Journal of Computer Science 4 (2): 95-102, 2008
ISSN 1549-3636

A Middleware Service for Increasing Applications Integration Availability

Nader Mohamed and Jameela Al-Jaroodi
The College of Information Technology, UAE University, P.O. Box 17551, Al Ain, UAE

Abstract: Middleware has become an integral part of many distributed applications offering effective integration and interoperability solutions. In some situations a problem may occur where the integration of distributed information applications may be affected by scheduled unavailability of one or more of these applications. The scheduled unavailability occurs due to several 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 research introduces a new middleware service called Active Persistent Service. Unlike regular middleware persistent services, this active service can provide consistent responses instead of the unavailable applications for other applications. This service helps increase the integration availability of distributed applications without fully replicating the application environment such as application platforms, application programs and application data. The proposed service can be used to maintain the integration among distributed applications during scheduled unavailability of one of these applications. In addition, the proposed service enables highly available applications integrations while maintaining the data and state consistency of the applications. This service provides a cost-effective solution for increasing the availability of applications integration.

Key words: Middleware, highly available applications, distributed information applications

INTRODUCTION

Integration middleware[6,10,17,20] 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 department 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 new business services and functions with these systems.

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 quality of applications such as business-to-business and e-commerce applications 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 distributed information 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 suspended for some time for data backups, end-of-day processing, end-of-month processing, system and software upgrades, maintenance and/or end-of-year processing. 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[3,7,9,11]. However, these techniques are

Corresponding Author: Nader Mohamed, The College of Information Technology, UAE University, P.O. Box 17551, Al Ain, UAE Tel: +971-3-7135519 Fax: +971-3-7672018

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.

In this research we introduce a new middleware service called Active Persistent Service (APS). APS is an object-oriented framework that provides a generic approach to active object replication. This service utilizes an object-oriented cost effective replication technique to increase the availability of distributed applications 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.

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. 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 while the 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 of the existing legacy applications.

One of the main challenges for these organizations is how to integrate these applications[12,14,21]. 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 systems need 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 the merchandise. Therefore, having a good integration among these applications not only maintains information integrity and consistency but also saves a lot of operational efforts and costs.

The difficulty and complexity of applications integration is attributed to many reasons:

The applications may be distributed on different machines with heterogeneous architectures and operating systems.

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. Integrating applications with different interfacing technologies is always difficult and challenging.

The applications may run on different machines with 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 if there is no mechanism to provide reliable communications and buffering techniques between them.

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.

The applications may have different availability modes and needs. For example, e-Commerce applications are designed to work continuously 7 days a week, 24 h a day, while many legacy applications such as accounting systems were designed to only operate during the 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[10] and Software AG EntireX[19] 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 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. In the case where full replication is available, the Integration middleware can be used as a router for requests between the original application and the replicated one.

APPLICATIONS INTEGRTAION AND 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. 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. Some legacy applications require to be suspended for a specific amount of time for periodic operations. These scheduled suspensions are usually not acceptable for high quality service applications. 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 of the problems in legacy systems such as unavailability for updates during end-of-period processes. This is due to the fact that most legacy systems require exclusive access on the databases, which renders all replicas to be suspended at the same time. Therefore general replication techniques cannot be used in such cases.

To explain the problem further, consider the applications case in Fig. 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. Consider that application X is

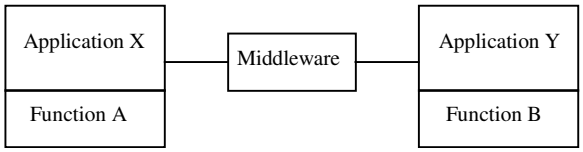

Fig. 1: A case of two integrated applications

designed to be available continuously 24 h a day while application Y is designed to be available for only 22 h a day to allow exclusive time to perform backups, end-of-day process and any other off-line application related functionalities. As a result function A in application X will only be available 22 h of the day.

There are many integration cases among applications where they need to exchange information with other applications through 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 which contains information about all customers, accounts related to the customer, accounts related to the bank branches, loans, time deposits, foreign currencies, cheque books, traveler's cheques, 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 to 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 mechanisms 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 may 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 functions and update functions. The read-only functions do not change the state of the server 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 straight forward. Simple replication techniques

Fig. 2: An integrated application with it replica

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 research.

Consider application Y has a backup version Y' in which each application has their own database and machine as shown in Fig. 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'. One of the main issues with the update functions is to maintain data consistency between applications Y and its backup Y'. Data replication between application Y and its backup Y' can be achieved using two methods:

- 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
- 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 consistency of the data by maintaining 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 application data.

ACTIVE PERSISTENT SERVICE

Active Persistent Service provides reliable mechanisms to replicate the needed subset of application functions 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. Before any integrated application is suspended, replicated objects that emulate the needed functions of the application are created. The types of these objects depend on the main integration class which is discussed next. 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 copying or recovery processes. Here we discuss the solution architecture and describe the mechanisms of maintaining availability and data consistency of the integrated applications.

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 systems may provide functions to request and manipulate customer accounts for other systems such as the electronic banking systems[1,5,8,15,16,18]. 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 in this research depends on identifying the main integration classes. These integration classes provide a number of functions for other applications. The main integration classes define the main object types used for the integration. Integration classes 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 a method in a server object. The client objects will receive responses from the server object. Both invocation and responding 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 research to simplify the explanation of active persistent service we will consider a single integration class case.

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 persistent service and the second process is to recover transactions that were completed during the suspension period.

Emulated integration objects creation and storage: To have active persistent service as part of any integration middleware, the middleware administrator needs to define the integration class for the application that may be unavailable for some 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 to a number of methods or functions. 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, the methods that need 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 basically 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 current account balance that needs to be sent to the application that made the request.

The defined class must also contain 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 the 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.

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.

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.

Object-based copying process: 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 persistent 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 suspension. 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 few milliseconds. During that copying, any request that belongs 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 a 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.

Object-based transaction recovery process: This process can be executed by the user when the original application is ready to be used again. It 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 data consistency for that integration object as explained earlier. After transaction copying of a specific integration object, the object will be inactivated such that any new request for that integration object will be performed by the original application. Copying a single object takes a very short time and will be generally unnoticeable by the requesting application or in the worst case will be perceived as minor delay.

APPLICATION

A real-life implementation of the proposed solution was used at Al-Ahli Bank, Bahrain, to increase the availability of the e-banking services. The e-banking system at Al-Ahli Bank includes ATM, POS and Internet banking. The Bank's main banking system runs on an AS/400 while the ATM/POS controllers run on RS/6000 and the Internet banking controller runs on another AS/400. The proposed solution was used as a part of the integration among these different applications. Before implementing this solution, the service was interrupted for at least 45 min on a daily basis for the end-of-day process. The proposed solution eliminated the service interruption while maintaining the reliability, security and integrity. In addition, the proposed solution is used when software or hardware maintenance on the banking system or machines is required. Long software and hardware maintenance operations are performed during weekends and holidays

```
Class Customer
{  // Integration object data
   private int customerNumber;
   private int noofAccounts;
   private int[] accountNumber;
   private int[] accountBalance;
   public Customer()
      {  // Integration object initialization
      }
      // Integration object functions
      public reply account_balance(String request)
         { ……
         }
      public reply account_statement(String request)
         { ……..
         }
      public reply cash_withdrawal(String request)
         { …….
         }
      public reply account_transfer(String request)
         { …….
         }
}
```

Fig. 3: Customer integration class

without interrupting the e-banking services. A customer class was used as the integration class with the main banking system, (Fig. 3).

Another application that the Active Persistent Service was used for is during the migration of the main banking system from an IBM Mainframe to an AS/400 machine. All e-banking systems were connected to the main banking system in the IBM Mainframe through a middleware with active persistent service. The migration of the main banking system was a three-day process. It was planned to take place during a three-day official holiday during which only the e-banking services were needed to be available for the customers. To avoid any interruptions for the e-banking services, the active persistent service was used to replicate all needed integration objects from the mainframe before starting the migration process. After three days, the migration process completed and a transaction recovery process was executed. This process moved the three-day transactions to the new banking system in AS/400 also using the APS technique. Therefore, the three-day migration was done without any interruptions of the e-banking services. At the same time, the consistency of the data of the main banking systems was maintained.

CONCLUSION

This research discussed a new middleware service to enhance the availability of distributed application integrations. This service is called Active Persistent Service (APS). APS is based on the object-oriented paradigm and 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 continue operations normally, while the middleware decides whether 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, airline reservation systems and many others. In the examples listed here the model includes some components that must be available 7-24 (7 days a week and 24 h a day), 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, upgrades, or migration, which will require active replication to avoid lengthy unavailability periods of the services.

ACKNOWLEDGMENTS

A primary version of this research was presented at The IEEE International Conference on Information Reuse and Integration, Las Vegas, USA, August 2007. This research is an extended version.

REFERENCES

1. Asokan, N., P. Janson, M. Steiner and M. Waidner, 1997. State of the art in electronic payment systems. IEEE Comput., (9): 28-35.
2. BEA Tuxedo web page, 2008. http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/tux/.
3. Bowen, N., D. Sturman and T. Liu, 2000. Towards continuous availability of internet services through availability domains. In: International Conference on Dependable Systems and Networks (DSN 2000), pp: 559.
4. Cummings, R., 2002. The evolution of information assurance. IEEE Comput., pp: 65-72.

5. Egner, F., 1991. The Electronic Future of Banking. Naperville, Illinois: Financial Sourcebooks.
6. Emmerich, W., 2000. Software engineering and middleware: A roadmap. In: Proceedings of the Conference on The Future of Software Engineering, (ICSE 2000), Future of SE Track, ACM Press, pp: 117-129.
7. Felber, P. and P. Narasimhan, 2004. Experiences, strategies and challenges in building fault-tolerant CORBA systems. IEEE Trans. Comput., 53 (5): 497-511.
8. Furche, A. and G. Wrightson, 1996. Computer Money, A Systematic Overview of Electronic Payment Systems. Heidelberg, Germany: Dpunkt-Verlag fur Digitale Technologie GmbH.
9. Garcia-Molina, H. and B. Kogan, 1988. Achieving high availability in distributed databases. IEEE Trans. Software Eng., 14 (7): 886-896.
10. Geihs, K., 2001. Middleware challenges ahead. IEEE Comput., (6): 24-31.
11. IBM WebSphere MQ web page, 2008. http://www-306.ibm.com/software/integration/wmq/.
12. Krishnamurthy, S., W. Sanders and M. Cukier, 2002. An adaptive framework for tunable consistency and timeliness using replication. In: International Conference on Dependable Systems and Networks (DSN'02), IEEE, Washington, D.C., USA.
13. Hasselbring, W., 2000. Information system integration. Communications of the ACM, 43 (6): 32-38.
14. Lee, J., K. Siau and S. Hong, 2003. Enterprise integration with ERP and EAI. Commun. ACM, 46 (2): 54-60.
15. Lipis, A. and T. Marschall, 1985. Electronic Banking. John Wiley and Sons, Inc., New York.
16. Lynch, D. and L. Lundquist, 1996. Digital Money. The New Era of Internet Commerce, John Wiley and Sons, Inc., New York.
17. Milojicic, D., 1999. Middleware's Role, today and tomorrow. IEEE Concurrency, 3: 70-80.
18. Sciglimpaglia, D. and D. Ely, 2002. Internet banking: A customer-centric perspective. In: Proceedings of the 35th Hawaii International Conference on System Sciences.
19. Software AG EntireX web page, 2008. http://www.softwareag.com/entireX/.
20. Vinoski, S., 2002. Where is middleware?, IEEE Internet Comput., (2): 83-85.
21. Vojdani, A.F., 2003. Tools for real-time business integration and collaboration. IEEE Trans. Power Syst., 18 (2): 555-562.