simplified systems incorporated

Enterprise Applications: An Overview

Introduction

This document is intended to provide an objective view of the current state of business application development, and describe how Simplified Systems intends to fulfill these needs. The scope of the class of systems that can be implemented under the Simplified architecture will be defined, and an item by item listing of the general functionality of these systems as they are today will be listed for comparison with the Simplified Systems product offering.

Characteristics of applications

The Simplified Systems product is designed to primarily support a class of business systems known as online transaction processing systems (**OLTP**). Once only used in very large corporations and governmental institutions, these database-backed business applications have become prevalent in companies of modest size in the last several years as the hardware necessary to support such systems has become cheaper and more readily available.

Most modern implementations of these types of business applications all share the same basic characteristics. The heart of these applications is almost always a commercial relational database, which is used to store all of the data tracked by the application. The database is also used to safeguard the data itself through the structural integrity requirements and the backup/restore capabilities of the database. In addition, the database's transaction control functionality is usually, but not always, used to prevent situations where errors in either the user input or the client software cause part of a user request to fail, resulting in data corruption.

All of these systems include business logic of varying complexity, implemented either in the database, the client application, or in a specialized middleware server tier. When examined at its lowest level, this business logic performs three basic functions, and is triggered whenever a user attempts to commit data to the underlying database system. The most obvious purpose of the business logic is data validation, which is the process of examining the values that the user is attempting to commit, determining whether or not the data is valid and either halting or continuing the user request. As an extension to the process of data validation, user input is often examined, and business rules are used to determine other values of the data that the user is attempting to commit, and 'filling it in' for the user in the process. The more complex scenario found in business logic, triggering other actions, is simply the process of examining the data that the user is committing, and then using it to manipulate data stored elsewhere in the system, which represents a different task in the context of the business that the application is servicing. While interfacing with other applications is often cited as another function of business logic, it is usually implemented as simply 'flagging' data for transfer to take place later (becoming data manipulation, rather than true real-time system integration) in lieu of actually talking to another application.

The user interfaces for the applications are often patterned closely after the underlying data structure, generally representing one or two of the database tables, with elements of other related tables pulled in for information purposes. These interfaces are presented as both desktop and web-based applications. The end-user generally has the capability to search and view the existing data, and to create, edit, or delete data through these forms. Additionally, the user interface generally provides the capability to jump to related forms (forms representing related tables) for a given record. The user interfaces themselves are usually not very sophisticated or robust, particularly when compared to consumer products, and do require training and adaptation on the part of the end-user to become proficient in their use.

The final characteristic of these types of systems is their monolithic architecture (one application, one application server, one database), and the mechanisms by which the application communicates with other applications within the business. The approaches selected for performing these integrations have taken on a wide range of forms in practice. At its crudest level, the integration is performed by one application reading from and writing to another application's underlying database. Automated importing and exporting of batch-file data has been one of the most popular approaches over the years, often using simple flat-file formats such as comma and tab delimited, or more sophisticated formats such as **EDI** and **XML**. Live transfer of information between applications using formats such as EDI has been in use for several years, while the equivalent using XML is rapidly gaining momentum. In a few high-end systems, real-time system integration has been done using **RPC** technologies such as **RMI** and **CORBA**, any number of messaging middleware solutions, or specialized integration middleware applications such as Vitria.

The driving motivation behind the development of these systems is to serve the business's needs. To achieve this end, it is imperative that the application be implemented to fulfill the business requirements as quickly and cheaply as possible. The most successful business applications do not deliver optimum performance, do not have the most sophisticated user interfaces, and do not implement business logic or integrate with other applications in the most technologically sophisticated ways. These tradeoffs are made because the time and resources needed to build a true state-of-the-art business system cannot be justified by the gains that they will provide to the business.

Historical progression

Starting with the appearance of commercial relational database systems and the client-server architecture, there has been a steady progression of tools and technologies to facilitate the development of business applications. The relational database was one of the first steps to simplifying the development of business applications by removing the necessity for the developer to deal with low-level issues of storing and retrieving data.

Soon after the advent of the database itself, a series of code-generating development tools appeared. This step in rapid application development reduced the amount of repetitive programming tasks that the developer needed to do, particularly for building user interfaces, enabling them to concentrate development efforts on more sophisticated elements of the application, such as business logic.

However, one of the biggest shortcomings of the client-server model was the problem of what to do with the business logic. Putting all of the business logic in the client was simple due to the ease of implementing business logic via general purpose programming languages in the client and the availability of the client's processing power to execute the business logic. However, the 'fat-client' approach did suffer from several drawbacks; the slightest change in the business logic required a redeployment of the client side program to every client that accessed the application, and implementing the business logic did nothing when another client or application needed access to the system. Placing the business logic in the database was a logical alternative to these issues; however, database-scripting languages were less than ideal for development or debugging, and the additional load placed on the database server by processing the business logic could severely impede performance.

The **N-tier** approach provided a viable solution to this problem, and CORBA and messaging-based middleware products simplified the development of these systems by eliminating the need for the programmer to deal with the internals of working with the networking APIs of the various operating systems involved in the deployment architecture. With the presence of a middleware system between the database and the client, more services and efficiencies such as pooling, caching, load-balancing, and client management began to appear in business applications as well.

Some of the services arising out of the N-tier application led to another advance in the architecture of business applications, the application server. The application server provided a mechanism for dealing with middleware-specific issues such as database access, client management, and transactional control, freeing the programmer to focus primarily on the data and business logic of the application itself.

All of these advances have made an enormous difference in the state of business application development today, compared to even a few years ago. By combining modern code generators with modern application servers, a relatively small team of developers can build and deploy a sophisticated business application in a short period of time. However, all of these developments have focused on simplifying and narrowing the tasks of the programmers.

Meanwhile, the user community has begun application development on its own. Almost every desktop in almost every business has a copy of Microsoft Access, and it has seen increasing use for business applications in companies everywhere. Individuals working in business specialties of the company, outside of IT, have developed small database applications in Access, based upon their understanding of the business that they deal with every day, which have been used by entire departments for some aspect of their work. These ‘rogue’ Access applications often demonstrate the need for custom application to address some area of the business, and are often used as models for the ensuing systems.

Advances in three areas of application development are making it increasingly easier to create and deploy custom software solutions. First, recent advances with application servers and their associated tools have resulted in more and more technical overhead functionality being embedded in third-party platforms. Secondly, code generators such as Computer Associates’ *Cool:Gen* are providing gains in development efficiency by automating redundant development tasks. Finally, the popularity of applications such as Microsoft’s *Visual Basic* has shown that database-savvy users can quickly design and implement simple database applications to address their business needs if given a graphical design tool they can understand.

Implementation of application functionality

The final part of this document will examine the functionality required to implement business applications and how Simplified Systems has addressed these areas in its product offering.

Database Functionality

- *Data Object Creation*.
  Users create the data structure through the object designer. The user must know to differentiate between a top-level table (no foreign key references), child table (one or more foreign key references, representing a 1-N relationship), and a join table (creating a link between two or more tables, representing an N-N relationship). The basic guidelines for deciding what table type to choose can be addressed in documentation and training. In the case of "child" and "join" table types, the user must specify the ‘parent’ object(s) for the table. Keys and referential integrity will be done automatically for the user, and cannot be modified by the user, preserving normalization. Columns are added by choosing a high-level type (such a ‘String’ or ‘Decimal’), and mapped to a database-specific type by the system.
- *Business Object Creation*
  Users can use the object designer to combine one or more tables into business objects, or views, of the underlying database. After choosing the initial table to view, the user is presented with a list of related tables that can be added as to the view. Any table that is related to any other table that is part of the database can be added to the business object. The user can also specify whether or not a table is read-only or editable within the business object; that is, whether or not that table can be modified, or is merely there to present information. Any of the columns of any of the tables can be removed from the business object, allowing the designer to hide data in specific objects.

- ***Transactional*** *Support*
  By default, every object (and all of its associated business logic) uses a '**flat transaction**'. There are programmatic mechanisms for bundling multiple operations into a single transaction, or launching a separate transaction outside of the current transaction. The Simplified product is not suitable for any system that requires the capability to perform nested transactions and nested transaction recovery; however, flat transactions are more than adequate for most business applications.
- *Locking*
  By default, every object uses an **optimistic lock** implemented by the use of a row version. Like flat transactions, this is sufficient for most business applications. A read-lock mechanism may be added in a future version.
- *Pooling*
  The translator server provides database pooling. A transaction is granted exclusive use of a connection for the duration of the transaction; data retrieval is always routed to an available connection. Additional connections are opened as required, and will be closed when use drops below a configurable threshold. In a deployment with multiple translators, connections will be pooled across multiple translators based on configuration usage thresholds.

Business Logic Functionality

- *Object Rules*
  Business logic runs on the middleware server based on user actions. All objects can run business logic after retrieving data (intended primarily to massage data for export to other systems), before inserting, updating, or deleting a record (to perform validation and manipulation of the data about to be saved), and after inserting, updating, and deleting (to create or modify data elsewhere in the application).
- *Rule Designer*
  The Rule Designer is a point-and-click user interface for designing business rules. The designer chooses the object that they are defining logic for, and can begin to define one or more rules for the object. The rules are presented to the user in a plain-English format; by clicking on underscored portions of the rule text, the user is presented with a list of valid options to choose from. The rules can contain basic arithmetic, simple or complex conditional statements, loops over a collection, or invocations on an object. The rule always contains the object that it is executing against, with full access to all of the data within the object. Additionally, the rule can access any other objects in the application for select, insert, update, and delete, enabling the rule to check or modify data elsewhere in the system. Finally, built-in functions are currently provided for manipulating strings and dates, performing more advanced mathematical functions, and gathering basic statistical information.
- *Rule Engine*
  The Rule Designer generates a propriety format for storing business rules,

which is executed by the Rule Engine on the server-side.

- *Object Descriptors*
  The Rule Designer uses a mechanism of *Object Descriptors* to present invocations on objects to the user in an intelligible manner. Analogous to *BeanInfos*, Object Descriptors will be an open API, allowing programmers to expose code libraries for use in the rule designer.
- *Custom Object Rules*
  If the need arises, custom Java classes can be written and attached to objects as rules. This would most likely be used for logic that is too complex for the Rule Designer, or to interface with other applications.
- *Custom Objects*
  As a last resort, a custom implementation can be written in Java for a specific type of object. This should only be needed for the business processes that are so complex that they cannot be implemented (or implemented efficiently) through the Rule Designer or the Custom Rule architecture.

User Interface Functionality

- *Desktop Client Application*
  One of the options for client applications is a Java-based desktop application. This application can be built entirely using the Application Builder user interface, and deliver an equivalent or better user experience than most custom-built user interfaces. Simplified includes a rich Swing-based component library that has been built to interact with the business objects, and provide all of the functionality that is found in the better custom business application user interfaces. If a more complex user interface is needed than the Application Builder can provide, custom Java code can be used for components of the application as needed.
- *Browser-based Client Application*
  Browser-based dynamic HTML clients can be built using the Web Builder user interface. The Java Servlet runtime architecture allows these interfaces to be served from the wide variety of server hardware and software configurations that support this architecture. The browser-based applications support the full level of functionality found in the desktop applications. Again, if a more complex user interface is needed, custom Servlet/JSP code can be used for parts of the application as needed.
- *Query/Display*
  All of the forms allow the end-user to perform custom queries to retrieve data. The designer has the option of choosing between Query-by-Example, where the end-user populates specific fields of the form with the values that they are searching for, or Search Screen, where the end-user is presented with a specialized form for creating complex queries using a range of comparison and clause-nesting options. For displaying the results, the designer may choose between displaying the data in a form, a table, a form and a table, or a form and table on a single screen.
- *Editing*

For editing the data, the designer may choose between editing in the display table, form, both, or by launching a separate form dedicated to editing. In addition to standard form components, text fields with attached calculators and calendars are provided for editing numeric and date fields, while data-driven combo boxes and tabular lookup fields are available for constrained values.

- *Related Data*
  The user interfaces provide a variety of options that can be used for accessing related data from any given form. The designer can place buttons on a form in order to launch another form displaying the relevant data for the current record. The designer may also place tables on the form that display the related records, and use these to interact with the related data itself.
- *Common Functionality*
  The designer may choose to place buttons directly on the form to control its actions, or create a common toolbar and menu system that are shared across all forms. The application can determine what form is active, which tasks can and cannot be performed, and automatically enable or disable the buttons or menus as necessary.

System Integration Strategies

- *Batch File Import/Export*
  The application does not currently support plain-text batch files import/export. XML is the preferred mechanism for batch-file transfer.
- *Shared Database Schema*
  While this is not recommended, it is possible for outside applications to read and write to a Simplified application's data structure. The Simplified middleware API also provides a mechanism for executing custom SQL statements directly against a database.
- *EDI*
  EDI-based transfers are not supported. The current trend in the industry is to migrate to XML, which is supported.
- *XML*
  The final release of the Simplified product will include a graphical tool for making XML on business object fields and actions, and a runtime for performing XML imports/exports in both batch mode and 'on-the-wire' using SOAP.
- *CORBA/RMI*
  The various components of the Simplified product communicate using CORBA, so CORBA access is readily available. The Java client library is available for use by external systems; a C++ client library is planned for a later release. The 'RMI-over-IIOP' technology shipping in the newer Java implementations should allow RMI-based clients access to the systems as well.
- *Custom APIs*
  The middleware server currently supports a 'Persistent Objects' API, which

allows programmers to make gateways to external systems available to custom business logic. This allows programmers to pool their external system connections, and keep them available for business logic. Additionally, through the use of Object Descriptors, these external APIs can be exposed through the Rule Designer if necessary.

- *Message-based Middleware*
  No explicit support for message-based middleware solutions is available at this time. Other mechanisms for access other systems can be used to implement a messaging-based integration.
- *Middleware Adapters*
  No third-party middleware adapters are available or planned at this time. If there is sufficient demand, this may be undertaken for a future release.

Definitions

1. Transaction

In the context of Simplified's product, a transaction is a sequence of information processing (such as updating a database record), treated as a unit, primarily for the purposes of ensuring database integrity. For a transaction to be completed and database changes to be made permanent a transaction has to be completed in its entirety; the operations are all or nothing. If an interruption or cancellation occurs before a transaction is successfully completed, all changes will be reversed. "Commiting" a transaction records its changes and makes it permanent. "Rollback" of a transaction brings things back to the way they were at the start of a transaction.

2. OLTP - Online Transaction Processing

An effective architecture for enterprise, large scale computing that enables and manages transaction-oriented applications (usually for data entry and retrieval). It involves tracking related database changes so that if anything goes wrong with any one of those changes, all associated entries made to that point could be reversed. Today's online transaction processing needs increasingly require support for transactions that span networks (and may include multiple organizations). Correspondingly, newer OLTP systems use client/server processing and brokering software that can allow transactions to run on different computer platforms across more than one network.

3. EDI - Electronic Data Interchange

Involves the exchange of structured data between two or more organizations in a form that allows automatic processing with no manual intervention. It is relevant to any business that regularly exchanges information such as client or company records, and is especially relevant for sending and receiving statements, payments, orders, and invoices. EDI allows transactions which have required paper-based systems for processing, storage and postage to be replaced and handled electronically - faster and with less room for error.

4. XML - Extensible Markup Language
A programming/formatting language similar to HTML. Unlike HTML (which defines information presentation), XML focuses on the structure of content, providing a mechanism to describe the organization and structure of data. The most important benefit of XML is that it defines a single syntax for documents and messages which can be shared on any platform.

5. RPC - Remote Procedure Call
A client-server based protocol that enables programs to request a service from a program located in another computer in a network without having to understand network details. (A procedure call is sometimes known as a subroutine, or function call.)

6. RMI - Remote Method Invocation
Enables programmers, using the Java programming language, to write object-oriented programs in which objects on different computers can interact in a distributed network. RMI is a Java-based version of the remote procedure call (RPC) protocol, with the added ability to pass one or more objects with the request.

7. CORBA - Common Object Request Broker Architecture
An architecture and specification for creating, distributing, and managing software objects over a distributed network. It allows programs at different locations, developed by different organizations, to communicate via an "interface broker." Similar to RPC and RMI it abstracts the network layer of communications, providing a higher level view for the programmer, in this case using an object oriented approach to programming distributed systems. This enables clients to request a service without knowing anything about what servers are attached to the network.

8. ORB - Object Request Broker
A component of the CORBA specification which provides a middle layer between clients and servers. In the CORBA model, different ORBs receive requests, forward them to the suitable servers, and finally pass the results back to the client.

9. N-Tier
Meaning “some number of tiers”, an n-tier application program is one that is distributed among three or more computers over a network. In addition to the advantage of distributing the data and programs over a network, such applications have the advantage that any one tier can be run on an appropriate

processor or operating system platform which can be updated independently of the other tiers. A common implementation of n-tier application is the 3-tier application, where the user interface resides on the user's computer, the business logic resides on a more centralized computer, and data storage on another computer (usually a ‘database server’).

10. Flat Transaction

An “all or nothing” proposition, this type of transaction that has only one layer of control by the application (as opposed to “chained” or “nested” transactions). Everything involved in a flat transaction either survives or is discarded: there is no way of committing or aborting parts of such transactions.

11. Optimistic Lock

Database locking in general involves the database preventing multiple users from accessing (reading or writing) the same record. This can in many instances cause detrimental database performance. Optimistic locking is a database record-locking scheme whereby a page of records is not locked until the user saves changes to the record. This is in contrast to Pessimistic Locking, where the page containing a record (and all other records in that page) is locked as soon as a user accesses that record. The advantage of optimistic locking is that when using this scheme, records are locked for only a very short period of time, minimizing potential locking “collisions”, also known as lock contention. The disadvantage is that more than one user can be editing the same record at the same time and the application must then deal with the situation when several users attempt to save different versions of a record. It is optimistic in the regard that it sees such a situation as the non-typical case. No user will be able to overwrite the data of other users without taking the others’ changes into account.

12. IIOP - Internet Inter-ORB Protocol

A specification created to enable CORBA functionality over the World Wide Web. Enables the exchange of complex data types, as opposed to simple text supported by HTTP.