Abstract

The integration of networked applications, especially those requiring continuous access to shared information resources and databases, may be affected by scheduled unavailability of one or more of these applications. The scheduled unavailability usually occurs due to a number of reasons including application or data backup, software or hardware maintenance for the application’s platform, executing periodic processes such as file reorganization or end-of-period processes, application maintenance, or application migration. This paper provides an object-oriented framework to increase the availability of integrated applications without fully replicating the application environment such as the application platforms, programs, and data. The proposed framework can be used to maintain the integration of networked applications during scheduled unavailability of one of the applications. In addition, the proposed framework enables highly available applications integrations while maintaining data and state consistency of the applications. This framework provides a cost-effective solution for increasing availability of networked applications integration.

Keywords: middleware, highly available systems, systems integration, object-oriented framework

1. Introduction

Integration middleware has become one of the important infrastructure components in many information technology departments. With the existence of robust integration middleware infrastructures, new applications can be easily developed and integrated with existing applications. Using integration middleware, the degree of information assurance and integrity within any sub-system can be easily maintained. In addition, the process of information exchange among the applications is performed in a reliable way. This allows companies to maintain their investment in high-cost legacy systems by using middleware to integrate them with new business services and functions.

Service reliability and availability is an important component of the information assurance requirements [4]. The quality of many business functions in many organizations relies on the existence of highly reliable and available computer services. The degree of quality of applications such as business-to-business and e-commerce is highly affected by the degree of reliability and availability of the supporting computer services. For example, the information flow in e-commerce services start from the user’s browser, through the Internet infrastructure, to the web server of the company that provides the services, and to one or more back office applications within the company. One of the main issues related to having high quality of services for such e-commerce applications is the availability of integration among the different applications involved in the system.

The main challenge in the integration process among networked applications is the integration availability. While new applications must be available continuously, many legacy applications were designed to work for a certain number of hours and suspend for other hours of the day. Some legacy applications are periodically suspended for data backups, end-of-period processing, system and software upgrades, and/or maintenance. These scheduled suspensions are usually not acceptable for high quality service-oriented applications. Therefore, some efforts were invested to solve this problem using server redundancies and full replication of databases. Moreover, some researchers have also worked on optimizing these methods to achieve better performance as in [8], [3], and [6]. However, these techniques are costly since they require full replication of the software, databases, and hardware. In addition, they cannot solve some of the problems in legacy systems such as unavailability due to database updates conducted during end-of-period processes. This is a result of the legacy systems requiring exclusive access to the databases to ensure consistency and integrity. Therefore, general replication techniques cannot be used effectively.

Highly available integration was discussed earlier [11] in the context of banking systems, where the basic concept was used to increase the availability of the e-banking services. In this paper we introduce an object-oriented framework that provides a generic approach to active object replication. We present an object-oriented cost effective replication technique that can be
used for increasing the availability of networked application integration during a scheduled unavailability of one or more involved applications. This technique is based on the replication of selected objects of the unavailable applications in temporary storage. These objects can replace the suspended application during the unavailability time. In addition, the proposed technique provides a mechanism to recover the new transactions performed on the replicated objects later to the main application without suspending the integrated application services.

In this paper, background information on application integration and middleware is discussed in Section 2. Section 3 discusses the issues of applications integration availability then Section 4 describes the new approach for enhancing integration availability of networked applications. Finally, Section 6 concludes the paper.

2. Background

Many organizations have a number of new and legacy business applications that are built by different vendors. These applications are built to support specific business functions and generally rely on some form of information resources to store and use the data needed for the business functions. These applications are usually built using different programming languages and they work on different hardware and operating system platforms. Some of these applications are new while others are older legacy applications. Examples of new applications are e-commerce, business-to-business (B2B), e-Government, and electronic banking systems and of legacy applications are accounting, banking, payroll, customer order, products management, and stock inventory control systems. The new business applications are not usually built from scratch and they normally rely on the functions and information resources of the existing legacy applications.

One of the main challenges for these organizations in how to integrate these applications [9][10][14]. Integration among applications is needed for many reasons including, to exchange and share information among the applications, to reuse functions and services provided by other applications, and to introduce new functions to an application. For example, Internet banking system needs to be integrated with a customer accounting system to process the transactions requested by the customer. Customer order systems need to be integrated with stock inventory control systems in order to exchange information about the availability of merchandise items. Therefore, having a good integration among these applications not only maintains information integrity and consistency but also save a lot of operational efforts and costs. The difficulty and complexity of applications integration is owed to several factors:

1. The applications may be distributed on different machines with heterogeneous architectures and operating systems.
2. The applications may have different interfaces. For example, newer applications may use XML, Remote Method Invocation, CORBA, and DCOM interfaces. While old applications may use CICS and nonstandard text messaging interfaces.
3. The applications may run at different speeds. For example application X can send 10 requests per second to application Y, which is only able to process 4 requests per second. The requests in this case may be lost or significantly delayed.
4. The applications may have different operational and transactional requirements. For example a request from an application should be processed as a single transaction by two different applications.
5. The applications may have different availability modes and needs. For example, e-commerce applications are designed to work continuously - seven days a week, 24 hours a day - while many legacy systems such as accounting systems will only operate during standard business hours.

As a result, the integration process of the new business applications with the legacy applications has become an important and complex task. Middleware commercial products such as BEA Tuxedo [2], IBM WebSphere MQ [7], and Software AG EntireX [13] are used to facilitate the integration process and provide the necessary functionalities to ensure reliability and integrity among other requirements. These products deal with applications as black boxes through their advanced programming interfaces (APIs) where each box consists of unknown application modes and databases. These middleware products solve the first four of the challenges listed above. For example most of these products support different types of platforms and operating systems. Some of these products provide mechanisms and development tools to integrate applications that support different types of interfaces. For example EntireX allows windows based applications to transparently use CICS transactions in a mainframe using DCOM interface. In addition, most of these middleware products provide brokering services to add reliability for the integration. Furthermore, most of these products support distributed transactions.

Some of these products also provide persistent services to partially solve the fifth mentioned point. The persistent service in middleware provides a mechanism to store a sent message in persistent storage whenever the receiver application is not available. The stored messages can be recovered later when the application becomes available. This type of service provides an easy way to allow integrated distributed applications to continue their operations. However, it cannot provide a solution for integrated distributed applications that require instant request/reply communications unless application and data replication is used. When full replication is available, the Integration middleware can be used as a router for requests between the original application and the replicated one. The introduction of web services [15] also provided a good method of integration; however, their application is limited to web-based functions. Some researchers introduced web services that could withstand some types of system suspensions such as software upgrades and maintenance [12]. However the solution does not
cater for situations were data stores are involved and exclusive access to those stores is needed during the suspension period.

3. Applications integration availability

Service availability is an important aspect of information assurance requirements [4]. In addition, one of the challenges in the integration process between new applications and legacy applications is the integration availability. As discussed earlier there are several causes for scheduled suspension of some applications. These scheduled suspensions are usually not acceptable for high quality service applications. Therefore, some efforts were invested to solve this problem using full applications redundancies and full replication of the databases. This solution requires complete software, hardware, and data replication which is very costly and not affordable by many small-size organizations. In addition, they cannot solve some problems in legacy systems such as unavailability due to conducting updates for end-of-period processes. This is due to the fact that most legacy systems require exclusive access to the databases, which requires all replicas to be suspended at the same time. Therefore general replication techniques cannot be used in such cases.

To explain the problem, consider the application in Figure 1. We have two integrated applications X and Y. Function A is part of application X and function B is part of application Y. Executing function A in application X requires executing function B in application Y. Assume application X is designed to be available continuously while application Y is can be available only for 22 hours a day. This is due to backup procedures, end-of-day process, or any other off-line application related functionality. As a result function A in application X will only be available for 22 hours a day.

There are many integration cases among applications where they need to exchange information with other applications by executing specific function APIs. In most of these cases, the number of these functions and the information needed for the integration represent a small subset of the whole application functions and data. For example, in any bank, there is the main banking system. This system contains information about all customers, accounts related to the customer, accounts related to the bank branches, loans, time deposits, foreign currencies, etc. When we integrate web applications that provide Internet-Banking services with the main banking system, only a small set of the functions and data from the main banking system will be used. Therefore, a backup system is needed for the main banking system to provide the services and data for the Internet banking system during the unavailability time of the main banking system, only the needed functions and data of the main banking system need be replicated.

The availability in applications integration can be solved if there is a full or partial replication for the functions provided by the unavailable server application and some mechanism to ensure data consistency at all times. Consider that application X is integrated with server application Y where application Y provides some services to application X. The mechanism of integration can be through remote procedure calls, request/reply text messages, XML messages, Web Services, etc. Consider that the set of functions provided by application Y to application X is \( \{f_1, f_2, f_3, \ldots, f_n\} \). These functions can be classified into two types: read-only and update functions. The read-only functions do not change the state of the application, while update functions do change or alter the state of the server applications. Since read-only requests do not affect the state of the database in application Y, then the solution is straightforward. Simple replication techniques will provide the required availability. The problems occur when updates are made on the data in application Y and this is the main focus of this paper.

Consider application Y has a backup version \( Y' \) in which each application has their own database and machine as shown in Figure 2. In the normal case, the integration middleware forwards application X requests to application Y. If application Y is not available, the middleware can forward application X requests to the backup version \( Y' \).

![Figure 2. A case of two integrated applications.](image_url)

One of the main issues with the update functions is to maintain data consistency between applications Y and its backup \( Y' \). Replication between application Y and its backup can be achieved using two methods:

1- Before switching off application Y, the up-to-date data of application Y should be manually transferred to application \( Y' \). This can be done by interrupting application X requests and the middleware and copying the latest data from application Y to backup application \( Y' \). After the copying process, application X requests and the middleware can resume their operations with application \( Y' \). After application Y becomes available, the journal of application \( Y' \) should be recovered to application Y. During the recovery, the new requests from application X should be suspended. The suspension of application X requests is needed to maintain consistency of the data during the replication and recovery processes. The result of this approach is the unavailability of
the application Y or Y' for same duration it would take to transfer the data from Y to Y' and vice versa. This time will greatly depend on the size of the databases and the number of transactions taking place, which could be very long for large scale applications.

2- Having a data replication process between applications Y and Y' continuously running such that Y' will always have an up-to-date copy of the data. Therefore, whenever application Y needs to be suspended, the middleware directly forwards the new requests from application X to application Y'. At a later stage when application Y becomes available, the journal of application Y' can be recovered to application Y. During the recovery process, the requests from application X should be suspended to maintain the order of the updates. In this case, the unavailability of the application will be reduced to that of the time it takes to recover the data from Y' to Y.

In both methods, there is suspension time, where the integration between application X and application Y or its backup Y' is unavailable. This suspension time is needed to maintain the consistency of the data.

4. Active replication

Active Replication provides reliable mechanisms to replicate the needed subset of application functions and data before suspending the integrated application. This replication is done by creating a number of objects that emulate the functions of the unavailable application. These objects can provide the same services provided by the suspended application. Therefore, before any integrated application is suspended, replicated objects for the needed functions are created. The types of these objects depend on the main integration class which is discussed in the Section 4.1. The main advantage of the service is that it allows increasing the availability of integration without fully replicating the unavailable application and without suspending the service provided by other applications during the information copying or recovery processes. Here we discuss the solution architecture and describe the mechanisms of maintaining availability and data consistency of the integrated applications.

4.1 Identifying the main integration class

The set of functions provided by a server application to the clients can be related to one or more real-life object types. For example, application Y may provide functions related to customers, courses, accounts, or orders to application X. Application X can view these objects in Application Y as a set of objects related to one or more class types. Each of these object types can be considered as the main integration class. In each class type, multiple related functions can be performed. For example, in banking systems, the main banking system may provide functions to request and manipulate customer accounts for other systems such as the electronic banking systems [1][5]. For each customer, the main banking system provides interfaces for different functions such as account balance inquiry, account transfers, cash withdrawal, and bill payments. All these functions are related to the customers. Therefore, the customer class can be identified by the integration middleware as the main integration class with the main banking system. Within that main class subclasses may be defined. For example under a customer object, multiple account objects can be defined. In general, if an application provides information about customers to other applications, then the middleware takes the customer class as the main integration class for this application. Other applications can get information for different customer objects through that integration middleware. Each customer object provides complete services to replace the suspended application with regard to a specific customer. Therefore, to verify if the customer class is the main class we need to verify that all functions in the main application (e.g. the main banking system) used by other applications for a specific customer can be done within a single customer object related to that specific customer.

The approach used here depends on identifying the main integration classes, which provide a number of essential functions for other applications. The main integration classes define the main object types used for the integration and can be easily identified in distributed object applications. In distributed applications implemented using distributed object middleware such as CORBA, objects are distributed in multiple machines. Each set of objects in a machine can represent an application. Client objects in one of the integrated applications can invoke methods in server objects and receive responses from these server objects. Both the invocation and response actions are achieved through messages sent across the network. The server objects class can be classified as the main integration class.

If an application is not implemented using the object-oriented approach, the integration middleware can still view the services provided by this application as one or multiple integration classes. An integration class for any application represents and combines all services related to the application. In this paper to simplify explanation of the active replication service we will consider a single integration class case.

4.2 The solution architecture

The solution can be embedded in any integration middleware platform. The solution consists of a set of components including emulated integration objects creation and storage, emulated application process, transaction logs, and request forwarder. In addition, for each integrated application, there are two processes: object-based copying and object-based transaction recovery. The first process is used to transfer the function of a specific application to the active replication service and the second process is to recover transactions that were completed during the suspension period.
4.2.1 Emulated integration objects creation and storage. To have the active replication service as part of any integration middleware, the middleware administrator needs to define the integration class for the application that may be unavailable for the other applications in the system for certain periods of time. This integration class should be implemented by the middleware administrator. In this class the data structure for the integration class must be defined. In addition, a number of methods or functions should be defined. Each of these methods represents a specific function provided by the integration class. For example, for the main banking system, the main integration class is the customer class. When this class is integrated with other applications such as Internet Banking, then the methods needed to be defined in this class are for getting a list of customer accounts, account balance, account transfer, account balance enquiry, mini-statement, etc. All these functions are needed by the Internet Banking system from the main banking system. Therefore, the created class should also have a method to support each function needed by the other applications. For each method there is input and output. The input can be considered as the request sent by the other applications. The output generated from calling that method is the response that needs to be sent back to the application that sent the request. For example, a request for checking an account balance should contain the customer and account numbers. This request can be considered as an input for the method related to the balance enquiry function. The output generated from calling that method is the response that is needed to be sent to the application made the request.

The defined class must also contains a default and public method update(msg). This default method will be called by the active persistent service to update the data structure of the customer object. The user needs to implement that method to update the object fields. Java can be used to implement integration classes.

Emulated integration objects are stored in the object storage. Each emulated integration object has a unique key and a state. Each emulated integration object can have one of two states, active or inactive. Active state means that the object can be used to serve a request. Inactive state means the object can not be used and a request should be forwarded to the original application which will serve the request. For example, in the banking environment if a request that belongs to a specific customer is coming to the middleware, the request can be served by the corresponding object if the object is active. Otherwise, the request will be forwarded to the main banking system which will serve the request.

4.2.2 Emulated application process and transactions log. This process invokes the appropriate method at the specified emulated integration object if a request for that integration object is received. The output of that method will be returned as a response for the request. The application emulation process will receive other applications' requests if and only if the requests belong to the active emulated integration objects. If a request belongs to an active object then, the request will be performed by the application emulation process and the request will be recorded on a transaction log. That transaction log keeps all requests conducted with active objects for later recovery.

4.2.3 Requests forwarder. The request forwarder is a process that checks the state of the integration object when a corresponding request arrives. In a normal operation mode when a server application is available, all related emulated integration objects are inactive. When the request forwarder receives a request for an object with a specific key, it checks the state of the corresponding object. All requests for inactive objects should be forwarded by the request forwarder to the main server application, while all requests for active objects should be forwarded by the request forwarder to the emulated application process. If the object for a specific request is not available in the object storage, the object can be considered inactive and all its corresponding requests should be forwarded to the original server application.

4.2.4 Object-based copying. The object-based copying is a process that relates to a specific server application. A user can start this process before switching off the server application. This process copies the information of the integration objects from the server application to the object storage. For each integration object, an emulated integration object will be created in the object storage. This copying is done by the active replication service object by object. For example, if the main integration object is the customer, then an object will be created for each customer. The list of the customers will be taken from the server application as the first step to create all integration objects. After creating each customer object, the object will be immediately activated. At any given time during the execution of that process some integration objects will be active while others will be inactive. Therefore, the process of applications integration will continue without any suspensions. An application request will be served by either the active replication service if the corresponding objects are active or by the original application if the corresponding objects are inactive. The process of copying individual objects usually takes a few milliseconds. During that copying, any request belonging to that object will be suspended during the copying process. This is to make sure that all changes because of the requests that belong to the integration object will be conducted in consistent manner. Any changes to the data of that object or customer are done either in the original
application or to the data structure of that object at the object storage of the active persistent service platform.

4.2.5 Object-based transaction recovery process. This process can be executed by the user when the original application is ready to be used again. This process recovers the transactions completed during the suspension period. The recovery is done based on the integration objects. All transactions belonging to a specific integration object are recovered at the same time. During that recovery, the middleware will suspend any new request for that specific individual integration object. This is to maintain the data consistency for that integration object as explained in the previous subsection. After transaction copying of a specific integration object, the object will be made inactive such that any new request for that integration object will be directed to and performed by the original application. Copying a single object will take a very short time thus will generally be unnoticeable by the requesting applications or in the worst case will be perceived as a minor delay.

5. Conclusion

This paper discussed an object-oriented approach to enhance the availability of integrated distributed applications. The approach utilizes the object-oriented design to achieve its goals. A technique to identify the integration classes and actively replicate and use these classes was introduced. When an application needs to be suspended, the other integrated applications can rely on the integration middleware and the replicated integration objects to continue their operations. Objects for each instant of the integration class are created in the middleware with the most recent state and accompanying data made available. The integrated applications will continue operations normally, while the middleware decides to provide the service or forward the request to the main system depending on the status of the integration object.

The approach is very useful for any type of application integration involving some applications that cannot execute continuously while other applications require continuous availability. Examples are banking systems, online shopping stores and airline reservation systems. In the examples listed here the model includes some components that must be available 24-7 (twenty four hours a day and seven days a week), while some components are only available for a limited time of the day (e.g during working hours or all day except the times for updates and backups). In addition any of these systems will periodically require to be suspended for maintenance, upgrade, or migration, which will require active replication to avoid lengthy unavailability of the services.

References