Chapter 13: Distributed Objects

This chapter presents a case study of an airline reservation system using distributed object technology in a three−tier client−server environment. The discussion surrounding the case study focuses on customer requirements and constraints, distributed object technology, three−tier client−server environments, service objects, service object replication, partitioning, load balancing, and failover. A distributed object three−tier client−server development tool called Forté is referenced throughout this chapter.

Case Study 13−1: An Airline Reservation System (OnTime Airline)

Customer Requirements and Constraints

The requirement in this case study is to create a program to handle flight reservations and accept customers complaints. This system will also provide each customer service representative with an inbox for receiving internal memos from another application. A note window (Figure 13−1) will be developed to handle special customer needs (e.g., wheel chair access, a child traveling alone). A complaint window will allow customers to register informal and formal complaints, the distinction being that a formal complaint must be in writing. The system must be able to handle up to 100 customer service representatives making airline reservations for customers who phone in requests. A legacy system is not currently in place, but a flight schedule and fare information database exists.

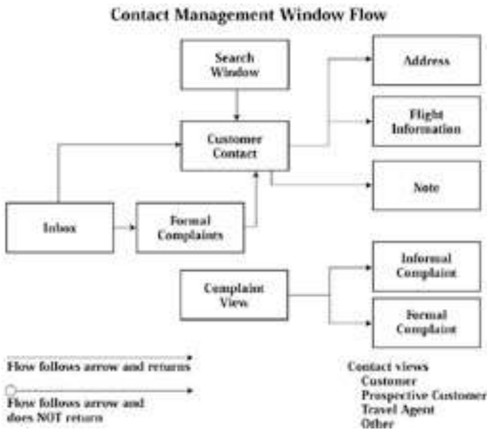


Figure 13−1: Window flow diagram

Distributed Object Technology

Distributed objects have the same properties as nondistributed objects, such as inheritance, abstraction, encapsulation, and polymorphism (overloading and overriding). However, distributed object technology allows objects to be distributed over many computers within a single environment. Objects in either a distributed or nondistributed application have a single location. A distributed application can provide access to many computers and external connections through a single integrated system. To interact with a particular object in a distributed environment, the application must interact with the computer on which the object is located; normally the object is located on the machine on which it was created. For example, if developer A builds the complaint component and developer B builds the customer contact component, then the code that developer B constructs must interact with developer As computer to access the complaint component.

Two− and Three−Tier Client−Server Environments

An application environment is a combination of hardware and software located at development and deployment sites. A two−tier client−server environment consists of the client (tier one) and server (tier two). A simple client−server application runs on two computersa desktop computer (client) and a database (server) as shown in Figure 13−2. For example, Forté can access multiple databases, residing on the same server and using a single database resource manager. One method to accomplish this task is by creating a file on the server, consisting of environment variables, and then accessing these environment variables as needed. Other examples consist of third−generation language applications and an application program interface (API). The phrase three−tier client−server environment refers to a client (tier one), a business layer (tier two), and a server (tier three) (Figure 13−3). Figure 13−4 illustrates the business class logic for this case study.

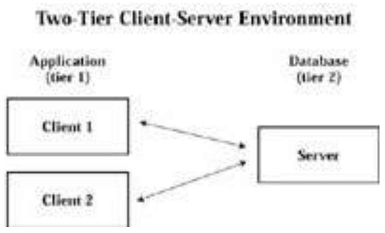


Figure 13−2: Client−server application

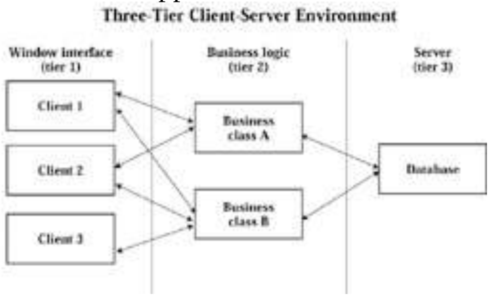


Figure 13−3: Various phases

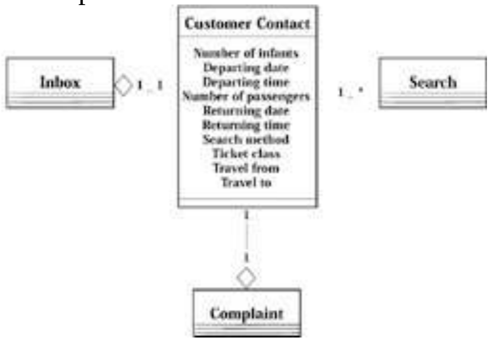


Figure 13−4: Class interaction diagram

Service Objects

A service object is a global object that can be accessed from any method within a program in the same manner as other objects. It is the means whereby an application communicates with external systems, such as databases or other applications. For example, a single database service object can be used to invoke multiple database sessions asynchronously.

Service objects can be replicated and distributed on multiple computers in the environment. For example, the customer contact window is displayed on up to 100 computers, and each computer can search the flight reservation database. Each computer must be able to register complaints on the complaint database located on the same server computer. The two objects used to interact with each database can reside on two different computers, thus reducing network traffic.

Shared objects have an automatic locking mechanism to prevent conflicts when multiple tasks attempt to access or change an objects state. The object is locked until the transaction is complete. Thus if two customer contact representatives are viewing the availability of seats on a flight, then the one who clicks the complete button first will hold the lock until he or she completes the commit sequence.

Service Object Replication

Replication is the process of copying an object to use it for failover and load balancing asynchronously. Once the object is replicated it can be partitioned onto other computers, either client systems or servers in the environment. The objects of interest to be replicated are service objects, such as the connection to an external corporate messaging application, and shared business services, such as the flight reservation database. Server partitions can also be replicated and applied for failover and load balancing synchronously.

Partitioning

A partition consists of one or more service objects. It decouples a distributed application from the details of its deployment in a runtime environment. In the partitioning process for this case study, the airline reservation application is configured for a specific target deployment environment. The application is divided into separate logical sections called *logical partitions.* These partitions are then assigned to the computers in the specified deployment environment. Each partition is an independent process that can run on a computer. For example, an application has a client partition on the desktop that provides the graphic user interface.

Other partitions include the DBMS server, on which flight reservation and customer databases reside, that runs on a server machine. The 3GL−message system service is also partitioned onto a different computer. Forté automatically coordinates all communication between the partitions. As new hardware architectures or additional machines are added to the environment, Forté can repartition an application to take advantage of the new capabilities without requiring an application rewrite. When a Forté runtime system starts up a partition, it creates several system objects that coexist with all of the application objects within the partition. These runtime system objects support the running of each partition on its respective platform and the functioning of distributed objects as a single application. If a service object creates other objects, then they will all be located in the same partition. For example, if a service object is used to obtain flight information from a flight reservation database, then the object that was created to hold this information will be on the same partition as the service object that created it.

Load Balancing

A service object, such as the flight reservation database manager object, can be replicated any number of times for load balancing. Load balancing is the use of multiple copies of a service object, located on different computers, for simultaneous use by a multiple number of clients. A service object must be replicated to provide load balancing. That replicate can then be installed onto either the computer on which it was created or a different computer in the environment. Placement of the flight reservation database service object onto several computers reduces bottlenecks. Forté automatically provides a router partition that coordinates the requests for a service to the next available object, parallel processing, and coordinates of connections to all copies of a service object. For load balancing, parallel processing can be provided by distributing the demand for a service among several replicates of the partition that provides the service.

Failover

A service object can also be replicated any number of times for failover, which is a means to provide a backup service object to be used if the primary object fails. Failover provides built−in−fault tolerance for an application. The process for location of replicates of a service object onto different computers is the same for failover and load balancing. However, Forté will only maintain a connection to the primary service object. If that fails, then the secondary replicate will be accessed until the secondary service object will not be used. In the case of combined failover and load balancing, the router partition is replicated so that one is the primary router and the replicate is the secondary router. If the primary router partition fails, then the secondary router partition automatically becomes active.

Use Cases

Use cases are provided by the customer. The following is a sample use case for generation of a flight confirmation number.

Use Case General Information

- ***Title.*** Handle customer telephone call
- ***Functional area.*** Contact management
- ***Author.*** Bob Jones
- ***Business contact.*** Joe Smith
- ***Update date.*** 3/11/98
- ***Business context.*** A customer calls an OnTime Airline customer service representative to book a ticket for a flight
- ***Actors.*** Customer service representative and supervisor
- ***Overview.*** The actor enters all relevant information needed to book a flight

Use Case Analysis Information

- ***Preconditions.*** The actor wishes to reserve a seat for a customer using the customers desired flight destination, date, and time.
- ***Basic course.*** The actor provides the customers flight destination, date, and time to the system and then searches the system to confirm that the flight date and time exist. If the flight exists, then the system will return all available seat numbers. Based on the customers preferences, the actor reserves the desired seat number. The system issues an associated confirmation number.
- ***Postconditions.*** The actor has reserved a seat number for the customer.
- ***Alternate course.*** When the actor cross−references the flight destination, date, and time and finds no flights, then the system returns a message asking the actor to re−enter the flight information.
- ***Business rules.*** If the actor makes three unsuccessful attempts to reserve a seat on a flight, then he or she suggests alternative routes consisting of connecting flights.

Use Case Design Information

Use Case Design Information

Field:	Field Name:	Type:
Requirements	First name	String

Middle		String
Last name		String
Address		String
City		String
State		String
Zip		TextData
Home phone		TextData
Returning date		DateTimeData
Travel from		String
Travel to		String
Search criteria		Integer
Office phone		TextData
Total number of passengers		Integer
Number of infants		Integer
Departing date		DateTimeData
Travel class		Integer

- ***Interface elements.*** From the customer contact window (Figure 13−5), the actor can navigate to any of the other windows. The customer contact window offers customer information and a flight confirmation number (Figure 13−6).

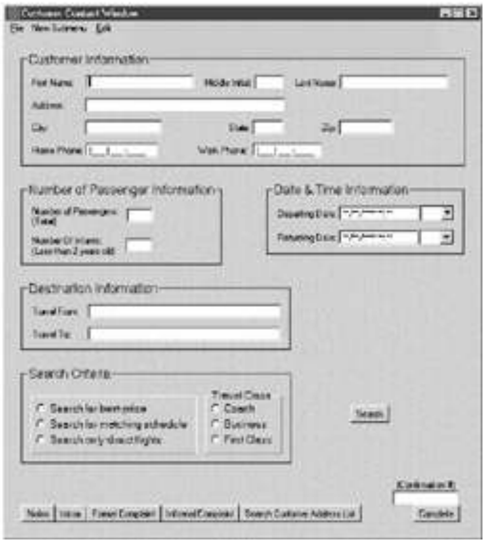
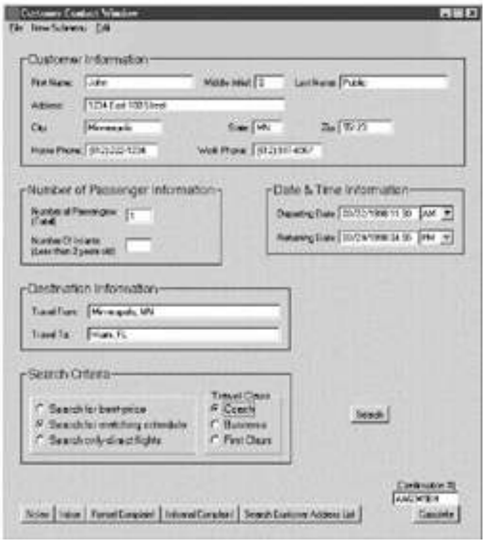


Figure 13−5: Customer contact window



Customer Information

First Name:	John	Middle Initial:	[illegible]	Last Name:	Public
Address:	1234 East 100 Street				
City:	Minneapolis	State:	MN	Zip:	[illegible]
Home Phone:	[illegible]	Work Phone:	[illegible]		

Number of Passenger Information

Number of Passengers (Total): 1

Number Of Infants (Less than 2 years old):

Date & Time Information

Departing Date: [illegible] AM

Returning Date: [illegible] PM

Destination Information

Travel From: Minneapolis, MN

Travel To: Miami, FL

Search Criteria

- Search for best price
- [✓ Search for matching schedule]
- Search only direct flights

Travel Class

- [✓ Coach]
- Business
- First Class

Confirmation ID: [illegible]

Figure 13−6: Flight information

- ***Preconditions.*** The actor intends to search the flight reservation systems database.
- ***Basic course.*** The customer contact window is displayed on the screen and has a file menu bar at the top and a toolbar at the bottom of the screen. The file menu has the following items:

  Save. Saves the data on the current active screen

  Print. Prints the data on the current active screen; the windows default printer is used to print a bitmap image of the window

  Close. Closes the current active screen

The toolbar provides short cuts to other windows or menu items. When the customer contact window is being displayed, there are validations for the following fields:

Departing date and time. This date should be the same as today or later than today.

Arriving date and time. This date should be later than or same as departing date.

Destination information. Both cities and states must exist in the database.

- ***Postconditions.*** The actor returns a confirmation number to the customer.
- ***Alternate course.*** The actor searches for flight information with the given information. The database returns more than one possible match. The actor redefines the search criteria and then selects a flight that is suitable to the customer.

Discussion

The flight reservation system application is platform independent; the same program can run on any supported client system. Thus this distributed application can be deployed in either a single environment or multiple environments; however, the effort in deploying in multiple environments consists of system administrative overhead (Forté Software, Inc., 1800 Harrison Street, Oakland, CA, 94612).

For more information concerning Forté and their associated product line, please refer to the Forté Internet site at www.Forte.com.

Born Information Services Group, 445 East Lake Street, Suite 120, Wayzata, MN, 55391.