Basic Issues for Developing Distributed Applications Interacting with Legacy Systems and Databases

Rodolphe Nassif, Jianhua Zhu, and Pankaj Goyal
U S WEST Advanced Technologies
4001 Discovery Drive
Boulder, CO 80303

Abstract
Legacy systems have been developed over many years. Each system was developed independently. This resulted in duplication of data and of semantics. Legacy systems and new systems communicate with other legacy systems through their user interfaces or through files. Problems that need to be addressed include selective migration of data and functionality to corporate subject databases, maintenance of business rules, redundancy of data and distributed transaction management across heterogeneous environments. Our partial solutions to these problems include: vertical slicing of applications, building procedures to support an application slice at low level of granularity, forwarding of invocation of migrated operations to existing as well as to new applications, constraint enforcement at several levels, extensive use of remote procedure calls technology. Transaction processing systems will help in integrating our partial solutions. However, there is a need for a comprehensive approach to solve these problems.

Keywords: legacy systems, business rules, transaction processing, data migration.

1 Introduction

Today's software systems supporting business operations are data intensive. They require frequent access (including update) to a variety of information from large, and heterogeneous data sources. Their enormous complexity makes it increasingly difficult for the systems to keep up with the business' operational requirements, from both a maintenance viewpoint and a new development perspective. Legacy systems (LS) were developed independently of other systems. They lack any application programmer interface.

Communication to LS is typically done through their user interfaces or through intermediate files.The semantics of data updates are buried into application code. This has resulted in complex interactions between systems. Furthermore, transaction boundaries are defined by independently developed systems. In many applications, several operations are invoked on different systems and further down the process, another application would have to check the results of the operations and take appropriate actions. In many cases some manual intervention is needed to handle inconsistencies that should have been discovered earlier in the process if appropriate interfaces for applications were available. In managing this environment, issues of data redundancy and associated ramifications arise. The intention of this paper is to illustrate, using an actual application (section 2), some of the issues related to data access (section 3), to outline some approaches to the solutions (section 4), and to stress the need for further investigations to address inadequacies of current solutions.

2 Field Access System

The Field Access System (FAS) provides for field technicians the means to directly access their job loads, review jobs, and commit work, among other features. Jobs consist mostly of phone/cable installations, repairs, and testing. A portable PC would be used by the field technician to connect to a UNIX[1] application processor which executes the transactions submitted by the technicians. The transactions involve interactions with other systems such as the Work Force Administration/Dispatch Out (WFA/DO) which actually manages most of the information manipulated by FAS. WFA/DO is responsible for assigning jobs to field technicians, pricing, dispatching and committing of jobs. The databases involved are stored in IMS and DB2. WFA/DO interacts with other systems such as Service Order Processing, TIRKS which manages the inventory and LMOS for trouble ticket maintenance. Some important issues to FAS project that are not covered in this paper include access security and internetworking.

3 Issues

We will present some of the issues that are faced by FAS and other U S WEST application developers in general terms as the problems are common to many applications.

3.1 Heterogeneous databases

Applications need to access data managed by a multitude of database management systems (for relational, IMS, flat files) under different operating systems (DOS, UNIX, MVS).

3.2 Application to application interface

Most systems provide only a textual or screen interface. Therefore, applications (e.g. Billing) needing services of (an)other system(s) (e.g. work force administration), usually would make calls to the application's user interface through

1. UNIX is a trademark of AT&T

terminal emulation. Getting results could be instantaneous or accomplished at a later stage by a message sent by the called system to the originating system possibly through intermediate files.

3.3 Long transactions

Conceptually transactions could span days or weeks (e.g. installation of a phone line/repair) and involve several systems. A conceptual transaction is broken into several small transactions. Status codes are extensively used to determine the status of a job. When a system accomplishes a transaction on behalf of another system, a message is sent back, status codes checked and appropriate actions taken. Manual intervention is often needed (a human being has to initiate transactions to several systems). Great savings could be accomplished if a system could handle long transactions. For example, in one region of U S WEST territory there are 100,000 monthly service order requests (new services + maintenance). Request for Manual Assistance exceeds 10,000 per month.

3.4 Duplication of data

This is a very widespread problem. Traditionally, when two systems needed common information, the adopted solution consisted of replicating the data in both systems. For example, currently at U S WEST there are hundred of these systems with hundreds of databases containing around 10 terabytes of data stored in relational, network, hierarchical DBMSs and flat files. The address field, for example, is partially replicated in tens of systems. Maintaining the consistency of the address between a handful of these systems would save millions of dollars (cost associated with manual intervention to reconcile addresses, dispatching work force to wrong address, billing, etc.). Another example from the FAS application: the work request segment (in IMS) duplicates circuit information from service order segment (in IMS). The technician log segment duplicates information on the jobs with the work request segment (e.g. price, commitment dates, access times). Some of the fields may be calculated from other information in the database (e.g. estimated price computed from parts needed, technician time, etc.).For the purpose of this discussion, we distinguish three categories of inconsistencies, namely, lexical, syntactical and structural inconsistencies.

3.5 Lexical inconsistency

Misspelling or synonyms of field values are typical examples of lexical inconsistency. These problems may lead to wasted time and money when it comes to making business decisions such as whether or not a customer should be given telephone service based on a credit check, and whether or not two apparently different billing addresses are actually the same physical address, etc.

3.6 Syntactical inconsistency

The source of syntactical inconsistency mainly come from synonyms of field names (or attribute, column names in relational terminology) and record names (or table, relation names). The challenge here is to be able to identify the synonyms, with little or no documentation, so that all pertinent information required by applications become accessible to them.

3.6.1 Structural inconsistency

Variations in schema design result in structural inconsistency. For example, customer names in a customer information table can be represented by a single character string, or three character strings, for last name, first name and middle name, respectively. A third possibility is to store customer names in a separate table, with a foreign key (say, customer-number) establishing its connection to the customer information table. The challenge here is to map the variations in structure into a form that applications can interpret and process.

3.7 Semantics of updates

Semantics of updates to data fields is buried in application code. For example, when a field technician completes a work request, some checks need to be made and a message is sent to the Service Order Processing as well as an update to the local database.

3.8 One record type for multiple concepts

In order to save space, it is common in IMS databases to store in the same field different sets of data depending on the content of other fields. For example, an installation-or-maintenance field in the work request database may contain information on new installations or on trouble ticket for maintenance. This aspect complicates the mapping of segments from an IMS database into a relational database using commercial tools.

3.9 Distributed systems

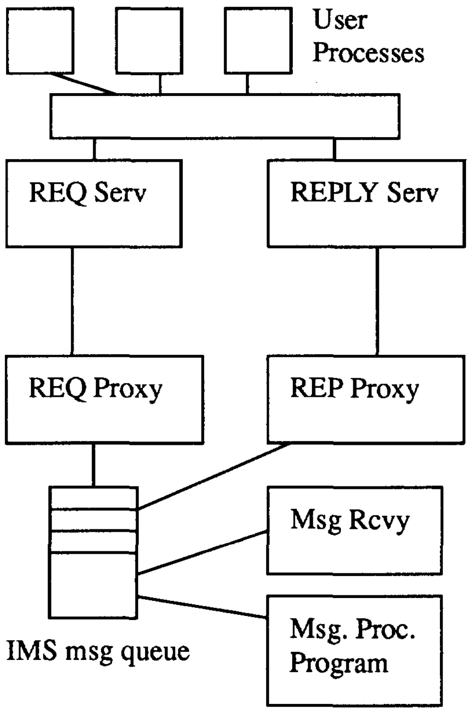
Several complex systems interact in order to provide a service such as installation of a phone line. The different systems exchange messages synchronously (e.g. a message through the user interface of another system) and asynchronously (e.g. an update to some fields in a database so that it would trigger some action in the other system). Some architectural guidelines should be developed on how new applications should be designed in such an environment.

4 Questions

There have been many approaches proposed to address the issues similar to the ones above. We briefly overview some of them, and raise a few questions, which we feel need further investigation.

4.1 Heterogeneous data access

Some tools are starting to appear on the market that provide a conceptual relational view of information that may be stored in relational, network, hierarchical databases, or flat files. The databases might reside on UNIX, MVS or other operating systems. These tools are oriented more towards ad-hoc queries. They are not appropriate to handle invocation of existing procedures (e.g. message processing programs on IMS, or procedures encoding Create, Update, Delete rules for DB2). Efficient handling of procedural code is needed for production systems where thousand of users may be making the same query. For the field access system we developed a queued based message processing mechanism that ensures delivery of messages to a resource manager (e.g. IMS) and replies to application processes. We have used ONC remote procedure calls over IBM 3172 TCP/IP connection for communication from UNIX to MVS. This solution does not address broader issues of transaction consistency across several systems.



4.2 Long/Nested/Multidatabase Transactions

Most database management systems do not support nested transactions. This forces us to write our procedures without a begin/end transaction, so that calls could be nested. Some forms of nested transactions could be used to support long-lived transactions, e.g. Sagas and split transactions. These variants allow for sharing of data before commitment of the long lived transactions. Compensation procedures are used in some of the solutions. They specify what to do if an abort is done on a subtransaction that communicated its results. This is appropriate in many applications. For example, if a new service order installation is to be cancelled, depending on the status of the work it is possible to define what needs to be done to cancel the order. Current systems in fact use the concept of compensation procedures with flat transactions. An explicit call is made to a "compensation" transaction to roll back.

Most approaches to multidatabase transactions cannot be applied with existing DBMSs (e.g. require access to internal scheduling of DBMS, relaxation of ACID properties) or would result in large occurrences of rollbacks as the ordering of transactions is basically left to chance. Transaction processing systems and database vendors are starting to provide partial support to multidatabase transactions (2PC, nested transactions, compensation procedures).

4.3 Multiple database access

Proposals for supporting multiple databases accessing can be grouped into three categories.

4.3.1 Total schema integration

This approach is predicated on the premise that a common schema can always be derived for a set of related schemata that exist. Some relevant questions to ask are:

- What are the criteria based on which we can say the target schema is equivalent to the set of original schemata?
- Is pure structural transformation sufficient to derive the target schema?
- Do integrity constraints need special treatment and are they always preservable?
- What additional difficulties, if any, dependencies of schema interpretation on attribute values bring into the picture? (e.g. same field in an IMS database may be mapped to several fields in different relational tables. The actual mapping of specific database records will depend on the value of a specific field in the record being mapped)

4.3.2 Partial schema integration

This is the approach advocated by federated database management systems, in which the schemas are arranged into distinct layers, much in the same way as the architecture in the ANSI/SPARC proposal. A component database in such a system supplies a view on top of its conceptual schema, selectively exporting information contained there for the purpose of joining selected federations. A federated schema is an integration of the partial schemata from each of its member databases. In a sense, this approach recognizes the difficulties in a complete integration. At the same time, a total integration may not always be necessary, e.g., on disjoint databases. The issues here include:

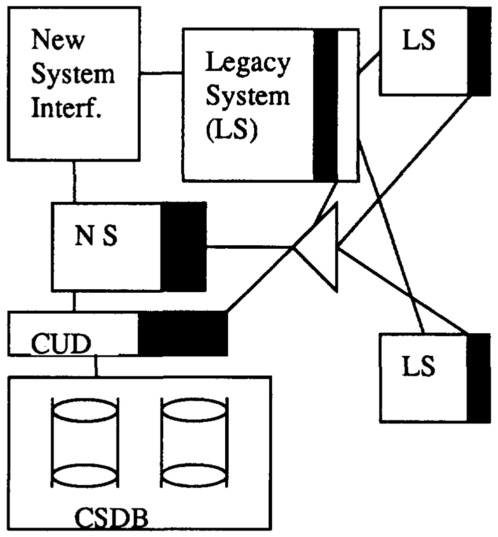
- Significant complexity is introduced by these additional layers of schemata and their associated mapping processors.
- The trade-off between added flexibility by allowing portion of a component database to participate in an integration and its complexity, vs. how often this kind of flexibility is absolutely essential.
- What do we lose by completely eliminating the additional layers by forming federations on (possibly overlapping) subsets of the original schemata?

4.3.3 Distributed views

This is a no-integration approach. No global coordination processor is assumed. Each database is itself responsible for maintaining one or more views of the other databases containing related data. Each view in turn is responsible in obtaining physical data and populating them in the way prescribed by the view. This approach has very limited applicability but can be cost effective in simple cases where, for example, only lexical and syntactical inconsistencies are of concern. In these cases, only a very thin layer of mapping software is required to support a view. It is flexible in that views can be selectively built on demand, in an incremental manner. No changes to existing applications are necessary, while newly developed applications may access multiple databases in a unified or even a transparent fashion.

4.4 Update semantics

The approach taken to ensure consistency of data is to put the constraint enforcement as close to the data as possible. The first level is constraint supported by the DBMS (e.g. foreign key), the second level are stored or general procedures for Create/Delete/Update (CUD) at low level of granularity. We are investigating the use of a constraint specification language. A constraint could be supported via the generation/invocation of several CUD procedures. We are taking an incremental approach to capture the semantic of updates. Critical functions from LS that need to be enhanced (or new functions) are selected. Relevant information from the corporate subject data bases (CSDB) are identified and CUD procedures are written that ensure the semantics of the data. The new systems (NS) keep forwarding to the LS the operations they support. Another approach to developing new applications that is being applied in some cases is to encapsulate operations on LS at a high level of granularity using existing LS application code. The new system would invoke these operations. This approach is less appealing as it leaves the semantics buried in application code. However, it is less expensive to implement.



5 Discussions

Ensuring data integrity and managing transactions in an efficient and robust manner are perhaps the most critical issues in developing business applications. Technologies addressing these issues have to work with legacy systems. We have emphasized the need for transaction management to ensure consistent update semantics. The other important aspect of data integrity is detection and correction of inconsistencies in existing databases. The remaining issues, purification and transition, are considerably harder and require a great deal more effort. The questions are:

- How can we move as quickly as possible into the target environment, without interrupting existing systems' normal operation? Are there any strategies or architectures to migrate some of the data/processing from mainframe to workstation environments?
- How can we make sure that a rigorous and effective purification process is in place so that we do not pass inconsistencies in to the target environment?
- Different transition strategies may better suite different target environments: Are there identifiable characteristics which we can use to make sure a best fit?
- Incremental transition will play an critical role in smooth and non-interruptive transition: Is there an effective methodology in identifying those increments, determining granularity as well as dependencies? Tools to capture business rules from existing code/databases are needed.

Acknowledgment: We thank Dave Hill, Robert Keyser, Steve Melville, Dave Pezzutti, Joe Wetzel, and the reviewers for their comments which enhanced the quality of this paper.