Managing Fault Tolerance Transparently using CORBA Services

René Meier and Paddy Nixon

Department of Computer Science, Trinity College, Dublin 2, Ireland
{Rene.Meier, Paddy.Nixon}@cs.tcd.ie

Abstract. Fault tolerance problems arise in large-scale distributed systems because application components may eventually fail due to hardware problems, operator mistakes or design faults. Fault tolerance mechanisms must be employed to reduce the susceptibility of a given system to failure. In this paper, we describe the design of an architecture to overcome potential application component failures, using *CORBA*, a distributed object middleware specified by the *OMG*. Of primary importance to this architecture is *OMG's CORBA Object Trading Service* as the mechanism to advertise and manage service offers for fault tolerant application components. This mechanism enables clients transparently to detect a failed connection to a service object, to discover a similar backup service object and to re-connect to it. This improves overall system stability and enables scalability.

1 Introduction

Fault tolerance problems [4], associated with the use of large-scale distributed systems [1][6], arise because application components may eventually fail due to hardware problems, operator mistakes or software faults [2]. Within most environments, and in particular within a banking environment, such failures are not acceptable.

The author [5] identifies four possible causes of application component failure [6] in a large corporate banking system, which is described in Section 2. All of them can be overcome by providing a fault tolerance mechanism that re-connects a client from a failed service to a similar backup service. In Section 3, we introduce such a mechanism, which is described more completely in [5]. It is essential that the suggested mechanism fit into the existing banking system. The prototype implementation of the fault tolerant architecture was successfully tested and evaluated as shown in Section 4.

2 The Target Banking System

In this Section, we present an existing large corporate banking system, into which the suggested fault tolerance architecture must fit. Although the proposed architecture was designed to fit into this banking system, it can easily be applied to any similar client-server based architecture.

The banking system is based on a three-tier client-server architecture. It is implemented using CORBA [2][3][7], a distributed object middleware specified by the OMG [7]. Clients are grouped together in locations accessing service objects in the middle-tier (service-tier). Service objects are logically associated with locations and may be located on one or more servers. The servers access a distributed database (database-tier) that is kept consistent using a data replication protocol.

The client-tier and the middle-tier are of particular concern. The middle-tier includes the *Service Manager* and the *Notification Service*, two components that will be used in the design of the suggested architecture.

- The *Service Manager* maintains the status of each of the service objects, using *pings*, requests that ask if the service is still running. It will shutdown and / or try to restart any service objects that have failed. The status of the service objects is used by the Notification Service to publish service object status notifications. The Service Manager guarantees the service objects to be fail silent.
- The *Notification Service* publishes several different types of notifications. A particular notification type is published in the event of a service object status change. To receive notifications, clients register with the notification type they are interested in.

2.1 Problem Statement, Requirements, and Assumptions

The banking system lacks a mechanism to re-connect clients from unavailable master services to backup services. Master services may not be available because of failures[1] or for maintenance reasons. Currently, banking system clients connected to an unavailable master service are not able to retrieve data requested by a user, despite the presence of other similar services (i.e. in another location) that could provide the requested data. The basic assumption is that re-connection delays and even a lower access performance are acceptable, but a total loss of a service is unacceptable.

It is essential that the suggested architecture fit into the existing banking system. Fault tolerance issues must be *hidden* from the client application program and therefore from the user and must also be Object Request Broker (ORB) [7] *independent*. The suggested architecture must support *on-line* application component management by configuration adjustment, without rebooting the rest of the system. It should provide a highly available system with a good trade-off between *system scalability* and *service performance*.

It is assumed that the architecture of the given banking system includes the service manager and the notification service components and that it adequately addresses single point of failure.

[1] Application component failures can be caused by service, server or network failures. Furthermore, an application component might be unavailable due to maintenance.

3 Managing Fault Tolerance

To solve the fault tolerance problems identified in the previous sections, we propose an architecture that introduces the OMG Object Trading Service [7] to the existing banking system as the mechanism to advertise and manage service offers for fault tolerant application components. The basic architecture is a simple low cost solution that solves the fault tolerance problem without involving the Notification Service. Clients query the Trading Service for master and backup service offers and cache the retrieved service references. If the service in use fails, the service user (client) is re-connected to the backup service. During re-connection, the service user is idle. Then, the client starts pinging the original service and re-connects the service user to it as soon as it is back on-line. Pinging is inefficient and causes unnecessary network traffic. To eliminate this, an *improved architecture* is proposed, that makes use of the Notification Service and a new component: the *Trading Service Manager*. This improvement lets clients use notifications to detect service failures and therefore eliminates the need to ping services[2]. This results in less network traffic and significantly reduced service user idle time due to re-connection in background. The Trading Service Manager also uses notifications to detect service failures and is responsible for marking invalid service offers in the trader.

A more detailed description of the basic and the improved architecture, its components and the configuration of the OMG Object Trading Service is omitted due to the limited space but can be found in [5].

4 Implementation and Evaluation

A prototype of the basic fault tolerance architecture was implemented in Java using Iona’s OrbixWeb and OrbixTrader. Client side fault tolerance algorithms were implemented as smart proxy classes, in order to hide them from the client application program. Unfortunately, the smart proxy feature is ORB vendor specific. An ORB-independent way to implement a fault tolerance algorithm is to place it between the IDL interface and the client application program, thus removing transparency.

The prototype implementation was successfully tested on both Solaris and Windows NT platforms. During integration testing, services were killed to force clients to re-connect to backup services and to re-connect to the original service, as soon they were re-started. To evaluate the performance of the implemented prototype, we measured the duration of method invocation on a service object running on a remote server with and without (using bind) the fault tolerance mechanism. Furthermore, we measured the duration of re-connection to the backup and re-connection to the original service object running on a remote server. It was observed that invocation time on a bound service object and on a fault tolerant service object is *essentially identical*. Re-connection to the backup service object (worst case < 2 seconds) and re-connection to the original service object (worst case < 15 seconds) may both cause service user idle time. The measurements show that user idle time is

[2] Instead of several clients having to ping the same service, the service manager pings the service and informs client on status change.

within an acceptable range. The worst case is less than 15 seconds, as opposed to minutes or even hours due to a temporary or total loss of a service in absence of a fault tolerance mechanism.

5 Conclusions

This paper has described an architecture to transparently manage fault tolerance in a large-scale distributed system. The presented architecture is designed to fit into an existing banking system and allows the development of fault tolerant application components. Of primary importance to our solution is the inclusion of the OMG CORBA Object Trading Service into the fault tolerance architecture as the mechanism to advertise and manage service offers for fault tolerant application components. The mechanism enables clients transparently to detect a failed connection to a service object, to discover a similar backup service object and to re-connect to it. The architecture allows application component management by configuration adjustments, without re-booting the system, supports the different needs of the banking system's clients and adequately addresses the scalability requirements of the system's infrastructure.

A prototype of the suggested mechanism has been implemented and evaluated. The implementation shows that performance issues are appropriately addressed and that performance can be improved by completely implementing the architecture. The limitation of the implementation is the trade off that had to be made between fault tolerance hiding and ORB independence. In order to hide fault tolerance algorithms from the client application program, an ORB specific feature, the so called smart proxy, was used. This issue is subject of further research. In conclusion, it has been demonstrated that this architecture supports the development of fault tolerant distributed application components, which results in *improved availability*.

Acknowledgements. René Meier completed this project as part of the M.Sc. in Networks and Distributed Systems at Trinity College Dublin and is now a Ph.D. student in the Distributed Systems Group funded by the Higher Education Authority (HEA) of Ireland.

References

[1] J. Bacon, *Concurrent Systems.* Addison-Wesley, 1993.
[2] M. Banâtre and P. A. Lee, *Hardware and Software Architectures for Fault Tolerance.* Springer Verlag, 1994.
[3] S. Landis and S. Maffeis, *Building Reliable Distributed Systems with CORBA.* Theory and Practice of Object Systems, John Wiley, New York, April 1997.
[4] P. A. Lee and T. Anderson, *Fault Tolerance: Principles and Practice (second edition).* Springer Verlag, 1990.
[5] R. Meier and P. Nixon, Managing Fault Tolerance Transparently using CORBA Services. Technical Report TCD-CS-1999-05, University of Dublin, Trinity College, February 1999. http://www.cs.tcd.ie/publications/tech-reports.
[6] S. Mullender, *Distributed Systems.* Addison-Wesley, 1993.
[7] Object Management Group. http://www.omg.org.