with one measurement: Uniqueness of balls in $\mathbb{R}^3$
Avner Friedman & Robert Gulliver, Organizers, Mathematical modeling for instructors, July 29 – August 16, 1996
G. Friesecke, Pair correlations and exchange phenomena in the free electron gas
Y.A. Li & P.J. Olver, Convergence of solitary-wave solutions in a perturbed Bi-Hamiltonian dynamical system
C. Huan, On boundary regularity of vortex patches for 3D incompressible euler systems
C. Huang, A free boundary problem with nonlinear jump and kinetics on the free boundaries
X. Chen, C. Huang & J. Zhao, A nonlinear parabolic equation modeling surfactant diffusion
A. Friedman & B. Hu, Optimal control of chemical vapor deposition reactor
A. Friedman & B. Hu, A non-stationary multi-scale oscillating free boundary for the Laplace and heat equations
X. Chen, Existence, uniqueness, and asymptotic stability of traveling waves in nonlinear evolution equations
J. Yong, Finding adapted solutions of forward-backward stochastic differential equations – Methods of continuation
J. Yong, Linear forward-backward stochastic differential equations
D.A. Dawson & M.A. Kouritzin, Invariance principles for parabolic equations with random coefficients
R. Lipton, Energy minimizing configurations for mixtures of two imperfectly bonded conductors
D.C. Dobson & F. Santosa, Noninvariance evaluation of plates using Eddy current methods
W. Littman & B. Liu, On the spectral properties and stabilization of acoustic flow
S. Sarkar & S. Sundar Sarkar, Normal distribution as a method for data replication in a parallel data server
S. Sarkar & S. Sundar Sarkar, Parallel view materialization with dynamic load balancing: A graph theoretic approach
S. Sarkar & S. Sundar Sarkar, Internet and relational databases in a multi-tier client/server model
J. Liang & S. Subramaniam, Numerical computing of molecular electrostatics through boundary integral equations
J. Wu, Inviscid limits and regularity estimates for the solutions of the 2-D dissipative quasi-geostrophic equations
P. Constantin & J. Wu, Statistical solutions of the Navier-Stokes equations on the phase space of vorticity and the inviscid limits
M.A. Kouritzin, Stochastic processes and perturbation problems defined by parabolic equations with a small parameter
M.A. Kouritzin, Approximations for singularly perturbed parabolic equations of arbitrary order
A. Novick-Cohen Triple junction motion for Allen-Cahn/Cahn-Hilliard systems