A Framework Providing Fault Tolerance Using the CORBA Trading Service.

A thesis submitted to the
University of Dublin, Trinity College,
in partial fulfilment of the requirements for the degree of
Masters of Science (Computer Science).

René Meier Inf. Ing. HTL,
Department of Computer Science,
Trinity College, Dublin.

September 1998.

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this or any other University, and that, unless otherwise stated, it is entirely my own work.

_____

René Meier,

September 1998.

Permission to Lend and/or Copy

I, the undersigned, agree that the Trinity College Library may lend and/or copy this thesis upon request.

_____

René Meier,

September 1998.

Thesis Summary

This thesis describes a body of research into the *fault tolerance problem* associated with the use of large scale distributed systems and a partial solution to the problem. Fault tolerance problems arise in such an environment because application components may eventually fail due to hardware problems, operator mistakes or software failures.

Within most environments, in particular within a banking environment, such failures are not acceptable, thus fault tolerance mechanisms must be employed to reduce the susceptibility of a given system to failure.

This thesis provides a framework to support the development of fault tolerant distributed application components within an existing banking environment, such as the Jetpac infrastructure of WDR London.

Potential application component failures, in the Jetpac infrastructure, were identified and an architecture to overcome these failures, using *CORBA*, a distributed object middleware specified by the *OMG*, was designed and implemented. Of primary importance to this architecture is *OMG's CORBA Trading Service* as the mechanism to advertise and manage service offers for fault tolerant application components.

Fault tolerance mechanisms were hidden from the client application program and therefore from the user. Quality of Service (QoS) in terms of performance was adequately addressed and evaluated.

The prototype implementation of the suggested architecture fits into the Jetpac infrastructure and provides a mechanism to (re)discover and (re)connect master and backup application components at run time. This helps in overall system stability and scalability, providing a *highly available system*.

Acknowledgements

First and foremost, I wish to thank my supervisor Dr. Paddy Nixon for his valued advice and assistance throughout the course of this research. I would also like to thank Warburg Dillon Read for supporting my work, John Nichol for his valuable input.

Thanks also to Paul Stephens for helping to proof read this script.

Finally, thanks to my family, friends and colleagues for their support during the execution of this thesis.

CONTENTS

1. INTRODUCTION

This thesis deals with an investigation into the *fault tolerance problems* associated with the use of large scale distributed systems. Problems arise in such an environment because application components may eventually fail due to hardware problems, operator mistakes or software faults.

Within a banking environment such failures are not acceptable, thus fault tolerant mechanisms must be employed to reduce the susceptibility of a given system to failure.

In undertaking this research, potential application component failures, in an existing distributed system, were identified and an architecture to support the development of fault tolerant distributed application components using *CORBA*, a distributed object middleware specified by the *OMG*, was designed and implemented. Central to this architecture is *OMG's CORBA Trading Service* as the mechanism to advertise and manage service offers for fault tolerant application components. Finally, the Quality of Service (QoS) in terms of scalability and performance of the suggested architecture was assessed.

1.1 Motivation

Application component failures in large scale distributed systems are inevitable. Within most environments, in particular within a banking environment, such failures are not acceptable. Thus fault tolerant mechanisms must be employed to reduce the susceptibility of a given system to failure.

A service in a distributed system is called fault tolerant when it behaves according to its specification even in the presents of failures in parts (processor, media, communication link, other service) of the distributed system on which it depends [Mullender, 1993].

Types of faults that affect a distributed system are :

Type of fault	Real World Example	Well-known technique to handle the fault
Processor faults	Machine or operating system crashes	Reboot using checkpointing / logging or recover critical state into another process
Communication faults	Lost, corrupted or duplicated messages	Time-out and retransmission
Media faults	Disk head crashes	Keep multiple copies of the data ("backup")
Process faults	Resource shortage, software bug	Anticipate likely problems and write exception handlers or write multiple copies of the application
User "aborts"	Control "C"	Anticipate likely problems and write exception handlers

Table 1-1. Types of Faults

This thesis provides a framework to make application components in a banking environment, such as the *Jetpac infrastructure* of WDR London, fault tolerant.

The Jetpac infrastructure, a three tier based client-server architecture, that is implemented using CORBA, a distributed object middleware specified by the OMG, already provides fault tolerance regarding some of the types of faults identified above. It includes several databases, that are kept consistent using a *data replication protocol*. Servers and their services are managed by a so called Service Manager (SM), that, together with the appropriate hardware, guarantees them to be fail silent.

The Jetpac infrastructure already provides low level fault tolerance mechanisms. However, it lacks a mechanism that reconnects clients from unavailable master services to backup services. Master services might not be available because of failures or for maintenance reasons. Currently, Jetpac clients, connected to an unavailable master service, are not able to retrieve requested data at all, despite the presents of other, similar services, that could provide the requested data. Reconnection delay and even a lower access performance are acceptable, but a total loss of a service is unacceptable.

The goal of this dissertation is to design and implement an architecture to support the development of fault tolerant distributed application components within the Jetpac infrastructure. Central to the project is the use of the *OMG Trading Service* as the mechanism to advertise and manage service offers of fault tolerant application components.

It is essential that the suggested architecture fits into the existing Jetpac infrastructure. Fault tolerant issues must be *hidden from the client application program and therefore from the user* and be *Object Request Broker (ORB) independent*, providing a highly available system with a good trade off between scalability and performance. The suggested architecture must support on-line application component management by configuration adjustment, without rebooting the rest of the system.

1.2 Objectives

The overall objective of this work is to explore one possible solution to the problem outlined in Section 1.1 and detailed in Chapter 2. This is broken down into four main steps.

- Collect all required information from WDR regarding the Jetpac infrastructure. Re-evaluate the Jetpac infrastructure and examine the role of each component in it. It will become evident that not only services or servers can fail, but that services need to be updated from time to time and therefore be taken out of service during the working day.
- Investigate the available CORBA, OrbixWeb3.0 and OMG Trading Service features. Because the default OMG Trading Service's matching algorithm is rather simple, a majority of the service selection algorithm must be located on the client side. This increases switching performance due to pre-fetching and caching of the backup service object references. OrbixWeb features, so called smart proxies, allow the manual implementation of proxy classes that are hidden from the client application program. Although other ORB's support similar features, smart proxies are Iona specific.
- Integrate the OMG Trading Service into the Jetpac infrastructure. Service offer property has to be defined to enable service offers to be discovered by clients at run time. Furthermore, service offer property must support overall system scalability, easy service maintenance and enable eventual system extension in terms of load balancing. To support these needs in an effective way, service offer property will include a Master-Service list and Backup-Service lists. Each list contains unique client group identifiers, for which clients query to discover their service offers.

- Design and implement a client smart proxy class, an interface to the Trading Service and a service offer query algorithm to discover service offers and to select the most appropriate one. These classes have to be hidden from the client application program whenever possible. The service's Interface Definition Language (IDL) interface has to be extended to include these features.

The essence of this thesis is :

By integrating the OMG Trading Service into the Jetpac infrastructure, a mechanism is introduced to enable clients to (re)discover and (re)connect services at run time. This increases overall system reliability and scalability.

This mechanism is provided as a software framework including client side support libraries, Trading Service query and service offer selection algorithm. These algorithms include intelligent pre-fetching and caching to reduce user idle time during reconnection.

A fully functional prototype of a fault tolerant client-server application has been implemented using this framework. The implementation will be discussed in Chapter 4, 5 and 6.

1.3 Results

The primary result of this work is the design and implementation of a framework to provide fault tolerance within the Jetpac infrastructure. It is based on the use of OMG's CORBA Trading Service as the mechanism to advertise and manage service offers for fault tolerant application components.

The secondary result of this work is a performance analysis of the prototype implementation of the proposed design.

1.4 Roadmap

Chapter 2 lays the foundation of this thesis by re-evaluating the Jetpac infrastructure and examining the role of each component in it. This is followed by the identification of possible causes of application component failures. Finally, the requirements to enable fault tolerance to be build into the Jetpac infrastructure are examined.

Having explored the problems in detail, Chapter 3 is a survey of fault tolerance related issues and object middleware solutions. This Chapter also includes application scenarios that evaluate CORBA features.

Chapter 4 presents the design of the fault tolerance mechanisms introduced to the Jetpac infrastructure. A basic architecture and improvements on it, that build fault tolerance into the Jetpac infrastructure, are presented.

The implementation of the proposed architecture is described in Chapter 5 and its performance is evaluated in Chapter 6.

Finally, Chapter 7 summarises the achievements of this work and possibilities for future research.

2. CUSTOMER REQUIREMENTS

2.1 Introduction

This Chapter introduces the WDR *Jetpac infrastructure* and its components, which is a WDR Interest Rates system. Jetpac is a not a specific product but an infrastructure and collection of components which will be utilised in different configurations to provide the functionality currently provided by a number of existing Interest Rates systems.

The description of the Jetpac infrastructure and its requirements is based on a paper submitted by WDR as well as on notes taken by the author in several meetings and discussions with members of the Jetpac development team.

2.2 Jetpac Terminology

Server A server is a physical machine, usually a Sun Solaris machine with several processors.

Service A service is a software application component running on a server.

Location A location is a collection of services running across one or more servers, providing functionality to a community of users.

2.3 The Jetpac Infrastructure

Jetpac is a *three-tier service based architecture*. The client tier is based on WindowsNT, the server component is a Sun Solaris machine that runs a SYBase database. The services in the middle tier of the Jetpac architecture are developed in *Java*. For the communication link *Iona's OrbixWeb3.0* was used until the very end of this thesis when OrbixWeb3.0 was replaced by *Visigenic's VisiBroker*. Services are collected together into locations. Locations may share some services such as the Trader (this is the Jetpac trader service, not to be mixed up with the OMG Trading Service) and Exception Reporting service. There are likely to be 10's of services per location in the long term. Currently there are five per location.

Multiple locations can co-exist on the same physical machine, although it is also possible for locations to be located on separate machines. For example, we might have several locations in London. Each location is identified by a name such as LondonCorporates and LondonGilts.

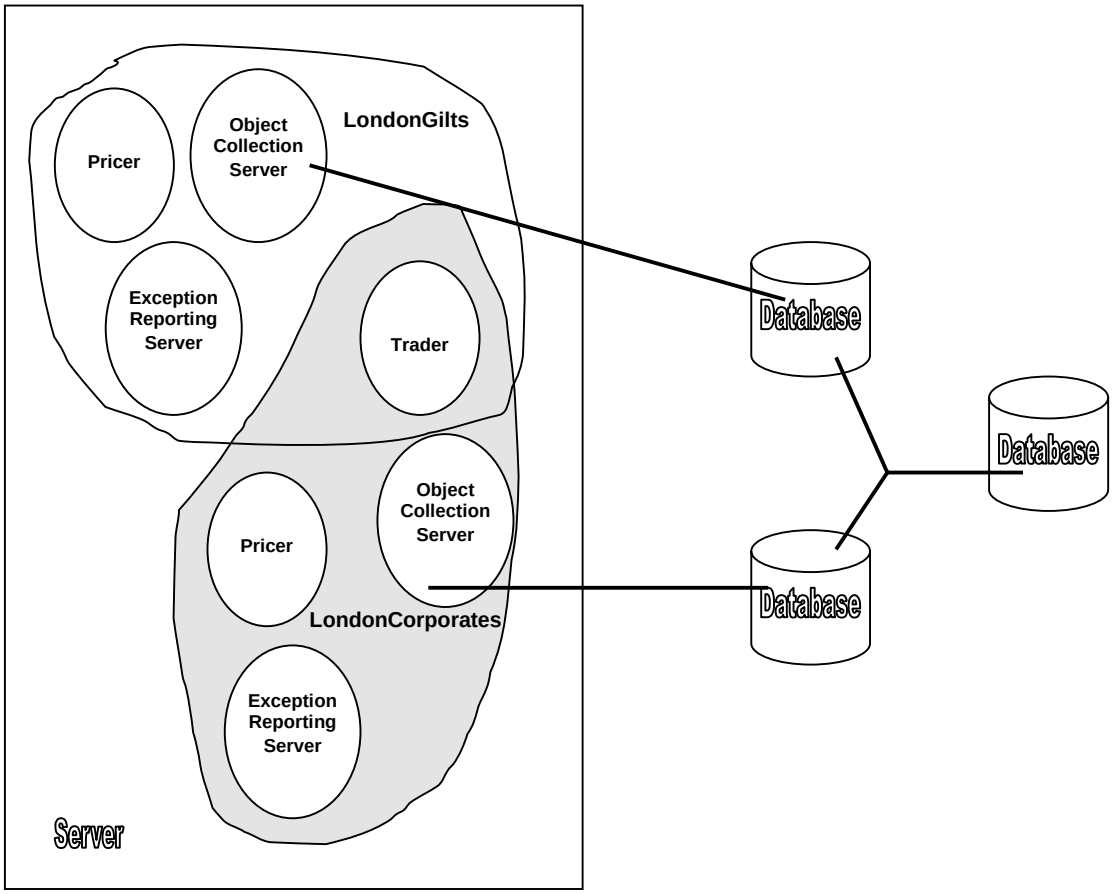

Figure 2-1. The Jetpac Infrastructure.

Each location accesses a database. There could be several databases associated with a location. The databases are kept in sync using a *database replication protocol*. More than one location can use the same database.

The configuration of the system is very flexible. An example of a high performance configuration, is where a desk, say 6 Traders, has its own location, which is the only location resident on a powerful Sun Solaris server box. The location may be the only one accessing a particular powerful database.

Another example is where a hundred users spread across 3 locations share a Sun Solaris server.

The Jetpac architecture has a *notification mechanism* at its heart. This is used to publish notifications which are then delivered to all subscribers that have registered interest in a particular type/class of notification.

In the initial release of Jetpac, each user connects from a client to a particular location of services and gets data based upon filters that are user specific. The data is then displayed at the client. The client will receive notification of any changes that occur to the data set.

2.3.1 The Jetpac Three-Tier Client Server Model

The former Jetpac infrastructure was implemented based on a *two-tier client server architecture*.

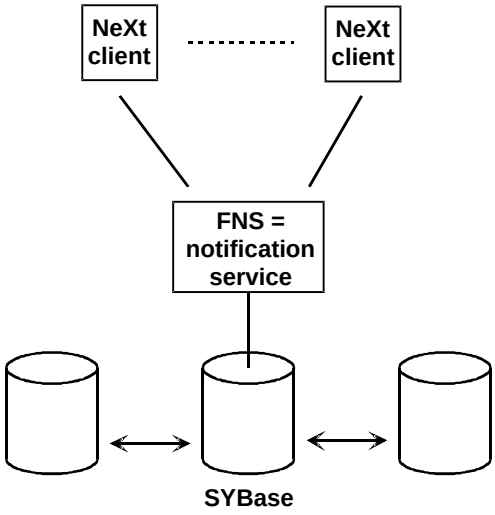

Figure 2-2. The Former Two-tier Jetpac Architecture.

The Sybase databases are kept in sync using a database replication protocol. A consistent change is made within a few seconds. The FNS notification service updates the NeXt clients, which cache, process and display the data.

The actual Jetpac infrastructure is implemented based on a *three-tier client server architecture*.

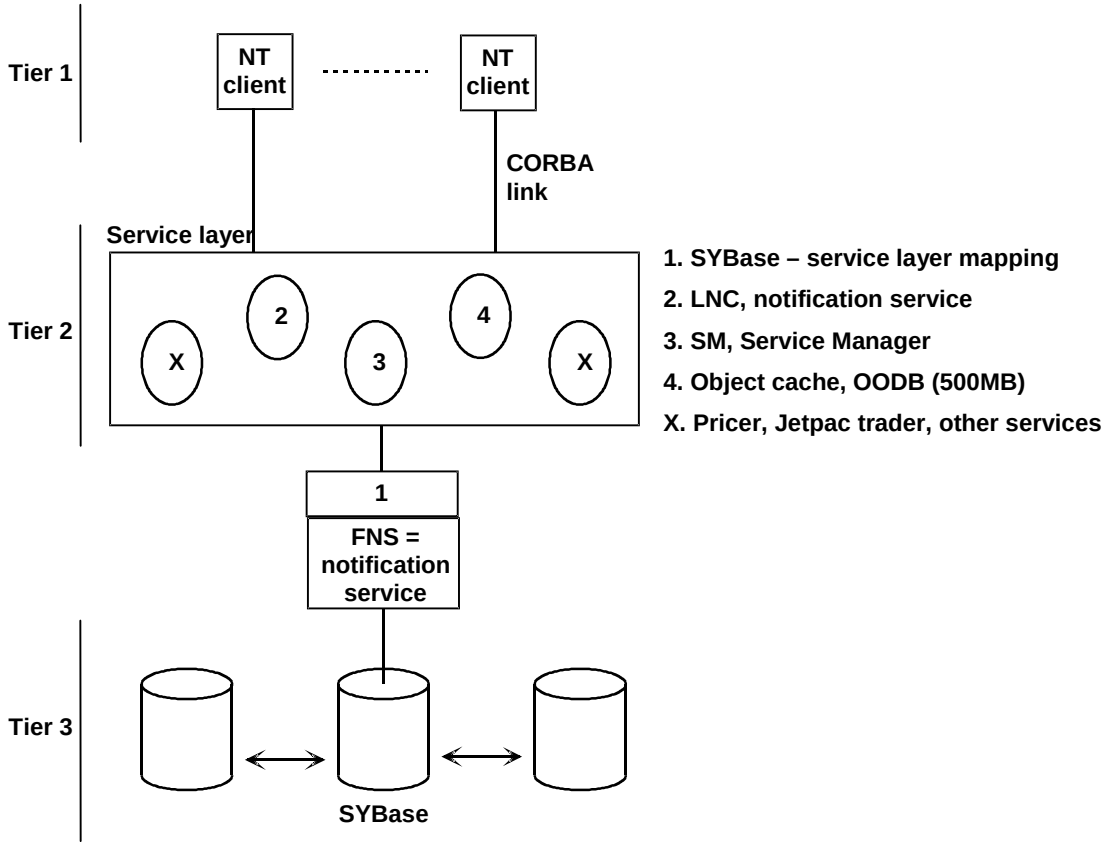

Figure 2-3. The Actual Three-tier Jetpac Architecture.

The client and the middle tier (service layer) are of further concern in this project. The database tier is provided and does not need to be improved regarding fault tolerance. Within the middle tier, the features of the Service Manager (SM) and the notification service (LNC) will be used in this project.

- The *Service Manager* (SM) maintains the status of each of the services, using pings, a request that asks if the service is still running. It will shutdown and / or try to restart services that failed. The status of each of the services will be used by the notification service (LNC) to publish service status notifications.

- The *notification service* (LNC) publishes several different types of notifications. To receive notifications, clients register with the notification type they are interested in. The LNC currently supports user generated notifications as well as database generated ones, which are updating the data set cached in clients. Service Manager generated notifications, which are essential for this project, will be supported by the Jetpac infrastructure in the near future.
- The *object cache* contains the relationship between clients and objects. Its main task is to update objects. The boot (or reboot) time of the object cache is approximately 40 minutes.

2.4 Application Component Failures

Four possible causes of application component failures, within the Jetpac infrastructure, were identified.

1. *Service failure* — A service can fail. The service may be used by clients or other services in one or more locations.
2. *Server failure* — A Sun Solaris server can fail. This will effect services in one or more locations.
3. *Network failure* — A part of the network can fail. This may result in disconnecting some of the services in a location from each other, and / or services from clients.
4. *Service maintenance* — Services will need to be upgraded from time to time. This might involve taking some services, used in one or more locations, out of service during the working day.

2.5 Requirements

The in Section 2.4 identified errors must be *hidden from the user* as much as possible, providing a *highly available system* without resorting to purchasing expensive fault tolerant hardware. In order to do so, procedures, algorithms and support libraries must be provided to enable fault tolerance to be built into the Jetpac infrastructure.

The fault tolerant Jetpac infrastructure must be able to support *client offices* (group of users) as well as *locations* with different needs.

On one hand, there are “large” main client offices, which can afford to buy redundant hardware. When the master server fails, clients must be reconnected to the standby backup server. On the other hand, there are “small” client offices, with limited resources, which have to be reconnected to a server of another client office. The latter will have to deal with considerable reconnection delays and increased response times due to potential lower bandwidth and increased server utilisation.

Each location is specialised for a particular part of the business. A location may only hold a small subset of objects or may not cache objects at all and can therefore not become a backup for another business part. In some cases, a location may cache additional objects when becoming an active backup for another location.

This requires that, besides fault tolerance, in terms of service *reliability* and *availability*, Quality of Service (QoS), in terms of *scalability* of the infrastructure and *performance* of services, is the main concern of this project.

To enable the development of application components that can be easily made fault tolerant, we need to have :

1. A framework into which fault tolerant application components must fit.
2. Software to support the integration of application components into the framework.
3. Client-side algorithms and libraries to enable the client to detect, or be informed, of failures and take action on them.
4. Configuration support to enable fault tolerant application components to be deployed.
5. A monitoring system to determine when failures have occurred and to notify interested parties via the Jetpac notification service (LNC).

One component of the fault tolerance framework will be the *OMG Trading Service*. The Trading Service provides functionality to allow services to be registered with properties. Properties will include fault tolerance status, e.g. Master, Backup, Location, etc. It is anticipated that the Trading Service will be a central part of the Jetpac infrastructure, brokering service offers between servers and clients.

2.6 Assumption

Based on the customer requirements, the following assumptions were made.

- The main aim of this thesis is to provide a framework to solve the fault tolerance problems identified in this Chapter. Hardware architecture related fault tolerance issues will not be considered. Other fault tolerance issues, such as *database consistency*, *Transaction Processing Monitors* (TPM), *fail silent services*, etc, are not addressed in this project and are expected to be included in the Jetpac infrastructure by other means.
- Server and services are considered to be fail silent. It is the Service Managers (SM) task, supported by the appropriate hardware, to deal with this.
- The loss of a service may affect a group of users as well as other services, whereas the loss of a client only affects a single user. Therefore, service fault tolerance will be considered as critical, whereas client fault tolerance will be considered as non-critical.
- To support a possible customer environment change, the project development has to be portable in terms of platform independence and whenever possible ORB independence. The project will be developed in Java, which guarantees platform independence, and will make use of Iona's OrbixWeb3.0 ORB and Orbix implementation of the OMG Trading Service, both of which are interoperable.
- The suggested framework includes one Trading Service only. Trading Service replication will be handled as with other Jetpac services, such as notifications service (LNC) or Service Manager (SM). Therefore, Trading Service replication and Trading Service federation will not be considered in this thesis.
- To make use of the available resources, all development work will be based on a single machine.

2.7 Summary

Four possible causes of an application component failure were identified. All of them can be overcome by providing a mechanism to reconnect a client from a failed service to a similar backup service.

Building such a mechanism into the Jetpac infrastructure will result in a highly available system.

The suggested mechanism must enable the Jetpac infrastructure to support all the different needs of its clients and locations, in terms of scalability and performance. Therefore, the OMG Trading Service will be a central part of this mechanism, advertising and managing service offers of fault tolerant application components.

3. BACKGROUND RESEARCH

3.1 Introduction

This project first reviews related work on *fault tolerance in software*. A software component is said to fail when it does not provide the behaviour in its specification. Fault tolerance means to make a system more *dependable*, this is to minimise the occurrence of system failures. To make a software component fault tolerant means that an error has to be identified and corrected. Four main schemas for software fault tolerance are identified.

Then, related work on *fault tolerance in distributed systems* is reviewed. Types of fault in distributed systems and well-known techniques to handle them are discussed. Furthermore, a classification of failures, according to the behaviour of a server, is introduced. Attention is paid to systems such as Isis and concepts such as transaction processing.

In distributed systems, *object middleware* applications are becoming more essential for software development. These object middleware applications basically provide services that describe how client applications can invoke on server objects. OMG CORBA and DCOM are such middleware applications. OMG CORBA and DCOM, including their services, are introduced and compared. Particular attention is paid to the OMG Trading Service that may manage replicated services. There are other OMG CORBA features that may be used to provide fault tolerance in a distributed system. Finally, several OMG CORBA applications are evaluated to explore these features.

3.2 Fault Tolerance in Software

Today's computer systems (set of components and a design) are becoming larger and more complex and suffer from the effects of faults which, if not tolerated, lead to dependability problems. These faults are caused by hardware problems, operator mistakes or software faults [Banâtre and Lee, 1994].

Analysis of computer system failures, such as [Gray, 1993] suggests that in general hardware problems are no longer the main cause for computer system failures. This is because the hardware on which the software executes becomes more and more reliable. Hardware problems and operator mistakes will not be considered further in this thesis.

A system is said to fail when it does not provide the behaviour prescribed in that systems specification. Fault tolerance means to make a system more dependable, this is to minimise the occurrence of system failures. System failures are caused by faults in a systems component. The execution of the faulty component raises an error which may lead to system failure. To make a system component fault tolerance means that the error has to be detected and corrected. These two steps are called *error detection* and *error recovery*.

The definition and a detailed discussion of the used terms may be found in [Lee, 1990].

3.2.1 Software Fault Tolerance Approaches

Turning to specific techniques for software fault tolerance, four main schemes can be identified [Banâtre and Lee, 1994] :

[1] Recovery Blocks

[2] N-Version Programming

[3] Recover and Retry

[4] Fix the Errors

Both, [1] (proposed in 1974) and [2] (subsequently proposed in 1977) are traditional fault tolerance approaches and have been shown to be successful. The expenses of these schemes are high (in terms of the need to produce multiple but different versions of the software system) and consequently a serious hurdle to the practical use. For details of these schemes refer to [Lee, 1990].

[3] is the base of a technique suggested by Gray [Gray, 1993] as an effective software fault tolerance technique. This approach is based on the premise that many of the errors caused by software faults are caused by transient situations. A backward error recovery is applied and the erring program is retried on a different machine, so that the same error is unlikely to occur again. The next Section proposes an architecture for a fault tolerant software component.

Finally, [4] is the least complete fault tolerance strategy. Here there are special programs, called audit programs, the purpose of which are to detect and correct errors in the system's database. Practical experiences suggest that this technique is very effective in practice [Banâtre and Lee, 1994].

3.2.2 An Architecture for Fault Tolerant Software Components

Under normal circumstances the component returns a normal response to a service request of the calling system. This activity may also include invocations of sub – components.

Abnormal responses (exception responses) can be generated due to a detected error in the component itself or an abnormal response of a sub – component.

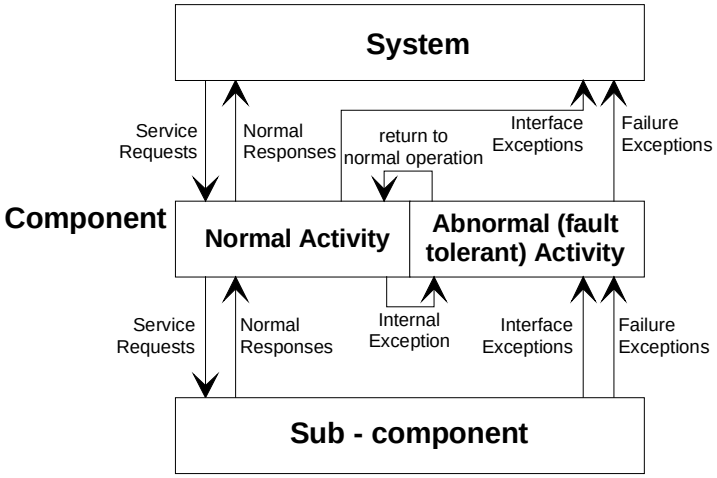

Figure 3-1. Architecture of ideal fault tolerant component,
based on [Banâtre and Lee, 1994].

There are two different exception responses. An *interface exception* indicates to the system that it had attempted to misuse the components interface. Note that in this situation there is nothing wrong with the component.

In contrast, a *failure exception* is generated by the component when it determines that it cannot provide its specified service. This is an indication that the component has failed. The generation of such an exception causes the component to transfer the control flow to its exception handling part (abnormal activity). Ideally, the control flow returns to the normal activity part of the component after the generation of the exception response.

3.3 Fault Tolerance in Distributed Systems

This thesis in general and this Section in particular does not consider hardware architecture related fault tolerance issues.

3.3.1 Distributed Object Middleware Solutions

In distributed systems, object middleware applications that cross the boundaries of different computing machines, operating systems and programming languages are becomming increasingly more popular. Such applications, that might be called “bridging technologies”, are becoming more essential for software development projects which work across heterogeneous environments [Baker et al., 1997].

These object middleware applications basically provide services which describe how client applications can invoke operations on server objects. CORBA and DCOM, which are introduced below, are such object middleware applications, both of them are commercially available. There is a number of other solutions, such as MOM, TPM or RMI, etc, that is not considered further.

3.3.1.1 Common Object Request Broker Architecture – CORBA

The *Object Management Group’s* (OMG) *Common Object Request Broker Architecture* (CORBA) is an open standard for distributed object computing. It defines a set of components that allow client applications to invoke operations on remote object implementations.

At this time, CORBA appears to be the most complete bridging technology solution. It provides an object *Interface Definition Language* (IDL) and a *Internet Inter ORB Protocol*

(IIOP) which reflect the attempt to standardise interoperability in between different computing environments.

CORBA also defines a wide range of services to extend its core capabilities. These services include :

- Naming
- Event Notification
- Life Cycle
- Persistence
- Relationships
- Externalisation
- Transactions
- Concurrency Control
- Licensing
- Query
- Properties
- Security
- Time
- Collections
- Trading
- Startup

Among these, the Trading Service is of special interest in this project. See also Section 3.3.2.

CORBA is well-suited for *best-effort*, client-server applications running over conventional local area networks, such as Ethernet or Token Ring [Maffeis et al., 1997]. Its synchronous method invocation model helps to avoid concurrency related problems. CORBA's services, such as the event, concurrency, and transaction service, help to address problems that arise when distributed network objects are executed in parallel.

However, building reliable and highly available applications with CORBA is hard because this is not directly addressed be the CORBA standard. In mission critical distributed systems reliability is an important quality. Even small amounts of down time may not be acceptable by customers or hurt business. A distributed system is called reliable if its behaviour is

predictable despite of partial failures, asynchrony, and run-time reconfiguration of the system [Maffeis et al., 1997].

[Maffeis et al., 1997] suggest three models that can help building reliable systems : Message Queues (MQ), Transaction Processing Monitors (TPM), and Virtual Synchrony (VS). Each model has its advantages and disadvantages. An overview of these models and their possible combination with CORBA can be found in [Maffeis et al., 1997].

3.3.1.2 Distributed Component Object Model – DCOM

Distributed Component Object Model (DCOM) is Microsoft's proprietary object middleware that evolved from *OLE* and *COM*. It appears to be the only other serious contender to CORBA [Baker et al., 1997].

DCOM enables *ActiveX* components to communicate with each other across a computer network. ActiveX components can be embedded into Web documents and be downloaded to the client's Web browser. The main drawbacks of DCOM include its support of an object oriented model that differs substantially from classical object oriented models, e.g. in its lack of polymorphism, and that it is available from a single vendor for a single operation system. See also Table 3.1.

3.3.1.3 CORBA in Comparison to DCOM

	CORBA	DCOM
Network	Yes	Yes
Language	Yes	Yes
Operation System	Yes	No
Administration	Yes	Yes
Legacy	Yes	Yes
Object model	Yes	Unknown
Vendor	Yes	No
Paradigm	Yes	Yes

Table 3-1. Boundaries versus Bridging Technologies [Baker et al., 1997].

Legend :

Yes = technology can bridge boundary

No = technology does not bridge boundary

Unknown = unable to assess or partial bridge of the boundary

A detailed description of the terms used in Table 3-1 may be found in [Baker et al., 1997].

3.3.2 OMG Object Trading Service

An OMG Object Trading Service is like yellow pages for objects in a distributed system. It lets a client discover objects on a server based on the service they provide. A service provider (server) advertises its services with a Trading Service. The service consumer (client) uses the Trading Service to discover services that matches its needs.

A new service provider will first register its service with the Trading Service. This means to give all the relevant information about its service to the Trading Service.

This includes [Orfali et al., 1997]:

- An *object reference*, used by the client to connect the service and to invoke on its operations.
- The *service name*, which includes the operations (or methods) to which the service will respond and their parameters and result types.
- The *properties of the service*, which describe the type (capability) of the service.

The Trading Service maintains all the service information in a repository. A client makes a request for a specific service type. Based on the properties of the services, the Trading Service performs a matching algorithm to return the result to the client. Based on the Trading Services information, it is now the clients responsibility to decide whether to invoke an object on a server. A Trading Service may extend its search to other Trading Services using a inter–trader protocol, such a schema is called *federated trading* [Orfali et al., 1997].

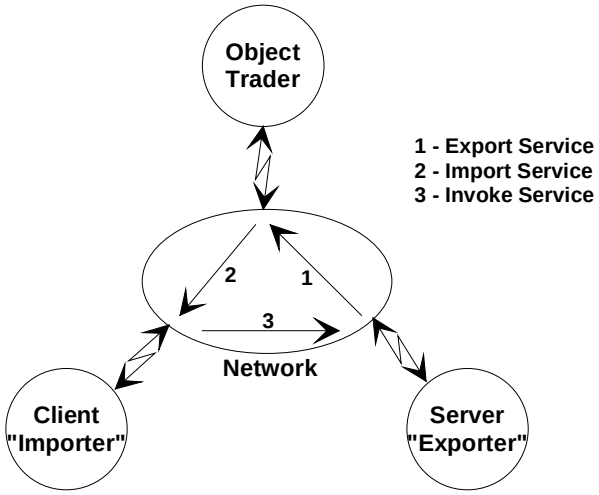

Figure 3-2. The OMG Object Trading Service.

Another reason for using an object Trading Service is system stability and scalability. Distributed systems consist of a large number of bindings between client and servers. Since these bindings are not reliable, (re)discovering and (re)connecting at run time helps in overall system scalability and stability in that the bindings can be reorganised.

3.3.3 Fault Tolerance in Distributed Systems

A service in a distributed system is called fault tolerant when it behaves according to its specification even in the presents of failures in parts (processor, media, communication link, other services) of the distributed system on which it depends.

3.3.3.1 Terminology

The definition and further explanation of the used terms may be found in [Mullender, 1993].

3.3.3.2 Types of Faults

Type of fault	**Real World Example**	**Well-known technique to handle the fault**
Processor faults	Machine or operating system crashes	Reboot using checkpointing / logging or recover critical state into another process
Communication faults	Lost, corrupted or duplicated messages	Time-out and retransmission
Media faults	Disk head crashes	Keep multiple copies of the data (“backup”)
Process faults	Resource shortage, software bug	Anticipate likely problems and write exception handlers or write multiple copies of the application
User “aborts”	Control “C”	Anticipate likely problems and write exception handlers

Table 3-2. Types of Faults.

3.3.3.3 Classification of Failures

Class of failure	Subclass	Description
Omission failure		A server omits to respond to a request
Response failure		A server responds incorrectly to a request
	Value failure	The server returns the wrong value
	State transmission failure	The server makes incorrect state transition
Timing failure		A server does not respond in the specified real-time interval
	Late or performance	The server responds too late
	Early	The server responds too early
Crash failure		A server repeatedly fails to respond to requests until being restarted
	Amnesia crash	The server restarts in initial state
	Partial amnesia crash	Some part of the state is as before the crash while the remainder is reset to initial state
	Pause crash	The server restarts in the state before the crash
	Halting crash	The server never restarts

Table 3-3. Classification of Failures.

A service may exhibit behaviours in the union of two or more failure classes. Such a server has a *weaker failure semantics* than one which exhibits behaviours in only one class. The stronger the failure semantics specified the more expensive it is to build the service. The weakest failure semantic, known as *arbitrary* or *byzantine*, is the union of all the failure classes introduced above.

3.3.3.4 Broadcast Specifications

The key for fault tolerance in distributed systems are reliable broadcasts [Mullender, 1993]. This paragraph describes different broadcast types. It does not address *multicasts*, a generalisation of broadcast.

Broadcast have the following three properties of reliable broadcast:

- *Validity :* If a correct process broadcasts a message m, them all correct processes eventually deliver m.
- *Agreement :* If a correct process delivers a message m, then all correct processes eventually deliver m.
- *Integrity :* For any message m, every correct process delivers m at most once, and only if some process broadcast m.

They only differ by the strength of their message delivery order requirements. There are three such requirements.

- FIFO Order *:* If a process broadcasts a message m before it broadcasts a message m', then no correct process delivers m' unless it has previously delivered m.
- Causal Order : If the broadcast of a message m causally precedes the broadcast of a message m', then no correct process delivers m' unless it has previously delivered m.
- Total Order : If correct processes p and q both deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

Therefore :

- Reliable Broadcast = Validity + Agreement + Integrity.
- FIFO Broadcast = Reliable Broadcast + FIFO Order.
- Causal Broadcast = Reliable Broadcast + Causal Order.
- Atomic Broadcast = Reliable Broadcast + Total Order.
- FIFO Atomic Broadcast = Reliable Broadcast + FIFO Order + Total Order.
- Causal Atomic Broadcast = Reliable Broadcast + Causal Order + Total Order.

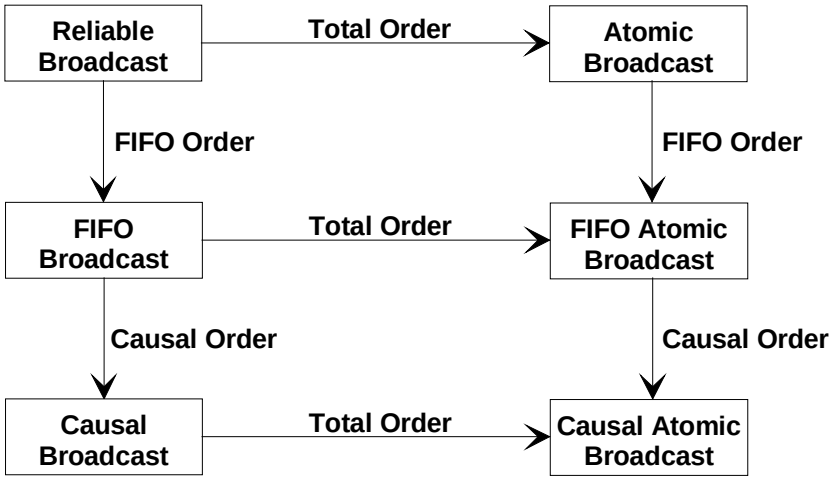

Figure 3-2. Relationship among Broadcast Primitives [Mullender, 1993].

3.3.4 Transactions

A transaction is a sequence of operations on shared data that is intended to transform the data from one consistent state to another [Bacon, 1993]. An individual operation may read the current value of a data item or write the value of a data item of a database or a distributed system. In a system that supports transactions, all operations on shared data must take place in the content of some transaction. Several transactions may be in progress simultaneously.

A transaction is typically started by using a primitive such as *BeginTransaction*. It executes by reading and writing volatile and persistent data like database relations, files or objects. Primitives such as *EndTransaction*, to cause the transaction to be *committed*, or *AbortTransaction*, to cause the transaction to be *aborted*, are used to end a transaction.

When processing distributed transactions, protocols such as *two phase commit* (2PC) are used to co-ordinate the *commit* or *abort* "votes" from the participating nodes.

3.3.4.1 The ACID Properties of Transactions

A transaction should have the following so called **ACID** properties [Bacon, 1993]:

- ***A**tomicity* : Either all or none of the transaction's operations are performed.
- ***C**onsistency* : A transaction transforms the system from one consistent state to another.
- ***I**solation* : An incomplete transaction cannot reveal its result to other transactions before it is committed.
- ***D**uration* : Once a transaction is committed the system must guarantee that the results of its operations will persist, even if there are subsequent system failures.

3.3.5 Isis

The Isis software developer kit enables the implementation of fault tolerant software in a distributed environment through a library of C functions. These functions include process group management and monitoring, reliable multicast communication, ordering of events and failure monitoring.

The supported reliable multicast communication are:

- *mbcast* : Unordered
- *fbcast* : FIFO ordered
- *cbcast* : Causally ordered
- *abcast* : Totally and causally ordered
- *gbcast* : Synch ordered

Isis runs on various operating systems including UNIX, Microsoft Windows, Microsoft Windows NT and VMS.

3.4 Evaluation Scenarios

To evaluate CORBA features, six basic OrbixWeb client-server applications were implemented in Java. The aim of these applications is to explore and understand CORBA and its features. Therefore, a simple echo service application interface was chosen to be implemented.

Unless stated otherwise, all applications use static method invocation and support the use of both the *Orbix protocol* and the *Internet Inter ORB Protocol* (IIOP). The advantages of IIOP, that connects objects via TCP/IP, over Orbix protocol are interoperability with other ORB's and the availability of servers which have no platform specific requirements.

The following Sections give a brief introduction to the implemented applications and outlines the aims of the features included.

The source code of these applications can be found in Appendix A.

3.4.1 Echo Service Application

In this basic echo service application a client connects, using OrbixWeb bind, to three servers. Each server holds two service objects. Markers are used to distinguish between the servers and the service objects.

A class called CCommandLine evaluates the given command line parameters to select the protocol and the hostname. This class is used in most of the applications described in this Section.

This application was implemented to understand a basic OrbixWeb application and to evaluate the relationship between a client and several service objects on several servers and the use of markers.

The source code of this application can be found in Appendix A1.

3.4.2 Echo Service Applet

This applet implementation evaluates the use of a *Graphical User Interface* (GUI) written using the *Java Development Kit* (JDK). This applet was downloaded into Appletviewer and Netscape browser.

The source code of this application can be found in Appendix A2.

3.4.3 Echo Service Smart Proxy Application

This application evaluates the client *smart proxy* feature. Smart proxies is an Iona specific feature that permits the manual implementation of proxy classes. Thereby allowing client interaction with remote services to be optimised. It may be beneficial to be able to implement proxy classes manually. Additional function, such as load balancing, caching or as in this case fault tolerance, can be included *hidden from the client application program*.

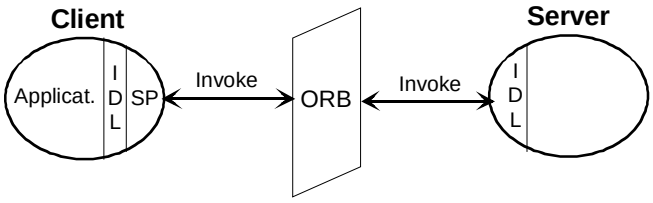

Figure 3-3. Client Smart Proxy.

Unfortunately, smart proxies is an Iona specific feature. Although other ORB's support similar features, e.g. VisiBroker calls them smart stubs, this feature cannot be used without loosing portability in terms of ORB independence.

When using smart proxies, the proxy itself has to be written and must then be activated by the client application. The activation of a smart proxy is the only change that has to be made in the client application. This is done by an instantiation in OrbixWeb and by a method call in VisiBroker. Smart proxies (or smart stubs) replace the default proxy generated by the IDL compiler. Thus, they have to be very similar regardless of the used ORB and should be easily migrated from one ORB to another.

The source code of this application can be found in Appendix A3.

3.4.4 Echo Service DII Application

In a normal (static) OrbixWeb client program, the IDL interfaces that the client accesses are determined at client compile time. The *Dynamic Invocation Interface* (DII) allows a client to call operations on IDL interfaces that were unknown when the client was compiled.

The DII allows invocations to specify the service object reference, the operation or attribute name and the parameters to be passed at runtime. A server receiving an incoming invocation request does not know whether the client which sent the request used the normal, static approach or the dynamic approach to compose the request.

This application uses the echo service applet's server and the Interface Repository (IFR) to obtain a reference to the service object. Object references may also be obtained from the OMG Naming Service or the OMG Trading Service.

Before invoking on a service object reference, all inout and out parameters have to be inserted as streamable objects. After the invocation, the return value has to be extracted.

The use of DII takes a lot of effort in the implementation of the client application. Furthermore, using DII's in an unnatural way of programming in a object oriented environment. Thus, the use of DII should be avoided whenever possible.

The client source code of this application can be found in Appendix A4.

3.4.5 Echo Service IDL Demo Application

This application evaluates OrbixWeb's mapping of *Interface Definition Language* (IDL) to Java and the rules used to convert IDL definitions into Java source code.

It includes the use of *holder classes*, *interface inheritance*, *interface method overloading* and *interface exception handling*.

IDL *in* parameters always map directly to the corresponding Java type. This is possible because these parameters are passed by value and Java supports passing by value of all types. However, IDL *inout* and *out* parameters have to be passed by reference and do not map directly into Java types. Therefore, holder classes, generated by the IDL compiler, have to be used.

Inheritance and method overloading are two well known object oriented programming features that may be included in IDL interfaces.

This application also includes interface exception handling. IDL specific exceptions cannot only be raised by IDL interface implementations but also by client smart proxies. They may therefore be used to inform clients of the status of a connection to service objects, e.g. a "reconnecting to backup service object exception" may be raised.

No extra application was implemented to evaluate these IDL to Java mappings. A good example, called idl_demo, is included in OrbixWeb3.0. This example uses a description of a bank account interface to outline the use of IDL to Java mappings.

3.4.6 Echo Service Trading Application

This application evaluates the use of the *Orbix Trading Service* by OrbixWeb clients and servers via IIOP.

The service offer type for the echo service was added to the Trading Service using the Trading Service GUI tool.

When starting up, the echo server registers its service offer with the Trading Service. After handling incoming client service requests, the echo server withdraws its service offer before exiting.

The echo service client queries the Trading Service for an echo service offer, obtains the servers service object reference and invokes on the echo service object.

Both, the echo service client and the echo service server have the Trading Service's Interoperable Object Reference (IOR) as a string. This string is then converted into an object reference, that can be narrowed into any Trading Service interface reference.

The source code of this application can be found in Appendix A5.

3.5 Summary

To make a computer system dependable, its software components must be made *fault tolerant*. This means that an error in a component has to be detected and corrected. Recovery and Retry is the base of a technique suggested by Gray [Gray, 1993] as an effective software fault tolerance technique.

In distributed systems, object middleware applications that cross the boundaries of different computing machines, operating systems and programming languages are becoming more essential for software development projects which work across heterogeneous environments. A open standard of such a middleware application is OMG's CORBA. CORBA is well-suited for *best-effort*, client-server applications running over conventional local area networks, such as Ethernet or Token Ring. Building reliable and highly available applications with CORBA is hard because this is not directly addressed by the CORBA standard.

CORBA also defines a wide range of services to extend its core capabilities. One of these services is the *Trading Service*, used to let a client discover server objects at runtime. Distributed systems consist of a large number of bindings between client and servers. Since these bindings are not reliable, (re)discovering and (re)connecting at run time helps in overall system scalability and stability in that the bindings can be reorganised.

Another object middleware feature is *smart proxy*, which is not part of the CORBA standard. Smart proxy permits the manual implementation of proxy classes, allowing client interaction with remote services to be optimised. Additional function, such as load balancing, caching or, as in this case, fault tolerance, can be included *hidden from the client application program*. Although smart proxy is an ORB vendor specific feature, which means loosing portability, smart proxy applications should be easily migrated from one ORB to another.

4. DESIGN

4.1 Introduction

To solve the fault tolerance problems identified in Chapter 2, an *OMG Trading Service* is introduced to the Jetpac infrastructure. The included component is Iona's OrbixTrader. OrbixTrader must have appropriate *service types*, including a sequence of property structures that describe services, to export and import service offers. Property structures include client group identifier lists. Clients with similar needs and location are grouped together and are given a *unique group identifier*. Servers offer their services to a particular client group by including client group identifiers in a service offer. Clients then query OrbixTrader for service offers that include their group identifier.

A basic architecture is proposed to be included into the Jetpac infrastructure. This *simple, low cost solution* solves the fault tolerance problems without involving the Jetpac notification service (LNC). Clients query OrbixTrader for master and backup service offers and cache the retrieved service references. If the service in use fails, the service user 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 illuminate this, two improvements are proposed, both of which make use of the *Jetpac notification service (LNC).* Within the first improvement, clients use notifications to detect service failures and therefore need not to ping services anymore. This results in less network traffic and reduced service user idle time. Within the second improvement, notifications are used to automate service offer maintenance and to further minimise network traffic.

A final improvement, that requires minor changes only, is suggested. *Dynamic properties* are used to provide load balancing.

4.2 Service Offers

Servers cannot export *service offers* to the OrbixTrader unless the Trader has appropriate *service types*. OrbixTrader can contain a number of such service types that describe services. These service types can be managed using OrbixTrader's TraderTool.

Service types consist of :

- a type name
- an IDL interface ID
- a sequence of property structures

Service offers describe a specific service provided by some server, based on the information defined in a service type.

Service offers consist of :

- a reference that is the actual service object reference
- a sequence of properties, where each property is a name-value pair

Each service property of a property structure is qualified by *mode attributes*.

Mode attributes are :

1. *Normal* :
   A service offer need not supply a value for this property, but if it does, the value must match the corresponding property type.
2. *Read - only* :
   The value of the property can initially be set when its offer is exported to the Trading Service, but cannot be modified afterwards.
3. *Mandatory* :
   A value for this property must always be provided when a service offer is exported to the Trading Service.
4. *Mandatory & Read - only* :
   A value for this property must be supplied when a service offer is exported to the Trading Service and cannot be changed afterwards.

These mode attributes are considered to increase in property strength where *Normal* is the weakest an *Mandatory & Read - only* is the strongest.

4.2.1 Querying for Service Offers

When OrbixTrader processes a client's query for a service offer, it gathers a sequence of offers together by *narrowing down* the set of potential offers. The query input is used to determine the following :

1. The *service name* is used to determine whether an offer is of an appropriate service type. Therefore, potential offers are narrowed down to the appropriate service type.
2. The *property constraints* are used to determine whether the offer matches.
3. *Preferences* are used to determine the order in which to place the offer in the created offer sequence.
4. *Desired properties* are used to determine which of the property values are returned.

Furthermore, *policies* are used by OrbixTrader to control its behaviour, e.g. the number of offers to be searched, the use of dynamic properties, etc.

The standard constraint language, that is OMG's "OMG 1.0", is used by clients to query the OrbixTrader. This constraint language includes basic comparative functions such as equality, inequality, sub-string matching, etc. These comparative functions are used to match basic property types such as boolean, long, string, etc. Other constraint languages may be supported by other OMG Trading Service implementations.

In this project, all properties are requested by the client to verify the correctness of the chosen algorithms. To increase query performance, the list of requested properties should be kept as small as possible, which means that properties of special interest, such as changing values or dynamic properties, should be requested only.

The random preference sort order was chosen because there is most likely only one (or very few) service offer in the created offer sequence. There will be only one server offering a particular master service for a particular client group.

This project is not looking at enhanced Trading such a federated Trading and therefore the default policies cover all client query needs.

4.2.2 Service Offer Property Sequence

The following table shows the core property sequence of the fault tolerant service offers used in this project and includes property value examples. Additional properties can be appended at will, as long as their mode attribute is *Normal* (neither *Mandatory* nor *Read - only*) or *Read - only* in order to keep service offers compatible.

Property name	Data type	Mode attributes		Value example
		Mandatory	Read - only	
ServiceTypeName	String	✔	✔	ft_echoService_if
ServerName	String	✔	✔	echoServer1
ServiceName	String	✔	✔	Echo Service One
MasterList	String	✔		“0001-0002”
PrimaryBackupList	String	✔		“0003-0004”
SecondaryBackupList	String	✔		“0005-0006-0007”
OfferIsValid	Boolean	✔		true
ServerUtilization	Long			18 %
NumOfUsersOnServer	Long			6

Table 4-1. Property sequence.

4.2.2.1 Mode Attributes

The property sequence includes properties that have different mode attributes. ServiceTypeName, ServerName and ServiceName’s mode attributes are *Mandatory and Read - only*. These properties uniquely identify a service. A property value must always be provided and cannot be changed during service lifetime. If one of these properties must be changed, the original service offer must be withdrawn and then be replaced by the new service offer.

The mode attribute of MasterList, PrimaryBackupList, SecondaryBackupList and OfferIsValid is *Mandatory*. These properties describe the target client groups and the status of the offered service. These values may change during service lifetime, e.g. a service may become backup for another client group.

ServerUtilization and NumOfUsersOnServer's mode attribute is *Normal*. The feature that supplies the values for these properties may not be supported by some services. Such services ignore these properties simply by not providing a value for them.

4.2.2.2 Names and Values

ServiceTypeName, ServerName and ServiceName contain string values which uniquely identify a service. A particular service type (ServiceTypeName) can be provided by several servers (ServerName), which can supply several similar services (ServiceName). These values are used when clients register with notification channels they are interested in and could also be displayed on the service user side.

MasterList, PrimaryBackupList and SecondaryBackupList contain string values that include a list of client group identifiers. Client group identifiers must be unique and need to be separated within the string by a delimiter. A client queries the Trading Service for an offer that includes its group identifier in the MasterList or in one of the BackupLists respectively.

The OfferIsValid boolean value marks a service offer as valid or invalid. A service offer is marked as invalid when it is temporary out of service, e.g. for maintenance reasons. Thus, clients expect such services to be back on-line eventually. This is not expected if no service offer was found in a particular list, e.g. there might be no secondary backup service available for a particular client group.

ServerUtilization and NumOfUsersOnServer's long values could be used to select the best available offer in terms of load balancing. Clients could select the service offer with the lowest server utilisation and / or number of users. This task could be delegated to the Trading Service by using the "min" preference. These properties should be implemented as dynamic properties. As mentioned earlier, load balancing related issues are addressed but not implemented in this project.

Using property names and values, clients query the Trader for a particular service type (ServiceTypeName) that is master (MasterList) or backup (PrimaryBackupList or SecondaryBackupList) for its group id. The imported service offer includes server and service name (ServerName, ServiceName), the current service status (OfferIsValid) and load balancing information (ServerUtilization, NumOfUsersOnServer).

4.3 Fault Tolerance Architecture

The following architecture and its improvements solves the fault tolerance problems identified in Chapter 2. It requires that an OMG Trading Service is available within the Jetpac infrastructure. The *basic architecture* is a low cost solution that does not require notifications. The improvements may be build on top of it to increase re-connection performance, reduce maintenance and to include additional features such as load balancing. These improvements require notifications generated by the Jetpac notification service (LNC). Even in the presence of a well designed, reliable and highly efficient notification service, there is always a possibility that *clients detect service failures **before** receiving the corresponding notification.* Clients are enabled to handle this by first implementing the basic architecture and then building the improvements on top of it.

By the time these architectures were suggested, the Jetpac notification service was not fully operable yet.

This table shows the tasks that have to be performed to provide fault tolerance within the Jetpac infrastructure. The following Sections are referring to this table when discussing the tasks.

Where	What
Clients	[A] have to detect a service failure [B] need to obtain a backup service reference [C] have to re-connect to the backup service [D] have to re-connect to the original service as soon as it is back on-line
Services	[E] have to be maintained in order to be fail silent
Servers	[F] have to be maintained in order to be fail silent [G] have to provide values for dynamic properties
Trading Service	[H] service offers have to be exported to it [I] temporary invalid service offers have to be marked [J] service offers that are no longer valid have to be withdrawn [K] dynamic properties have to be supported for load balancing

Table 4-2. Tasks to Provide Fault Tolerance.

4.3.1 Basic Architecture

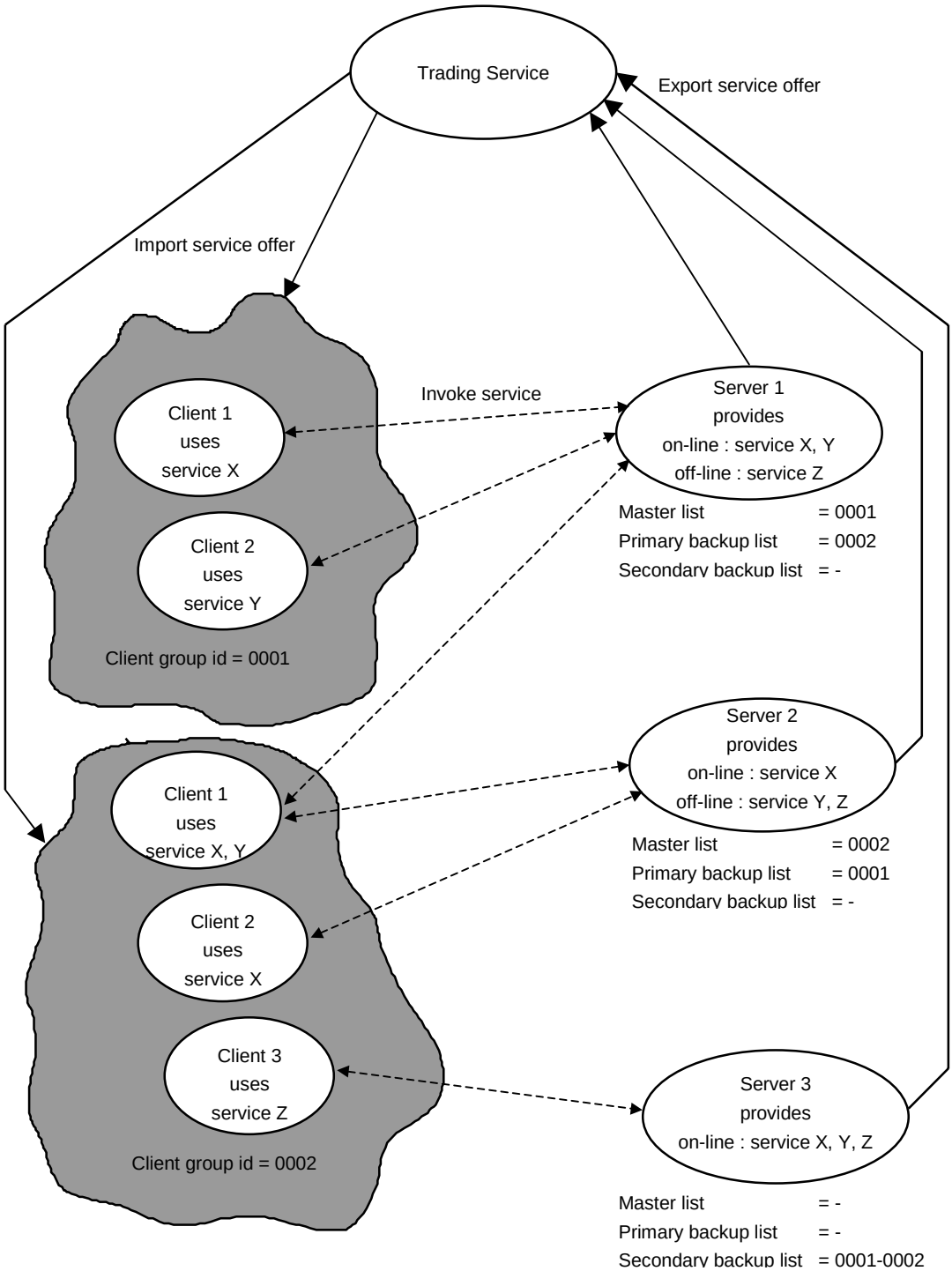

Figure 4-1. Basic Architecture.

Legend :

- Client 1 (group id = 0001) invokes master service X on Server 1.
- Client 2 (group id = 0001) invokes master service Y on Server 1.
- Client 1 (group id = 0002) invokes master service X on Server 2.
- Client 1 (group id = 0002) invokes primary backup service Y on Server 1.
- Client 2 (group id = 0002) invokes master service X on Server 2.
- Client 3 (group id = 0002) invokes secondary backup service Z on Server 3.

This simple, low cost solution does not involve the Jetpac notification service (LNC).

Clients detect service failures [A] by the time they invoke on a service object, even if the service failed earlier. An invocation on a failed service will eventually return an exception. The service user is idle during failure detection time which depends on timeouts and network topology. Backup service references [B] were pre-fetched and cached and should therefore be available. Thus, re-connection to the backup service [C] will be efficient. Because of the lack of a cache updating mechanism, backup service references might have become invalid. After detecting an invalid service, clients start pinging the original service and re-connect [D] to it as soon as it is back on-line.

Services and Servers are maintained [E], [F] by the Jetpac Service Manager (SM). Load balancing [G], [K] is not supported by this architecture. The maintenance of service offers, such as exporting [H], marking invalid [I] and withdrawing [J], is not automated and has to be performed by IT personnel.

Although this solution does the job, several improvements may be made. Service user idle time may be reduced and the client's cache need to be updated. Pinging the original service before re-connecting to it is not efficient and causes unnecessary network traffic. Furthermore, service offer maintenance should be automated and load balancing may be supported.

4.3.2 Architecture Improvement 1

This figure shows the improvements on the basic architecture.

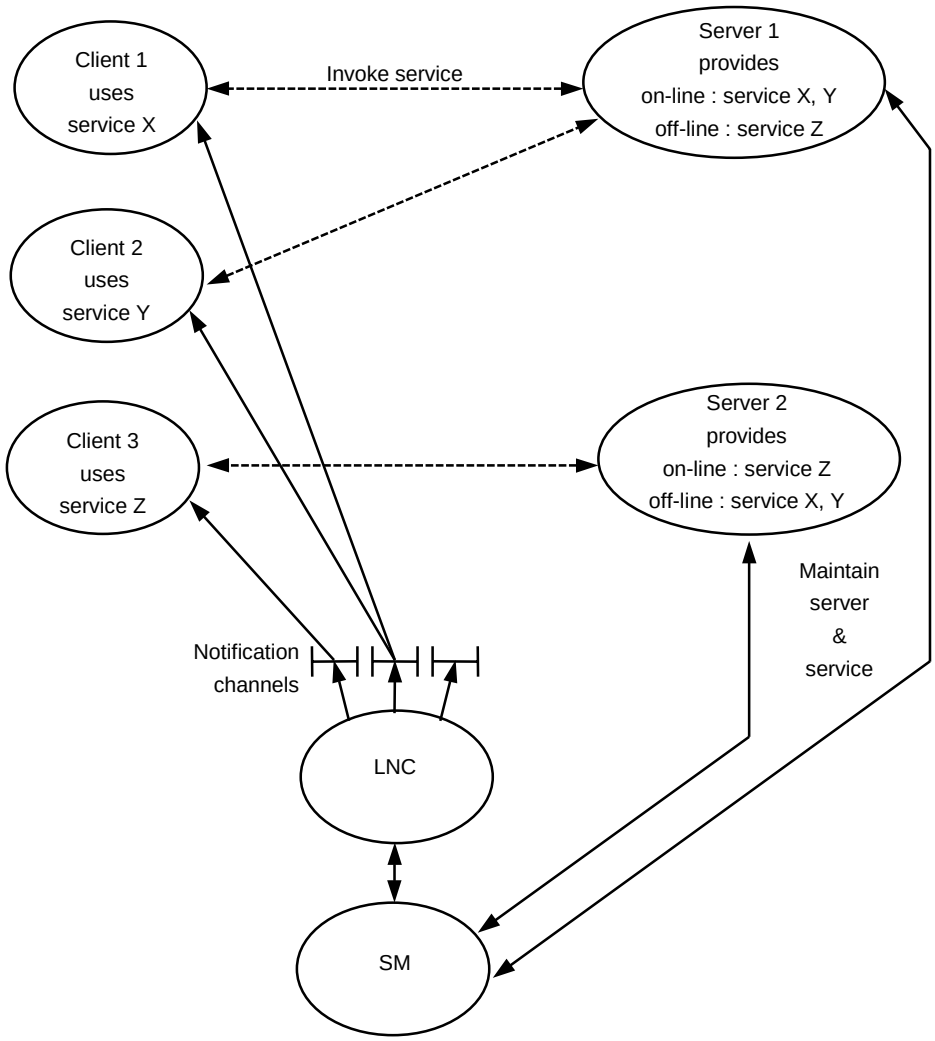

Figure 4-2. Architecture Improvement 1.

Legend :

- SM : Jetpac Service Manager.
- LNC : Jetpac notification service.
- Client 1 and Client 2 receive a notification : “Server 1, Service Z out of service”.
- Client 3 receives notifications : “Server 2, Service X out of service”, “Server 2, Service Y out of service”.

This improvement does include the *Jetpac notification service* (LNC) into its fault tolerance mechanism.

Clients receive notifications whenever a service goes out of service and when it is back on-line. Invalid service references are detected [A] and clients are re-connected [C] to cached [B] backup service references in most cases without service user idle time. There is no need for pinging the original service. Clients re-connect [D] to the original service reference immediately after receiving the corresponding notification.

Services and Servers are maintained [E], [F] by the Jetpac Service Manager (SM). Load balancing [G], [K] is not supported by this architecture. The maintenance of service offers, such as exporting [H], marking invalid [I] and withdrawing [J], is not automated and has to be performed by IT personnel.

The improvements on the basic architecture include reduced, in most cases no, service user idle time due to re-connection in background and updated client cache. There is no need to ping the original service anymore, which results in less network traffic.

Service offer maintenance is still not automated, which means that clients could receive invalid service offers from the Trading Service, due to delayed or forgotten service offer maintenance.

The load balancing feature is still not supported yet.

4.3.3 Architecture Improvement 2

This figure shows the improvements on the basic architecture.

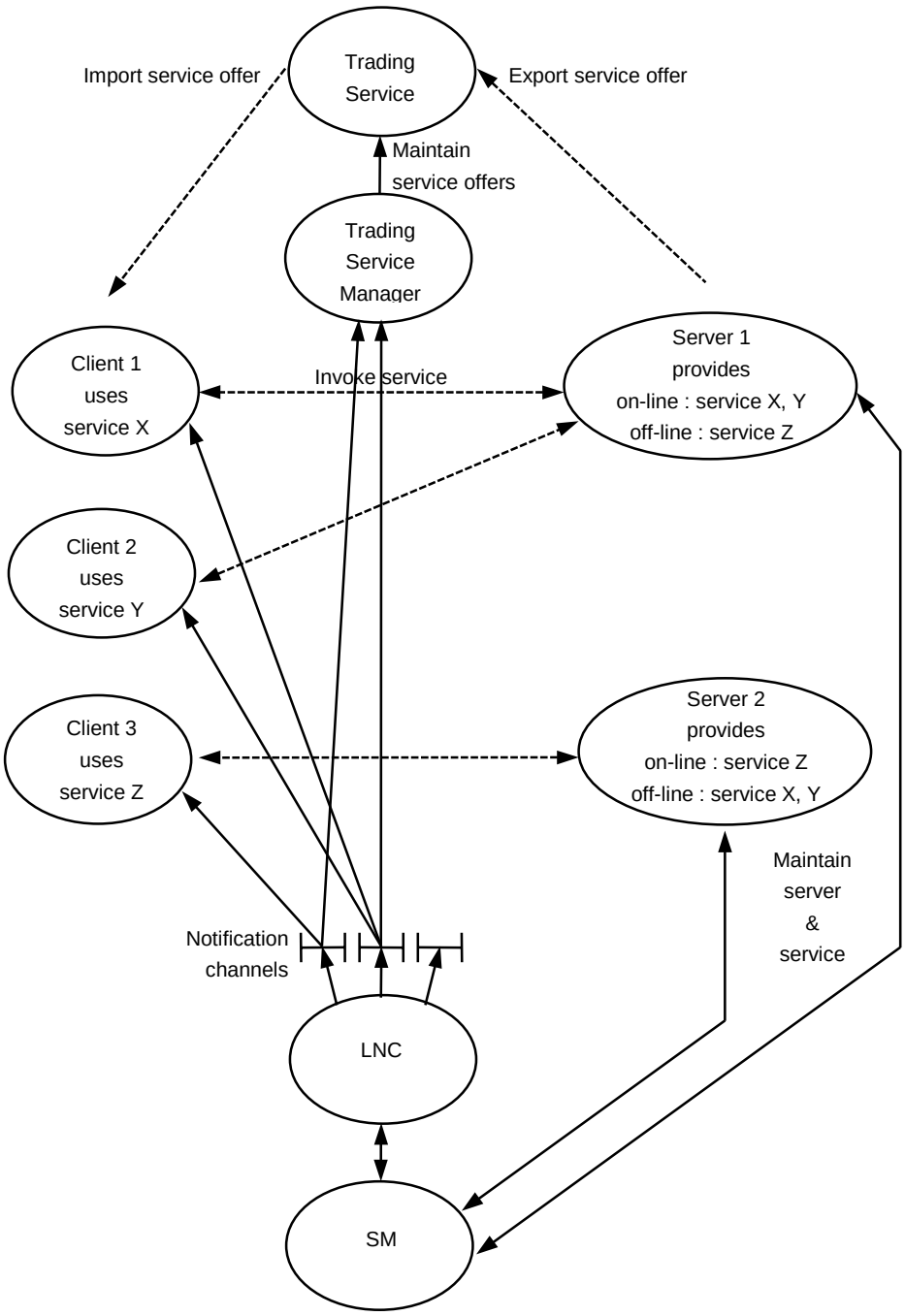

Figure 4-3. Architecture Improvement 2.

Legend :

- SM : Jetpac Service Manager.
- LNC : Jetpac notification service.
- TSM : Trading Service Manager.
- Client 1 and Client 2 receive a notification : “Server 1, Service Z out of service”.
- Client 3 receives notifications : “Server 2, Service X out of service”, “Server 2, Service Y out of service”.
- TSM receives notifications : “Server 1, Service Z out of service”, “Server 2, Service X out of service”, “Server 2, Service Y out of service”.

This improvement is build on top of the basic architecture and the first improvement which includes the Jetpac notification service (LNC). Another component, called Trading Service Manager (TSM), is introduced. This component also makes use of the Jetpac notification service (LNC). Its task is to update Trading Service’s service offers.

Clients detect invalid service references [A], re-connect [C] to cached [B] backup service references and re-connect to the original service reference [D] as they did in the improvement 1 architecture. Because the Trading Service Manager (TSM) marks temporarily invalid service offers [I], clients will recognise invalid service offers when receiving them from the Trading Service, this reduces the possibility of service user idle time further.

Servers cold export [H] their service offers at start-up time and withdraw [J] them before shutting down. Therefore, IT personnel has to maintain service offers of crashed services, that are not going on-line again, only.

Services and Servers are maintained [E], [F] by the Jetpac Service Manager (SM). Load balancing [G], [K] is not supported by this architecture.

The improvements on basic architecture and improvement 1 include automated service offer maintenance and minimum network traffic, due to an updated Trading Service.

The drawback of this improvement is the introduction of another component to the Jetpac infrastructure, that is, in absence of replication, another single point of failure. Attention has to be paid to the delivery order of the notifications. When a service offer updated notification is generated, the Trading Service must be updated before clients query for the affected service offer. This can be prevented by including a logical timestamp (sequence number) in the notification. This sequence number is added to the service offer by the Trading Service Manager (TSM) when updating it. Clients are then able to verify whether they received an up to date service offer from the Trading Service. Another way to guarantee notification delivery order is to send them to the Trading Service Manager (TSM) only, which updates the Trading Service and then forwards them, via Jetpac notification service (LNC), to the clients.

The load balancing feature is still not supported yet.

4.3.4 Load Balancing

This improvement provides the load balancing feature. It does not require changes in the architecture and may be build on top of each of the introduced architectures.

Clients use the “min” preference to receive the best available offer considering the load balancing issue.

The Trading Service must support dynamic properties to be able to retrieve the current dynamic property value.

Servers must export an object reference that can be invoked by the Trading Service to retrieve the current dynamic property value.

Including the load balancing feature in an existing architecture does not require major changes. Clients do not have to be changed at all and the Trading Service requires reconfiguration only. Minor changes have to be made on the server side, servers must provide the current values for the dynamic properties.

4.4 Summary

The three proposed architectures all solve the fault tolerance problems identified in Chapter 2. They provide fault tolerance, in terms of service reliability and availability, and Quality of Service (QoS) in terms if scalability of the infrastructure. Performance, in terms of service user idle time and network traffic overhead, increases with each improvement on the basic architecture.

By using the dynamic property feature of the Trading Service, load balancing can be included easily.

The way to implement these architectures is to start with the basic architecture and then build the improvements on top of it. The mechanism of the basic architecture will be needed even in the presents of the improvements. Regardless, how well designed, reliable and highly efficient the used notification service might be, there is always a possibility that clients detect service failures before receiving the corresponding notification.

Client side fault tolerance algorithms are implemented as smart proxy classes, in order to hide them from the client application program. Unfortunately, the smart proxy feature is ORB vendor specific. A way to implement fault tolerance algorithms ORB independent is to place them between the IDL interface and the client application program. There they are not hidden from the client application program, which, for what ever reason, could avoid to used them.

5. IMPLEMENTATION

5.1 Introduction

A prototype of the basic fault tolerance architecture, introduced in Chapter 4, was implemented in the Java programming language. The implementation takes full advantage of the strengths of the Java programming language, such as exception handling, strong type checking and the ability to run on multiple platforms.

The prototype implementation consists of four packages. Three of these, ft_echoService, ft_Components and ft_Exceptions, *implement the fault tolerance framework*. The fourth, ft_echoServiceApplication, implements client and server applications which use the framework. Packages ft_Components and ft_Exceptions are *re-usable*, whereas ft_echoService is service (IDL interface) specific and could be generated by a *fault tolerance compiler* that may be implemented.

Package ft_echoServiceApplication includes the javaclientStartup program that is used to automatically launch the server implementations and therefore to force them to export their service offers to the Trading Service. This program does not use the fault tolerance framework and was therefore used to evaluate invocation duration on bound service objects. This package also includes the programs javaclient1 and javaclient2, client implementations that make use of the fault tolerance framework. Finally, javaserver1, javaserver2 and javaserver3 are server implementations, each of which provides two echoServices and instantiates dynamic property implementations.

Package ft_echoService, used by clients only, holds a smart proxy and a smart proxy factory class as well as the algorithms used to query the Trader, to update the service object reference list and to (re)connect to the best available service object.

Package ft_Components, used by both clients and servers, includes a Trader interface, a configuration file parser and several classes that provide basic functionality.

Package ft_Exceptions, also used by both clients and servers, implements the exceptions thrown by this framework.

The *Unified Modelling Language* (UML) notation is introduced for graphically describing these packages. UML is a standard language for visualising, specifying, constructing and documenting elements of a software system. Use cases, interaction diagrams and class diagrams were developed for this project.

5.2 Basic Architecture Implementation

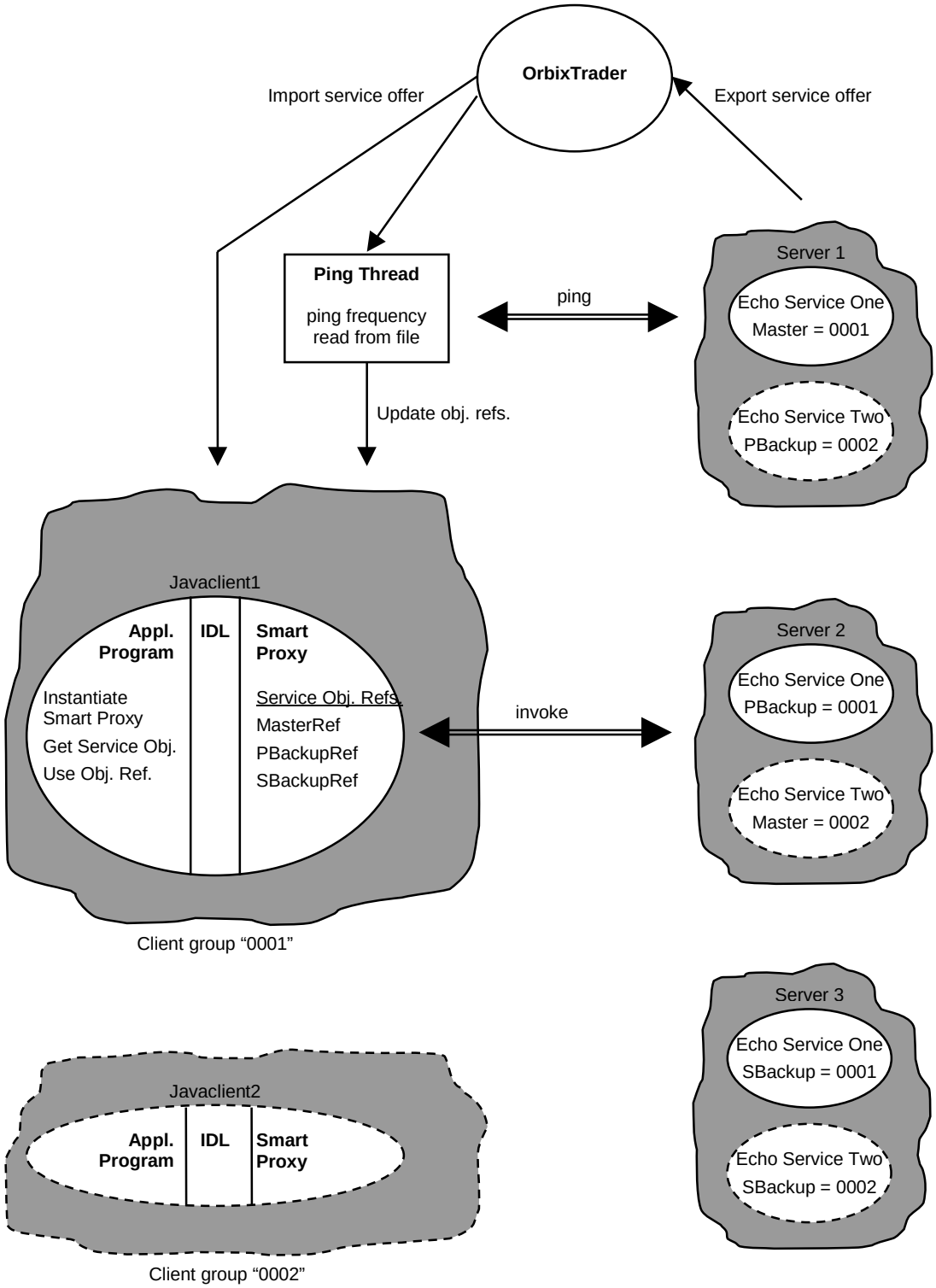

Figure 6-1. Basic Architecture Implementation.

To use this fault tolerance framework in existing client applications, the IDL interface needs to be *extended*. These extensions include the definition of an IDL exception, a Ping method and wrapper methods for each of the existing methods. Both, the original and the wrapper method provide fault tolerance.

In order to delegate method calls to the framework, client applications simply instantiate the smart proxy factory provided and retrieve a service object reference form the smart proxy factory instance. Client applications may then start invoking operations on the retrieved service object reference.

Client applications may invoke the original methods or the wrapper methods. If there is no service object available, wrapper methods throw an exception, whereas original methods return default values. Fewer changes need to be made in existing client applications when using the original methods.

When instantiated, the client smart proxy queries the Trading Service for master, primary backup and secondary backup service object references. The imported service object references are cached and the application program invocations are delegated to the best available one.

Whenever an invalid service object reference is detected, invocations are re-directed to the next best service object reference and a PingThread is started. The task of the PingThread is to periodically query the Trading Service for a new service object reference to replace the one of the failed service. In order to detect whether a service is on-line, the PingThread invokes (pings) on the imported service object reference. Object references of newly available services are cached and a scheduling algorithm re-directs application program invocations if necessary.

Configuration parameters, such as client group identifier, Trading Service IOR file name or PingThread frequency, are read from a configuration text file. This allows on-line configuration changes, e.g. of the PingThread frequency, without restarting clients.

5.3 Use Case Diagram

5.3.1 Client Application

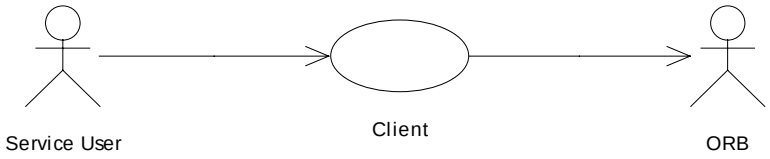

Figure 6-2. Client Application.

Client applications are started up by service users, who then select services to be used. Client applications import service offers from the Trading Service and invoke on service objects through the ORB.

5.3.2 Server Implementation

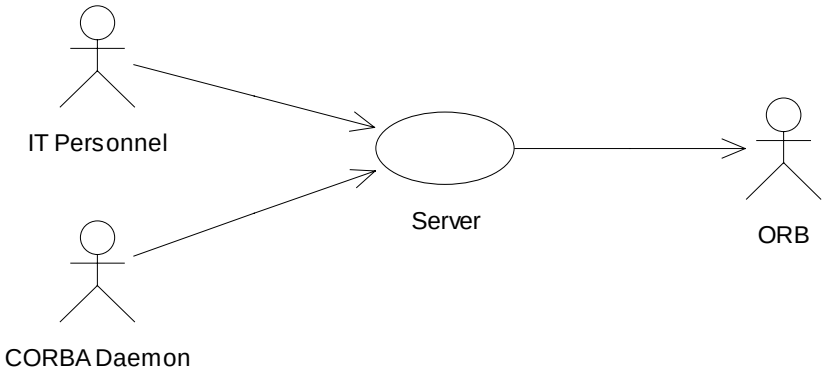

Figure 6-3. Server Implementation.

Servers are started up either manually by IT personnel or automatically by a CORBA Daemon process. Servers then instantiate their services and export service offers to the Trading Service. Clients use the exported service offers, which include service object references, to invoke on service objects provided by servers through the ORB.

5.4 Interaction Diagram

5.4.1 Client Sequence Diagram

5.4.1.1 Package ft_echoServiceApplication

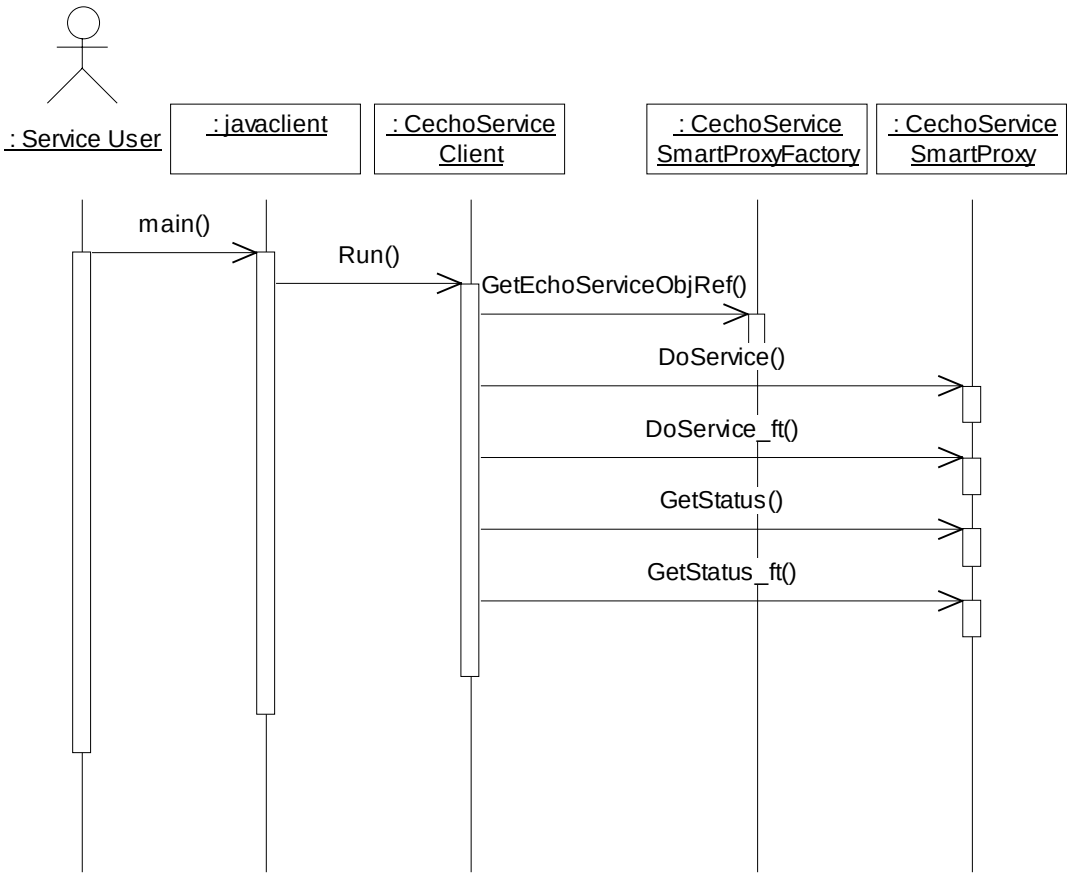

Figure 6-4. Fault Tolerant Client.

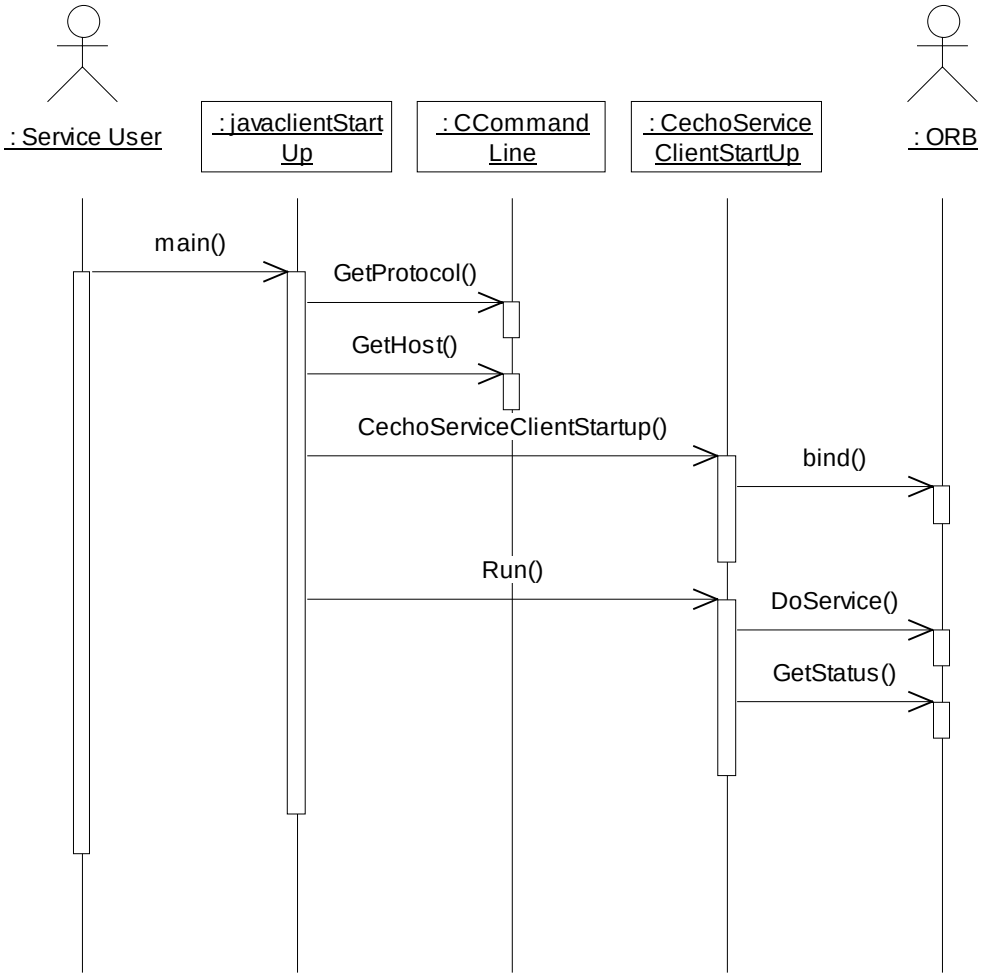

Figure 6-5. Startup Client.

5.4.1.2 Package ft_echoService

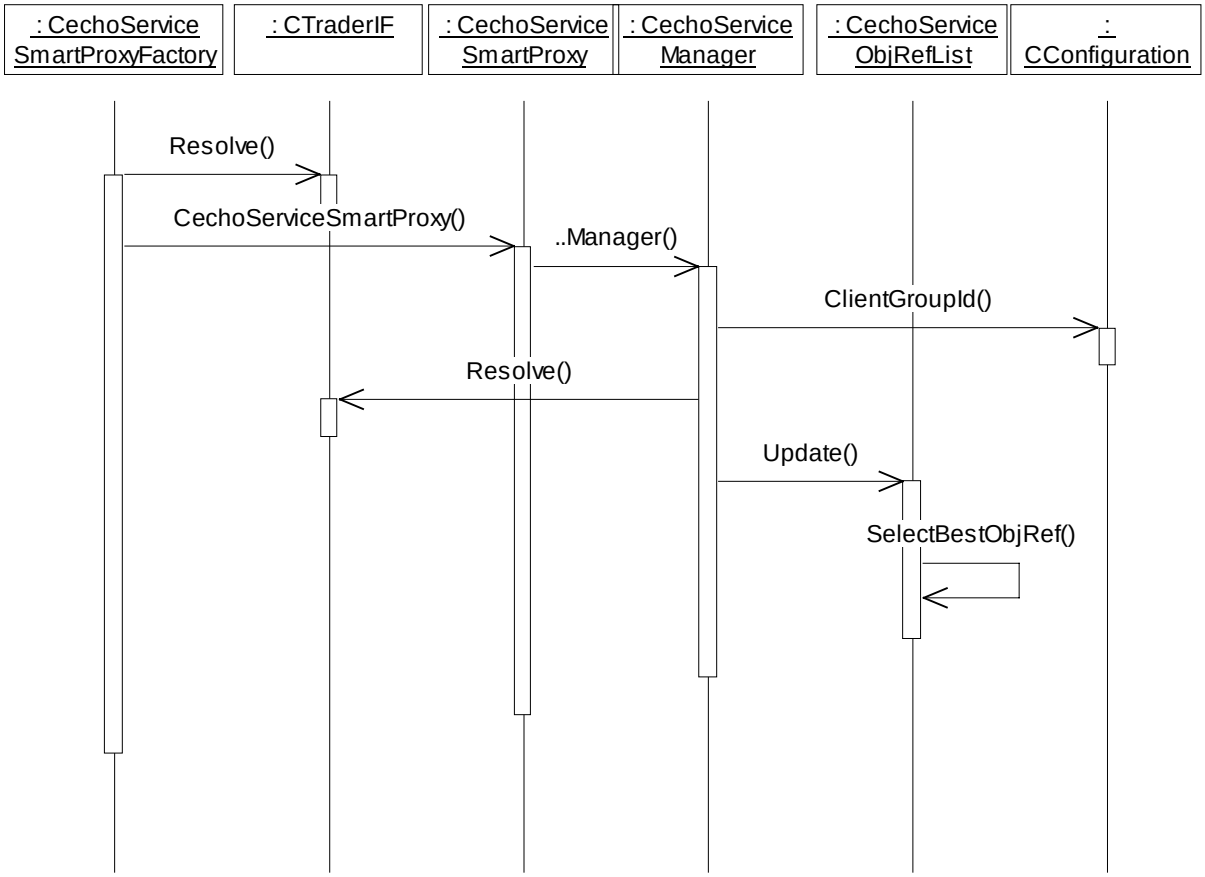

Figure 6-6. Fault Tolerant echoService.

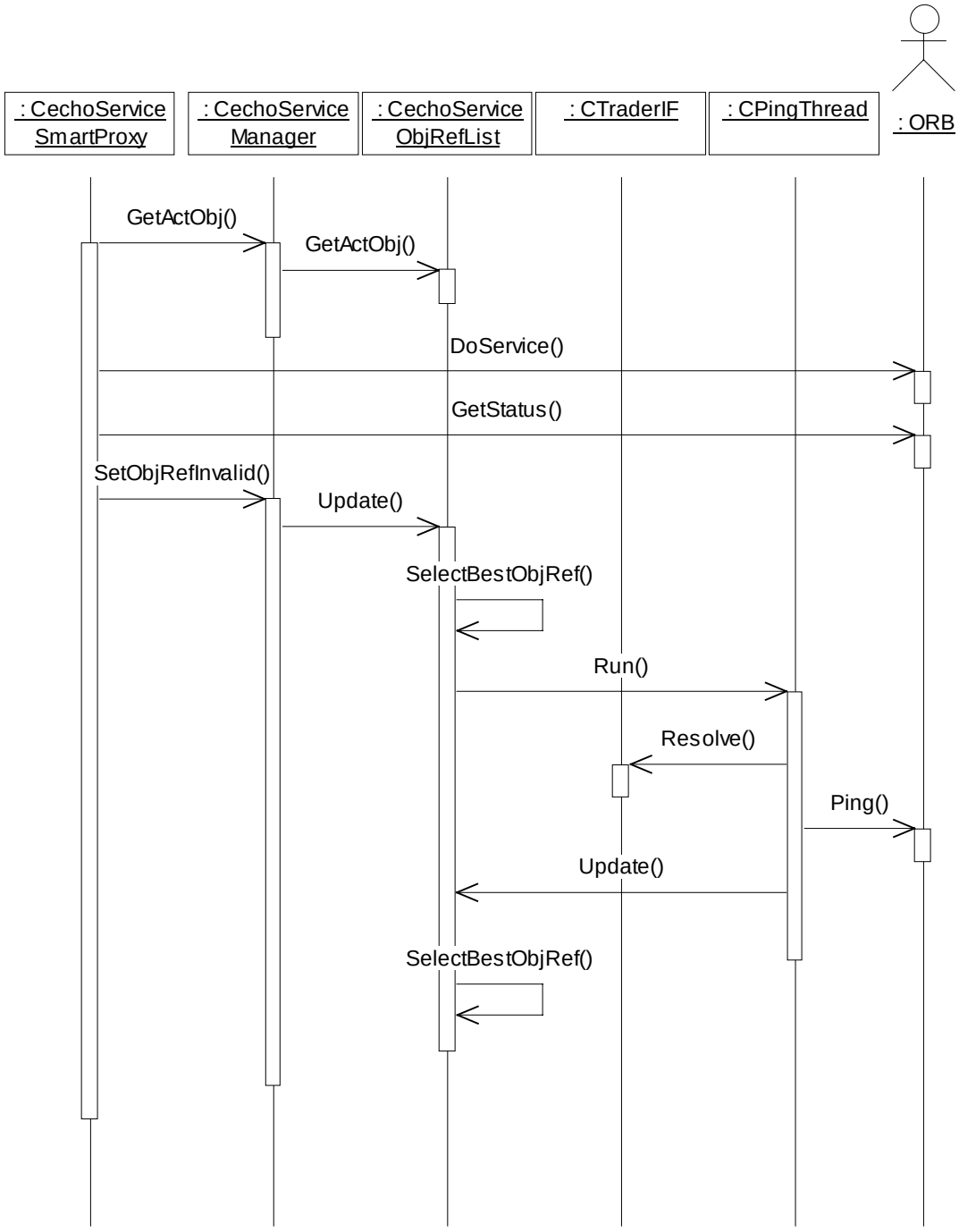

Figure 6-7. Fault Tolerant echoService (cont.).

5.4.1.3 Package ft_Components

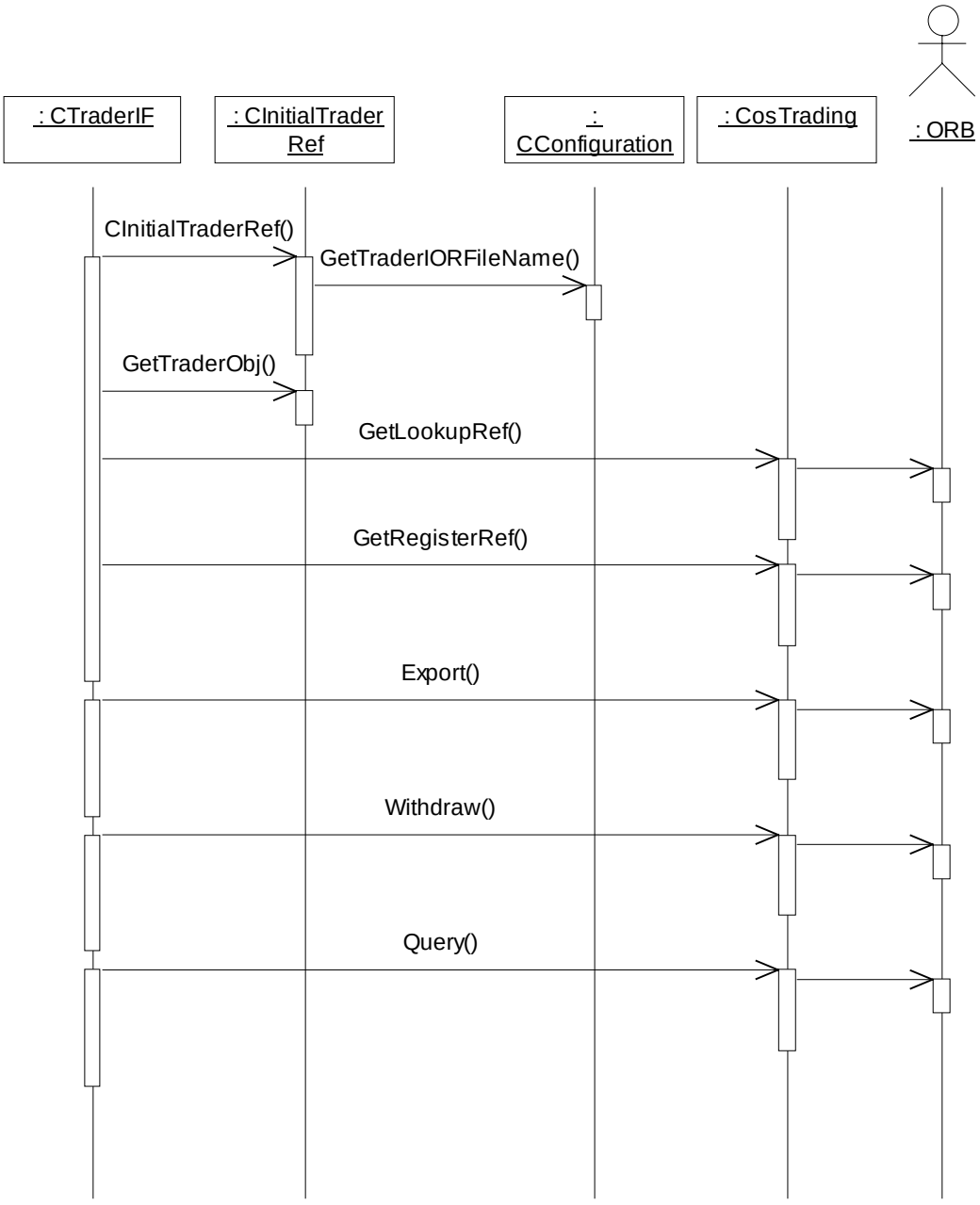

Figure 6-8. Fault Tolerant Trader Interface.

5.4.2 Server Sequence Diagram

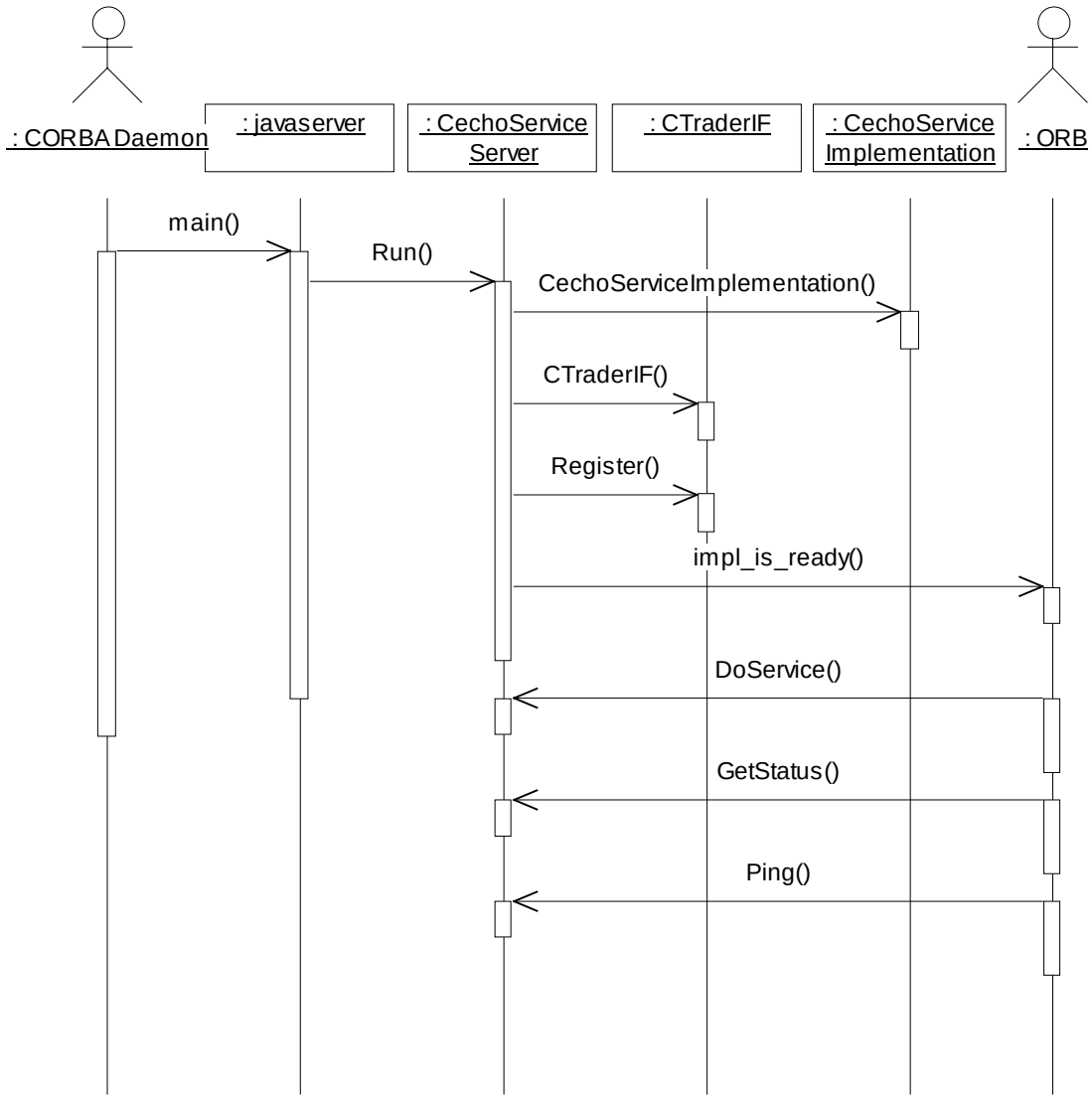

Figure 6-9. Server.

5.5 Class Diagram

5.5.1 Package ft_echoServiceApplication

5.5.1.1 Client Startup

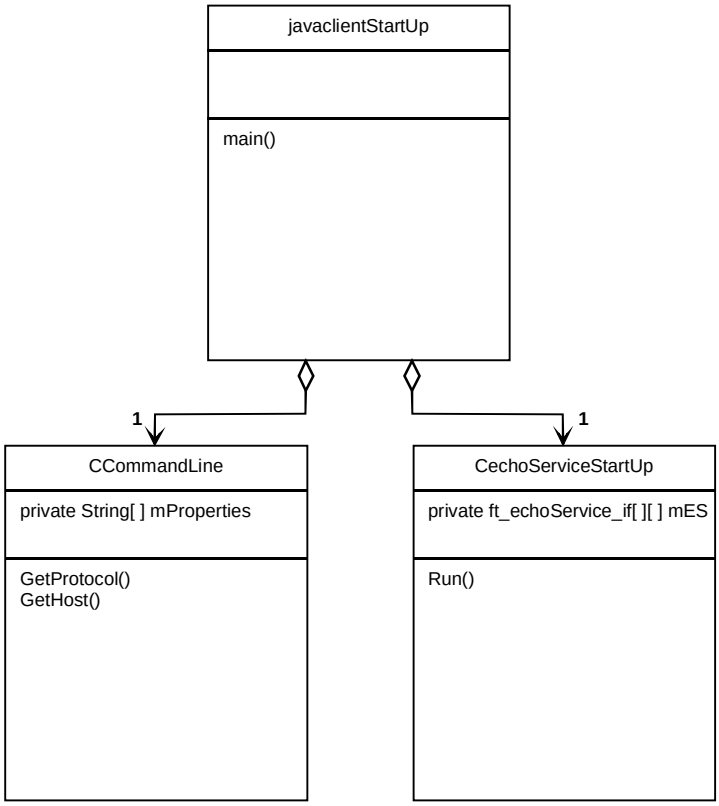

Figure 6-10. Client Startup.

5.5.1.2 Client

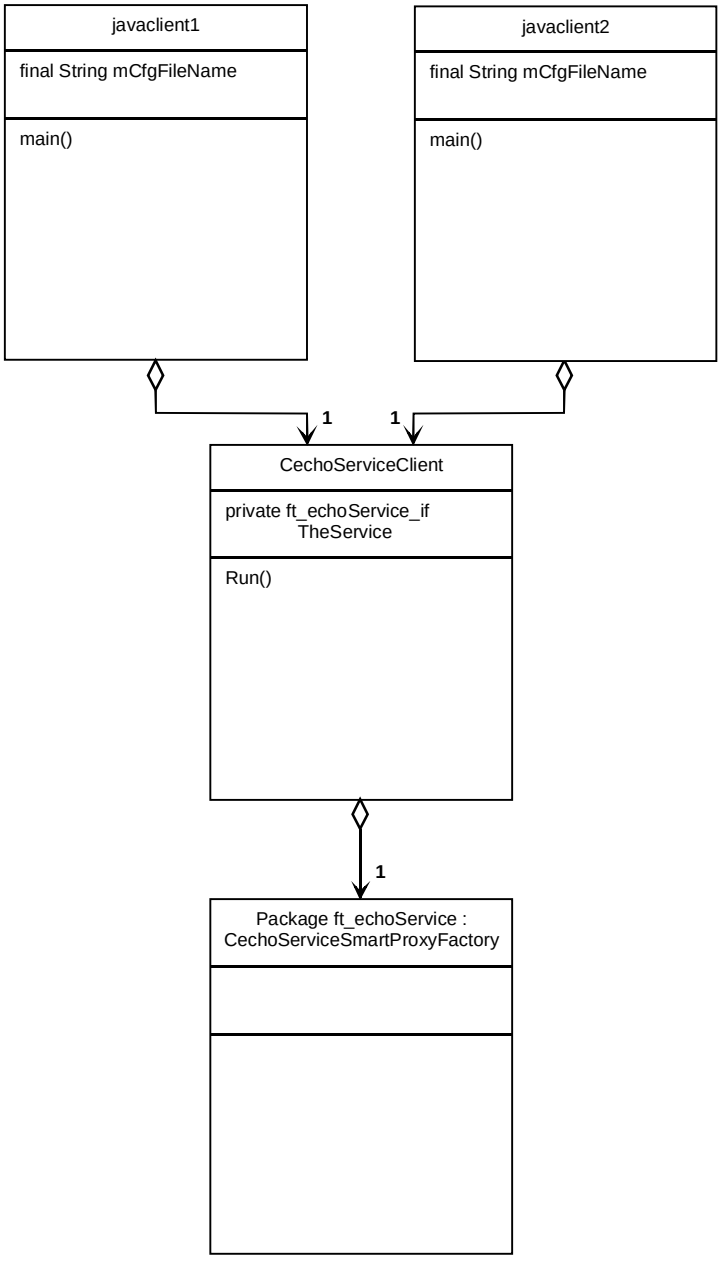

Figure 6-11. Client.

5.5.1.3 Server

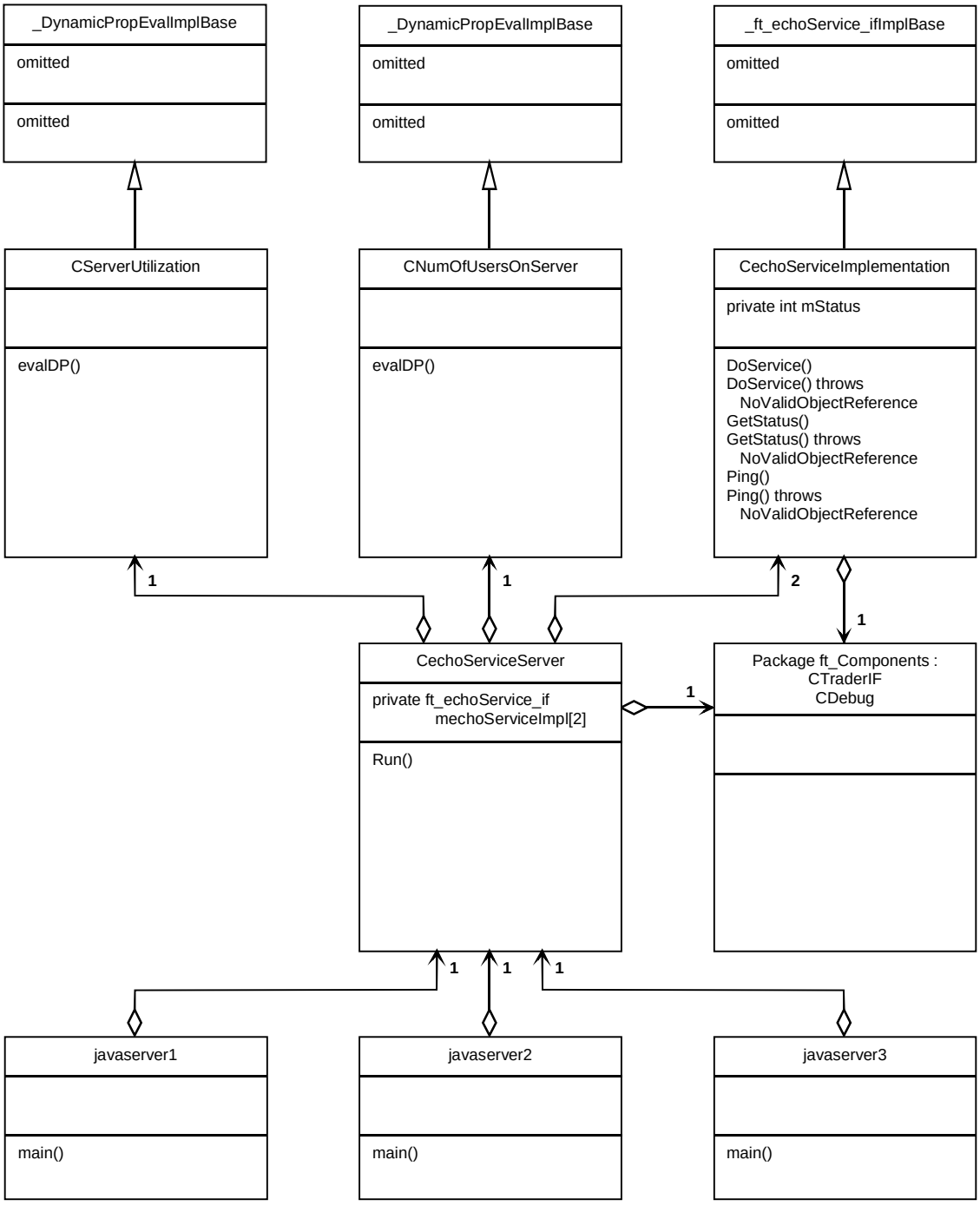

Figure 6-12. Server.

5.5.2 Package ft_echoService

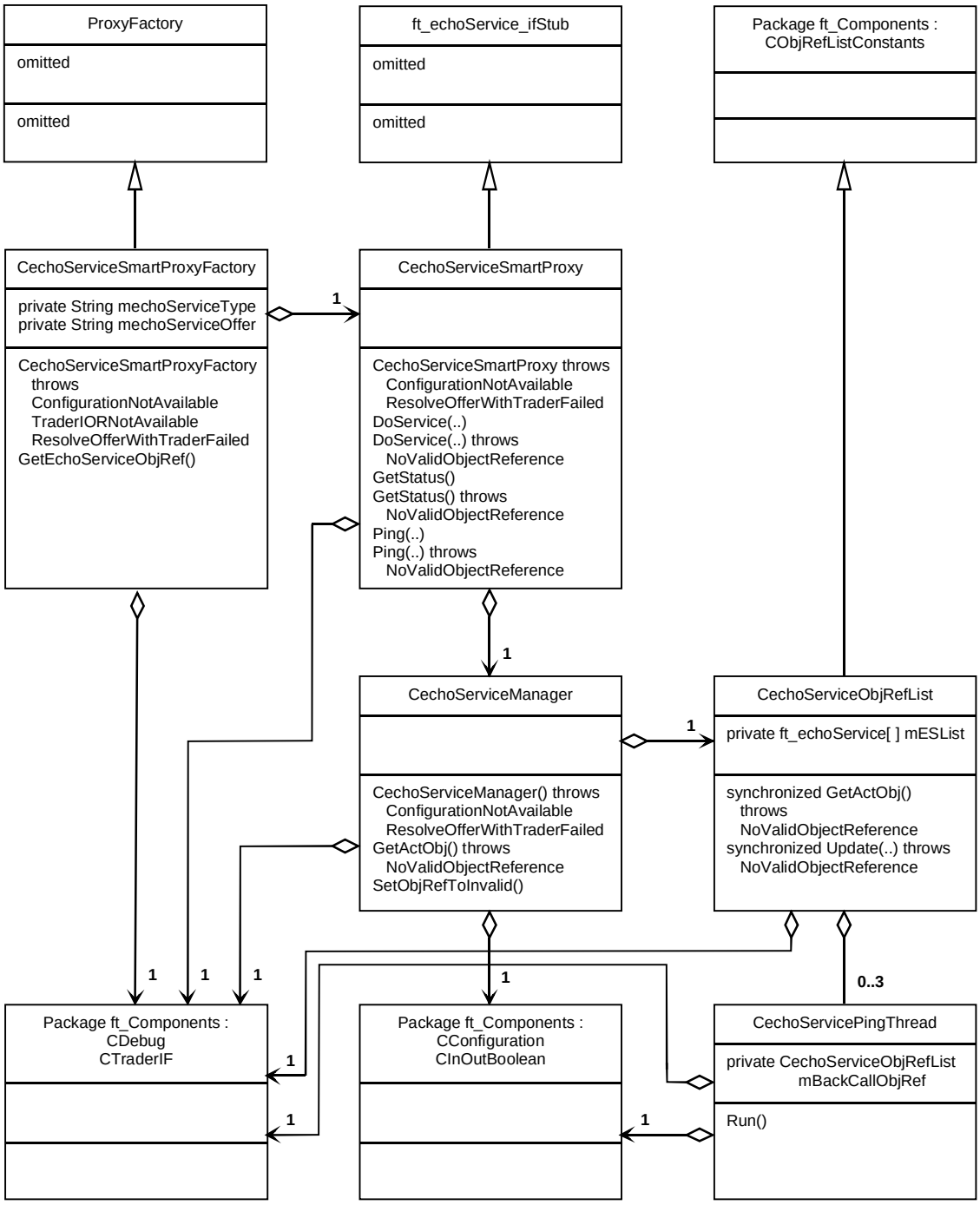

Figure 6-13. echoService.

5.5.3 Package ft_Components

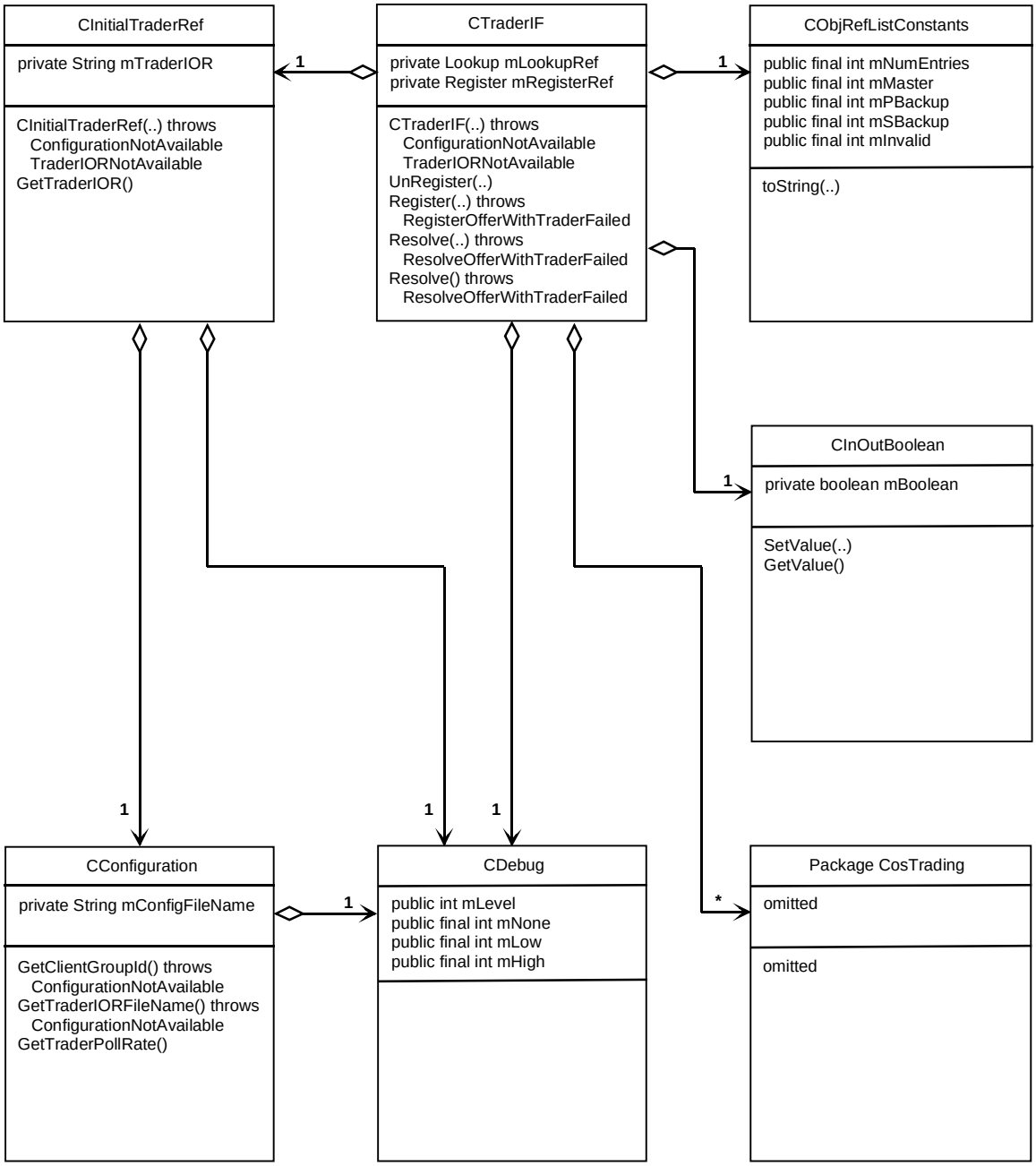

Figure 6-14. Trader Interface.

5.5.4 Package ft_Exceptions

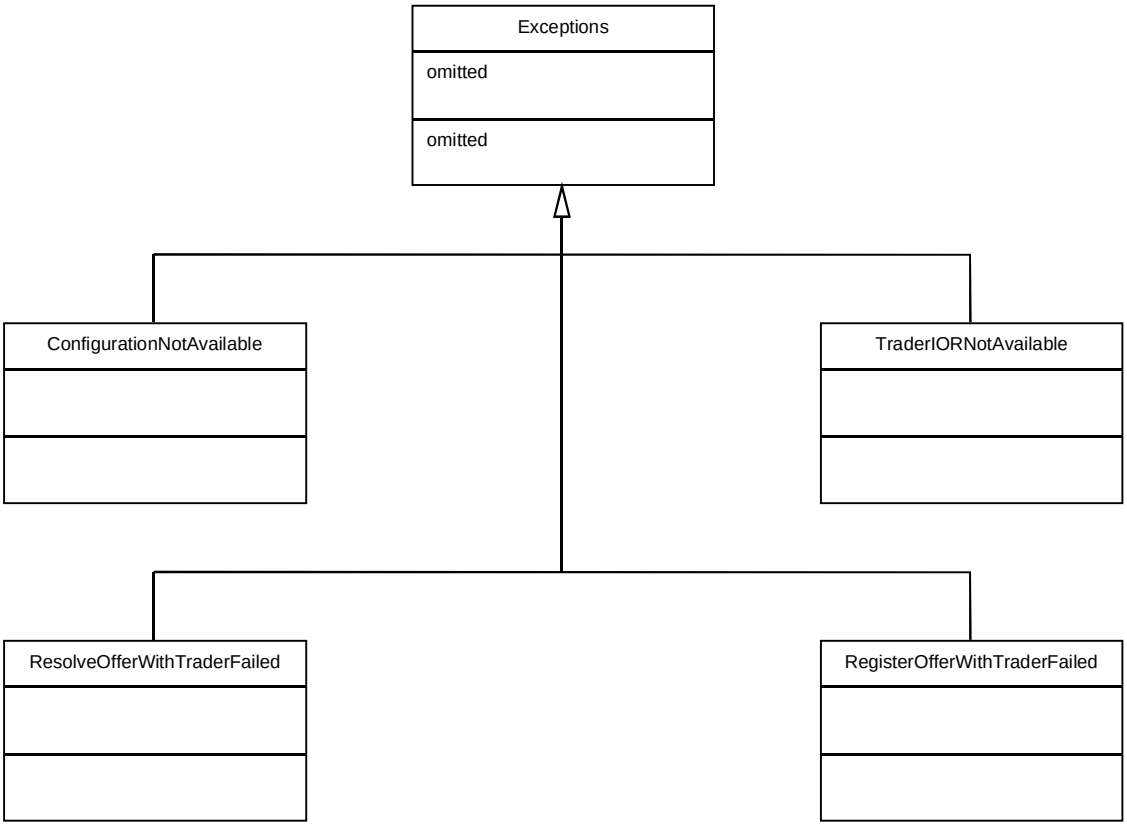

Figure 6-15. Exceptions.

5.6 Summary

This prototype implementation of the basic fault tolerance architecture, introduced in Chapter 4, was successfully tested on both, Sun Solaris and WindowsNT platforms. During integration testing, services where killed to force client smart proxies to re-connect to backup services and to re-connect to the original service as soon they were re-started.

The evaluation of the performance of this implementation is discussed in Chapter 6.

The source code of this prototype implementation can be found in Appendix B.

6. EVALUATION

6.1 Introduction

The performance of the implemented prototype is evaluated in this Chapter. The duration of method invocation on a service object running on a remote server, as well as re-connection to backup and re-connection to original service object running on a remote server was measured.

The method invocation duration times were taken for 1000 invocations on the GetStatus and DoService methods. The results were divided by 1000 to get the duration of a single invocation. The server was once connected using bind and once using the fault tolerance framework provided.

Service re-connection duration was measured by taking the time of an interrupted service invocation.

All measurements were made with OrbixDaemon, OrbixTrader, client and server process running on the same Sun Solaris Ultra SPARC box and OrbixWeb3.0's default configuration. The result of these measurements can be found in the following diagrams and tables.

6.2 Method Invocation Duration

Sample	1	2	3	4	5	6
GetStatus (bind) [Milliseconds]	3.961	3.946	4.174	4.161	3.922	3.903
GetStatus (fault tolerant) [Milliseconds]	4.021	4.034	4.041	4.006	4.034	4.013
DoService (bind) [Milliseconds]	5.089	4.911	5.409	5.265	4.945	4.848
DoService (fault tolerant) [Milliseconds]	5.209	4.988	4.984	4.998	4.995	4.986

Table 6-1. Method Invocation Duration (A).

Sample	7	8	9	10	11	12	Average
GetStatus (bind) [Milliseconds]	3.959	3.942	4.083	4.033	3.903	3.932	3.993
GetStatus (fault tolerant) [Milliseconds]	4.061	4.056	4.043	4.051	4.008	4.009	4.031
DoService (bind) [Milliseconds]	5.255	4.909	5.023	4.970	4.899	4.835	5.030
DoService (fault tolerant) [Milliseconds]	4.994	4.983	4.988	5.072	5.196	4.979	5.031

Table 6-2. Method Invocation Duration (B).

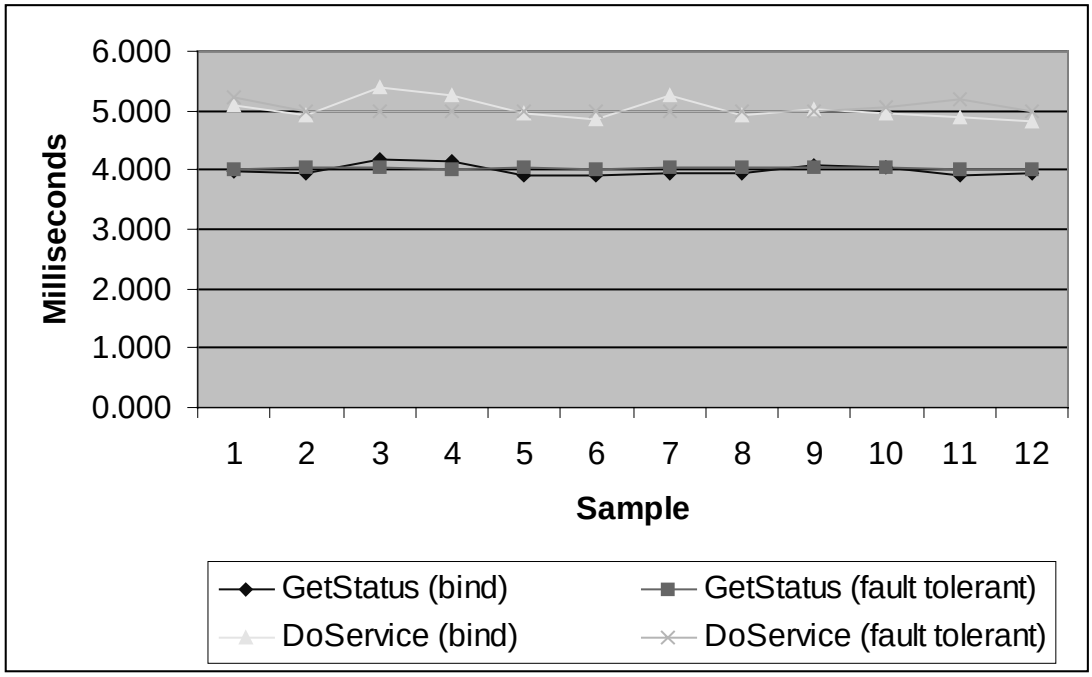

Figure 6-1. Method Invocation Duration.

6.3 Service Re-connection Duration

Sample	1	2	3	4	5	6
Re-connecting to backup service [Seconds]	12.227	12.217	14.521	12.166	14.615	12.155
Re-connecting to original service [Seconds]	1.785	1.695	1.946	1.686	1.695	1.766

Table 6-3. Service Re-connection Duration (A).

Sample	7	8	9	10	11	12	Average
Re-connecting to backup service [Seconds]	12.229	12.328	12.303	14.568	12.254	12.271	12.821
Re-connecting to original service [Seconds]	1.675	1.957	1.780	1.696	1.747	1.726	1.763

Table 6-4. Service Re-connection Duration (B).

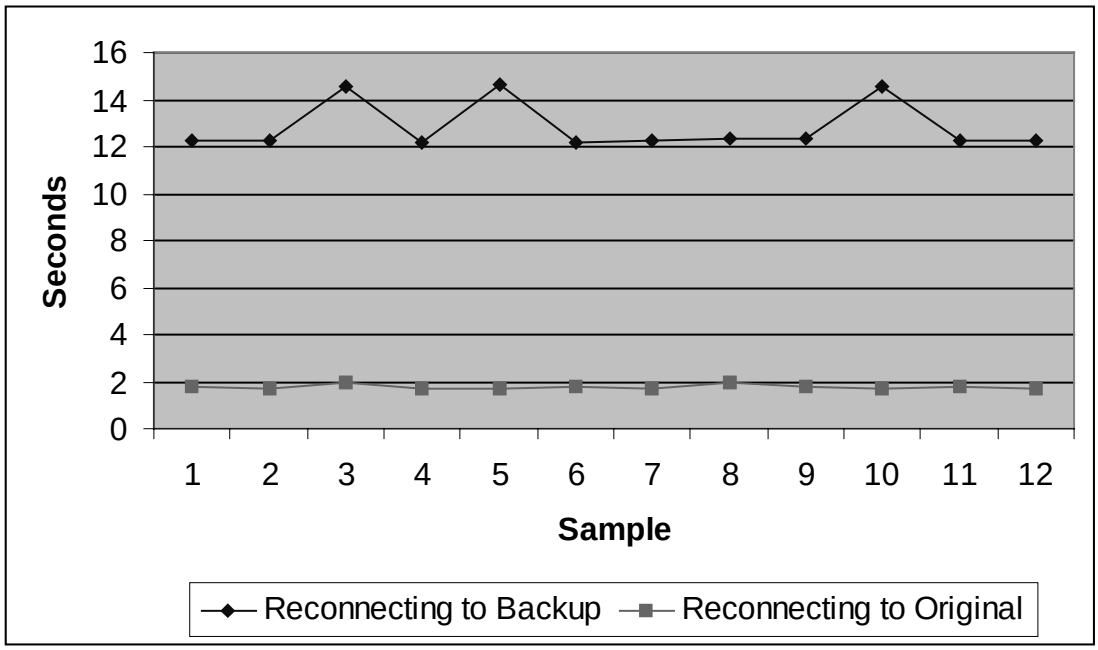

Figure 6-2. Service Re-connection Duration.

6.4 Summary

The method invocation measurements show that invocation time on a bound service object and a fault tolerant service object are *identical*. The difference between the two is that the invocation on the fault tolerant service object uses the provided smart proxy, whereas the invocation on the binded service object uses the default proxy. These are (almost) identical in absence of a service failure.

The re-connection to backup service consists of *failure detection*, which uses OrbixWeb3.0's COMM_FAILURE timeout, *message transmission*, which depends on the network topology and *client algorithms*. Client algorithms include a selection algorithm, to choose the best available backup service object reference, a cache update algorithm, to mark the invalid service object reference, and a thread creation and start algorithm, to ping the failed service object. OrbixWeb3.0's COMM_FAILURE timeout is configurable and could be reduced in order to increase performance. This should be done carefully, considering possible network delays.

The re-connection to original service consists of pinging and therefore starting up the original (failed) service, updating the client's cache, selecting the best available service object reference and killing the ping-thread.

Re-connection to backup service and re-connection to original service, both may cause service user idle time. During re-connection to backup service user idle time cannot be avoided but might be reduced by reconfiguring the COMM_FAILURE timeout. The measurements show that user idle time is within an acceptable range.

The measured re-connection to original service duration times are worst case values. There will be no user idle time unless service invocation on backup and re-connection to original service happened concurrent. Even in worst case, the user idle time is minor.

7. CONCLUSIONS

7.1 Thesis Review

The main goal of this thesis is to describe a body of research into the *fault tolerance problem* associated with the use of large scale distributed systems and to describe a partial solution to the problem. This took the form of a framework to support the development of fault tolerant distributed application components using OMG CORBA.

Fault tolerance problems arise in such an environment because application components may eventually fail due to hardware problems, operator mistakes or software failures. Within a banking environment, such failures are not acceptable. OMG CORBA does not address fault tolerance appropriately, thus fault tolerance mechanisms must be employed to reduce the susceptibility of a given system to failure. Of primary importance to our solution is the idea to include the OMG CORBA Trading Service into the fault tolerance architecture as the mechanism to advertise and manage service offers for fault tolerant application components. The work was structured as follows :

We began by re-evaluating the chosen banking environment, which is WDR's Jetpac infrastructure, and by examining the role of each component in it. This led to the identification of possible causes of application component failures and finally, to the examination of the requirements to enable fault tolerance to be build into the Jetpac infrastructure. This is documented in Chapter 2.

In Chapter 3, fault tolerance related issues and object middleware solutions were reviewed. This was followed by the evaluation of relevant CORBA features by implementing application scenarios.

With a clear understanding of the problem a solution was designed. This was focussed be the development of a basic architecture that builds fault tolerance into the Jetpac infrastructure. To increase fault tolerance related performance, improvements on the basic architecture were developed by making use of the Jetpac notification service (LNC). This is documented in Chapter 4.

The prototype implementation of the basic architecture is presented in Chapter 5 and 6. Chapter 5 starts with the description of some implementation choices including the framework package structure, the IDL interface extension and the usage of the fault tolerant components. It also includes the *Uniform Modelling Language* (UML) notation for graphically

describing the implemented prototype. Chapter 6 was devoted to the performance evaluation of the implemented prototype.

7.2 Achievements

An architecture that fits into the Jetpac infrastructure and allows the development of fault tolerant distributed application components was designed. Including this architecture into an existing infrastructure results in a highly available system as stated in Chapter 1.

The suggested architecture includes the OMG CORBA Trading Service, that allows application component management by configuration adjustments, without re-booting the system. The chosen OMG CORBA Trading Service configuration supports the different needs of Jetpac clients and the scalability requirements of the Jetpac infrastructure.

The suggested architecture has been realised in this framework that also includes the implementation of some of the fault tolerance mechanisms proposed. The implementation was evaluated as described in Chapter 6. It is shown that performance issues are addressed appropriately and performance can be improved by fully implementing the suggested architecture.

Finally, the limitations 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, so called smart proxy, was used. This issue is subject of further research.

7.3 Future Developments

As with all projects, there remains ample room for research on the problem of fault tolerance and the suggested framework.

Smart Proxy Generator

The package ft_echoService is IDL interface specific. It implements client smart proxy classes and related fault tolerance algorithms which can, for efficiency reasons, not be separated. In order to generate this package for different IDL interfaces, a compiler can be implemented.

IDL Call-back Object

In order to exchange additional information between the client smart proxy class and the client application program, the IDL interface may be extended by a call-back object. Such a call-back object could implement an early reply server to transmit the smart proxy status to the application program.

ORB Independence

The Jetpac infrastructure was developed using Iona's OrbixWeb. By the very end of this project, OrbixWeb was replaced by Visigenic's VisiBroker. To support ORB independence between these two ORB's, the smart proxy generator could not only generate OrbixWeb smart proxies, but also VisiBroker smart stubs.

Jetpac and Visigenic

Additional research may be considered on the impact of the use of VisiBroker instead of OrbixWeb in the Jetpac infrastructure and the suggested fault tolerance framework.

Pass-by-Value

An impact analysis of the extensive use of objects pass-by-value on the suggested fault tolerance framework might be considered.

7.4 Concluding Remarks

This thesis presents the research, design and implementation of a framework to provide fault tolerance within an existing environment using the CORBA Trading Service. The chosen banking environment is WDR's Jetpac infrastructure. The implementation was evaluated with a prototype application.

In conclusion, it has been demonstrated that this framework supports the development of fault tolerant distributed application components, which results in a *highly available* system.

8. GLOSSARY

Client	A computer program for which a server performs some computation, compare user [Bacon, 1993].
COM	Component Object Model.
CORBA	Common Object Request Broker Architecture [Orfali et al., 1997].
Customer	Mr. John Nichol, Warburg Dillon Read, 2 Finsbury, London, EC2M 2PP, Tel.: +44 171 568 0390 Email : john.nichol@wdr.com
DCOM	Distributed Component Object Model [Orfali et al., 1997].
GUI	Graphical User Interface.
IDL	Interface Definition Language [Orfali et al., 1997].
IIOP	Internet Inter ORB Protocol [Orfali et al., 1997].
IOR	Interoperable Object Reference [Orfali et al., 1997].
IT	Information Technology.
JDK	Java Development Kit.
LNC	Jetpac Notification Service.
MOM	Message Oriented Middleware [Baker et al., 1997].
OLE	Object Linking and Embedding.
ORB	Object Request Broker [Orfali et al., 1997].
QoS	Quality of Service.
RMI	Remote Method Invocation.
Server	A computer program designed to perform computations on behalf of other programs, its clients [Bacon, 1993].
SM	Jetpac Service Manager.
TPM	Transaction Processing Monitors [Baker et al., 1997].
TSM	Trading Service Manager.
UML	Unified Modelling Language.
User	A human operator of computer system, compare client [Bacon, 1993].
WDR	Warburg Dillon Read

9. REFERENCES

[Bacon, 1993]
Bacon J., Concurrent Systems. Addison-Wesley, 1993.

[Baker, 1997]
Baker S., *CORBA Distributed Objects Using Orbix*. Addison-Wesley, 1997.

[Baker et al., 1997]
Baker S., Cahill W. and Nixon P.*, Bridging Boundaries - CORBA in Perspective*. IEEE Internet Computing, Volume 1, Number 5, September/November 1997.

[Banâtre and Lee, 1994]
Banâtre M. and Lee P. A., *Hardware and Software Architectures for Fault Tolerance*. Springer Verlag, 1994.

[Gray, 1993]
Gray J. and Reuter A., *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[Lee, 1990]
Lee P. A. and Anderson T., *Fault Tolerance: Principles and Practice (second edition).* Springer Verlag, 1990.

[Maffeis et al., 1997]
Maffeis S. and Schmidt D. C., *Constructing Reliable Distributed Communication Systems with CORBA*. IEEE Communications Magazine 14(2),
URL = http://www.softwired.ch/people/maffeis/publications.html, February 1997.

[Mullender, 1993]
Mullender S., *Distributed Systems.* Addison-Wesley,1993.

[Orfali et al., 1997]
Orfali R., Harkey D. and Edwards J., *Instant CORBA*. John Wiley & Sons Inc., 1997.

10. BIBLIOGRAPHY

[Avizienis, 1984]
Avizienis A. and Kelly J., *Fault Tolerance by Design Diversity: Concepts and Experiments*. IEEE Computer, UCLA Computer Science Department, August 1984.

[Cahill, 1997]
Cahill V., *Fault Tolerance and Transactions in Distributed Systems*. Lecture Notes for Computer Science NDS103, TCD, 1997.

[Deitel, 1997]
Deitel H. and Deitel P., *Java How to Program*. Prentice-Hall International Inc., 1997.

[Flaviu, 1991]
Flaviu C., *Understanding Fault Tolerant Distributed Systems*. Communications of the ACM, Volume 34, Number 2, February 1991.

[Iona, 1997]
Iona Technologies PLC, *OrbixWeb Programmer's Guide*. URL = http://www.iona.com, 1997.

[Iona, 1998]
Iona Technologies PLC, *OrbixTrader Programmer's Guide and Reference*. URL = http://www.iona.com, 1998.

[Lamport et al., 1982]
Lamport L., Shostak R. and Pease M., *The Byzantine Generals Problem*. ACM Transactions on Programming Languages and Systems, Volume 4, Number 3, July 1982.

[Landis et al., 1997]
Landis S. and Maffeis S., *Building Reliable Distributed Systems with CORBA*. Theory and Practice of Object Systems, John Wiley, New York,
URL = http://www.softwired.ch/people/maffeis/publications.html, April 1997.

[Maffeis, 1997]
Maffeis S., *Client/Server Term Definition, in Encyclopedia of Computer Science*. URL = http://www.softwired.ch/people/maffeis/publications.html, 1997.

[Raynal, 1988]
Raynal M., *Distributed Algorithms and Protocols*. John Wiley & Sons, 1988.

[Visigenic, 1998]
Visigenic Software Inc, *Visigenic Programmers Guide Version 3.2, VisiBroker for Java*. URL = http://www.inprise.com, 1998.

[Zomaya 1996]
Zomaya A., *Parallel & Distributed Computing Handbook*. McGraw-Hill, 1996.

11. APPENDIX A – SOURCE CODE FOR EVALUATION SCENARIOS

This appendix contains the source code for the evaluation scenarios described in Chapter 2.

A1 - Echo Service Application

```
// echoServiceApplication.idl

interface echoService
{
  string DoService(in string Msg);
  long   GetStatus();
};

/************************************************************************/

package echoServiceApplicationPackage;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;

public class javaserver1 {

  public static void main(String args[]) {

    echoService echoService1Impl = null;
    echoService echoService2Impl = null;
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

    try {
     echoService1Impl = new EchoServiceImplementation("Service One",1100);
     echoService2Impl = new EchoServiceImplementation("Service Two",1200);

     _CORBA.Orbix.impl_is_ready("echoServer1");

     System.out.println("Shutting down server...");
     orb.disconnect(echoService1Impl);
     orb.disconnect(echoService2Impl);

    }
    catch(org.omg.CORBA.SystemException se) {
      System.out.println("Exception raised during creation of EchoService_Implementation" +
se.toString());
      System.exit(1);
    }
    System.out.println ("Server exiting....");
  }// end main
}// end class

/************************************************************************/

package echoServiceApplicationPackage;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;

public class javaserver2 {

  public static void main(String args[]) {

    echoService echoService1Impl = null;
    echoService echoService2Impl = null;
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

    try {
     echoService1Impl = new EchoServiceImplementation("Service One",2100);
     echoService2Impl = new EchoServiceImplementation("Service Two",2200);

     _CORBA.Orbix.impl_is_ready("echoServer2");

     System.out.println("Shutting down server...");
     orb.disconnect(echoService1Impl);
```

```
     orb.disconnect(echoService2Impl);

    }
    catch(org.omg.CORBA.SystemException se) {
      System.out.println("Exception raised during creation of EchoService_Implementation" +
se.toString());
      System.exit(1);
    }
    System.out.println ("Server exiting....");
  }// end main
}// end class

/**************************************************************************/

package echoServiceApplicationPackage;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;

public class javaserver3 {

  public static void main(String args[]) {

    echoService echoService1Impl = null;
    echoService echoService2Impl = null;
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

    try {
     echoService1Impl = new EchoServiceImplementation("Service One",3100);
     echoService2Impl = new EchoServiceImplementation("Service Two",3200);

     _CORBA.Orbix.impl_is_ready("echoServer3");

     System.out.println("Shutting down server...");
     orb.disconnect(echoService1Impl);
     orb.disconnect(echoService2Impl);

    }
    catch(org.omg.CORBA.SystemException se) {
      System.out.println("Exception raised during creation of EchoService_Implementation" +
se.toString());
      System.exit(1);
    }
    System.out.println ("Server exiting....");
  }// end main
}// end class

/**************************************************************************/

package echoServiceApplicationPackage;

import IE.Iona.OrbixWeb._OrbixWeb;

class EchoServiceImplementation extends _echoServiceImplBase {
  private int mStatus = 0;// counts the number of service calls

  // constructor
  public EchoServiceImplementation(String Marker,int Status) {
    super(Marker);
    mStatus = Status;
  }// end constructor

  // constructor
  public EchoServiceImplementation() {
    super("NoName service");
    mStatus = 0;
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // implementation of the echo service operation
  public String DoService(String Msg) {
    ++mStatus;
    String ServiceName = _OrbixWeb.Object(this)._marker();
    System.out.println("Server : Received message = " + Msg);
    String Answer = new String(Msg+" - "+ServiceName+" : Echo service answer!");
    return Answer;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
  // implementation of the echo service status operation
  public int GetStatus() {
    return mStatus;
  }// end method
}// end class

/*************************************************************************/

package echoServiceApplicationPackage;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class javaclient1 {
  public static void main(String args[]) {
    ORB.init(args,null);

    // read the command line parameter
    CCommandLine cln = new CCommandLine(args);

    // start the client application
    CEchoServiceClient clt =
      new CEchoServiceClient(cln.GetProtocol(),cln.GetHost());
    clt.Run();
  }// end main
}// end class

/*************************************************************************/

package echoServiceApplicationPackage;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class CCommandLine {
  private String[] Properties = new String[2];

  public CCommandLine(String Args[]) {
   /*
    * Command line syntax is as follows...
    *
    * javaclient1 [<IIOP>] [<hostname>]
    *
    * If the "IIOP" parameter is given then the client will use the
    * IIOP protocol when taking to the orbixd and the server. The
    * hostname parameter specifies where to find the orbixd.
    */

    if (Args.length < 1) { // use the Orbix Protocol and the local orbixd
       Properties[0] = new String(_CORBA.Orbix.myHost());
       Properties[1] = new String("IIOP");
    }
    else if (Args[0].equals("IIOP")) {
       Properties[1] = new String("IIOP");
       if (Args.length < 2)
          Properties[0] = new String(_CORBA.Orbix.myHost());
       else
          Properties[0] = new String(Args[1]);
    }
    else {
       Properties[0] = new String(Args[0]);
       Properties[1] = new String("POOP");
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String GetProtocol() {
     return Properties[1];
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String GetHost() {
     return Properties[0];
  }// end method
}// end class

/*************************************************************************/
```

```
package echoServiceApplicationPackage;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class CEchoServiceClient {
  private final int NumServer  = 3;
  private final int NumService = 2;
  private echoService[][] es = new echoService[NumServer][NumService];
  private echoService es1_1 = null;
  private echoService es1_2 = null;
  private echoService es2_1 = null;
  private echoService es2_2 = null;
  private echoService es3_1 = null;
  private echoService es3_2 = null;

  public CEchoServiceClient(String Protocol, String Host) {
    // do the binding
    if(Protocol.equals("IIOP"))
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(true));
    else
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(false));

    try {
      for (int Server=0; Server<NumServer; Server++) {
        es[Server][0] = null;
        es[Server][1] = null;
        es[Server][0] =
          echoServiceHelper.bind("Service One:echoServer"+(Server+1),Host);
        es[Server][1] =
          echoServiceHelper.bind("Service Two:echoServer"+(Server+1),Host);
      }

      es1_1 = echoServiceHelper.bind("Service One:echoServer1",Host);
      es1_2 = echoServiceHelper.bind("Service Two:echoServer1",Host);
      es2_1 = echoServiceHelper.bind("Service One:echoServer2",Host);
      es2_2 = echoServiceHelper.bind("Service Two:echoServer2",Host);
      es3_1 = echoServiceHelper.bind("Service One:echoServer3",Host);
      es3_2 = echoServiceHelper.bind("Service Two:echoServer3",Host);
    }
    catch (org.omg.CORBA.SystemException ex) {
      System.out.println("Exception during bind");
      System.out.println(ex.toString());
      ex.printStackTrace();
      System.exit(1);
    }
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void Run() {
    for (int Server=0; Server<NumServer; Server++) {
      System.out.println("");
      System.out.println("*****************************");
      System.out.println("*  Connecting echoServer"+(Server+1)+" :  *");
      System.out.println("*****************************");
      CommWithServer(es[Server]);
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Call the methodes of a server */
  public void CommWithServer(echoService es[]) {
    for (int Service=0; Service<NumService; Service++) {
      for (int i=0; i<5; i++) {
        PerformGetStatus(es[Service]);
        PerformDoService(es[Service]);
         System.out.println("- - - - - - - - -");
      }
      System.out.println("----------------------------------");
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Get the status of the remote echo object */
  public void PerformGetStatus(echoService es) {
    int EchoServiceStatus;
    try {
      EchoServiceStatus = es.GetStatus();
```

```
    } catch (org.omg.CORBA.SystemException ex) {
      System.out.println("FAIL\tException during GetStatus");
      System.out.println(ex.toString());
      return;
    }
    System.out.println("Echo service status = " + EchoServiceStatus);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Call the echo service */
  public void PerformDoService(echoService es) {
    String Msg    = new String("Client echo message");
    String Answer = null;
    try {
      Answer = new String(es.DoService(Msg));
    } catch (org.omg.CORBA.SystemException ex) {
      System.out.println("FAIL\tException during DoService");
      System.out.println(ex.toString());
      System.out.println("Minor=" + ex.minor);
      System.exit(1);
    }
    System.out.println("Echo service answer = " + Answer);
  }// end method
}// end class
```

A2 - Echo Service Applet

```
// echo.idl

interface echoService {

  string Service(in string Msg);
  long Status();
};

/*************************************************************************/

package echoServiceApplet;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;

public class javaserver1 {

  public static void main(String args[]) {

    echoService echoImpl = null;
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

    try {
     echoImpl = new EchoServiceImplementation(100);

     _CORBA.Orbix.impl_is_ready("echoServer",1000*60*5);//set server timeout

     System.out.println("Shutting down server...");
     orb.disconnect(echoImpl);

    }
    catch(org.omg.CORBA.SystemException se) {
      System.out.println("Exception raised during creation of Echo_Implementation" +
se.toString());
      System.exit(1);
    }

    System.out.println ("Server exiting....");
  }// end main
}// end class

/*************************************************************************/

class EchoServiceImplementation extends _echoServiceImplBase {
  private CEchoServer m_EchoServer;

  public EchoServiceImplementation(int Status) {
    m_EchoServer = new CEchoServer(Status);
  }// end method

  // echo service operation
  public String Service(String Msg) {
    return m_EchoServer.Service(Msg);
  }// end method

  // status echo operation
  public int Status() {
    return m_EchoServer.Status();
  }// end method
}// end class

/*************************************************************************/

package echoServiceApplet;

public class CEchoServer {
  private int m_Status = 0;

  // constructor
  public CEchoServer(int Status) {
    m_Status = Status;
  }// end constructor
```

```
  // implementation of the echo service operation
  public String Service(String Msg) {
    System.out.println("Server : Received message = " + Msg);
    String Answer = new String(Msg+" - This is the echo answer!");
    return Answer;
  }// end method

  // implementation of the service status operation
  public int Status() {
    ++m_Status;
    return m_Status;
  }// end method
}// end class

/**************************************************************************/

package echoServiceApplet;

import org.omg.CORBA.SystemException;
import org.omg.CORBA.INITIALIZE;
import java.applet.*;
import java.awt.*;
import org.omg.CORBA.ORB;

public class EchoApplet extends Applet {
  // main display panel
  EchoEvents echoEvents;

  public void init () {
    try {
      ORB.init(this, null);
    }
    catch (INITIALIZE ex) {
      System.err.println ("failed to initialize: "+ex);
    }

    echoEvents = new EchoEvents ();
    // add panel to applet
    this.add (echoEvents);
  }// end method
}// end class

/**************************************************************************/

package echoServiceApplet;

import java.awt.*;
import java.lang.*;
import org.omg.CORBA.SystemException;
import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class EchoEvents extends EchoPanel {
  // grid proxy object
  public echoService eRef;

  public EchoEvents() {
    super();
    ORB.init();
    nameField.setText(":echoServer");
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public boolean action (Event event, Object arg) {
    if (conBStr.equals (arg)) {
      bindObject ();
    }
    else if (disBStr.equals (arg)) {
      eRef = null;
      displayMsg ("Disconnected.");
    }
    else if (staBStr.equals (arg)) {
      getStatus ();
    }
    else if (serBStr.equals (arg)) {
      performService ();
    }
    return true;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
  public void bindObject () {
    String tmp;
    String markerServer;
    String hostName;

    // get server name from text field
    if ((tmp = nameField.getText ()) == null)
      markerServer = "";
    else
      markerServer = ":" + tmp;

    // get host name from text field
    hostName = hostField.getText ();

    // bind to server object
    _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(true));
    try {
      eRef = echoServiceHelper.bind (tmp , hostName);
    }
    catch (SystemException se) {
      displayMsg ("Connect failed.\n" + "Unexpected exception:\n" + se.toString ());
      return;
    }

    displayMsg ("Connect succeeded.");
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void getStatus () {

    int ServiceStatus = 0;

    // check that proxy exists
    if (eRef == null) {
      displayMsg ("Get status failed - not connected to server.");
      return;
    }

    // call the status methode
    try {
      ServiceStatus = eRef.Status();
    }
    catch (SystemException se) {
      displayMsg ("Get status failed.\n" + "Unexpected exception:\n" + se.toString ());
      return;
    }

    sField.setText (Integer.toString (ServiceStatus));
    displayMsg ("Get status succeeded.");
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void performService () {
    String Msg    = null;
    String Answer = null;

    // check that proxy exists
    if (eRef == null) {
      displayMsg ("Perform service failed - not connected to server.");
      return;
    }

    // get input string from text fields
    Msg = new String(iField.getText());

    // call the service
    try {
      Answer = new String(eRef.Service(Msg));
    }
    catch (SystemException se) {
      displayMsg ("Perform service failed.\n" + "Unexpected exception:\n" + se.toString ());
      return;
    }

    aField.setText(Answer);
    displayMsg ("Perform service succeeded.");
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
  void displayMsg (String msg) {
    MsgDialog msgDlog = new MsgDialog("EchoService Operation Result", msg);
    msgDlog.resize (380, 200);
    msgDlog.show ();
  }// end method
}// end class

/**************************************************************************/

package echoServiceApplet;

import java.awt.*;

public class EchoPanel extends Panel {
  // button string constants
  final String conBStr = "Connect";
  final String disBStr = "Disconnect";
  final String staBStr = "Get Service Status";
  final String serBStr = "Perform Service";

  // components for _bind info
  Label nameL = new Label ("Server Name");// _bind labels
  Label hostL = new Label ("Server Host");
  TextField nameField;                    // _bind text fields
  TextField hostField;
  Button connectButton;                   // _bind buttons
  Button disconnectButton;

  // components for grid operation info
  Label sL = new Label ("Service Status =");       // operation labels
  Label iL = new Label ("Service input =");
  Label aL = new Label ("Service answer =");
  TextField sField;                                // operation text fields
  TextField iField;
  TextField aField;
  Button getSButton;                               // operation buttons
  Button perSButton;

  // sub panels
  Panel bindPanel = new Panel ();
  Panel botPanel = new Panel ();

  public EchoPanel () {
    GridBagLayout ebLayout;

    // create text fields
    nameField = new TextField (16);
    hostField = new TextField (16);
    sField = new TextField (9);
    sField.setEditable (false);
    iField = new TextField (25);
    aField = new TextField (25);
    aField.setEditable (false);

    // create buttons
    connectButton = new Button (conBStr);
    disconnectButton = new Button (disBStr);
    getSButton = new Button (staBStr);
    perSButton = new Button (serBStr);

    // set layouts
    ebLayout = new GridBagLayout ();

    bindPanel.setLayout (ebLayout);
    botPanel.setLayout (ebLayout);
    this.setLayout (ebLayout);

    // add components to panels
    constrain (bindPanel, new Label (""), 0, 0, 1, 1);
    constrain (bindPanel, new Label ("Server Details:"), 0, 1, 1, 1,
      GridBagConstraints.HORIZONTAL);
    constrain (bindPanel, new Label (""), 0, 2, 1, 1);
    constrain (bindPanel, nameL, 0, 3, 1, 1,
      GridBagConstraints.HORIZONTAL);
    constrain (bindPanel, nameField, 1, 3, 1, 1,
      GridBagConstraints.HORIZONTAL);
    constrain (bindPanel, new Label (""), 2, 3, 1, 1);
    constrain (bindPanel, connectButton, 3, 3, 1, 1,
      GridBagConstraints.HORIZONTAL);
    constrain (bindPanel, hostL, 0, 4, 1, 1,
      GridBagConstraints.HORIZONTAL);
```

```
    constrain (bindPanel, hostField, 1, 4, 1, 1,
      GridBagConstraints.HORIZONTAL);
    constrain (bindPanel, new Label (""), 2, 4, 1, 1);
    constrain (bindPanel, disconnectButton, 3, 4, 1, 1,
      GridBagConstraints.HORIZONTAL);
    constrain (bindPanel, new Label (""), 0, 5, 1, 1);
    constrain (this, bindPanel, 0, 0, 1, 1);
    constrain (botPanel, new Label (""), 0, 0, 1, 1);
    constrain (botPanel, new Label ("Object Details:"), 0, 1, 1, 1);
    constrain (botPanel, new Label (""), 0, 2, 1, 1);
    constrain (botPanel, sL, 0, 3, 1, 1);
    constrain (botPanel, sField, 1, 3, 1, 1);
    constrain (botPanel, new Label (""), 1, 4, 1, 1);
    constrain (botPanel, iL, 0, 5, 1, 1);
    constrain (botPanel, iField, 1, 5, 1, 1);
    constrain (botPanel, aL, 0, 6, 1, 1);
    constrain (botPanel, aField, 1, 6, 1, 1);
    constrain (botPanel, new Label (""), 0, 8, 1, 1);
    constrain (botPanel, getSButton, 0, 9, 1, 1);
    constrain (botPanel, perSButton, 1, 9, 1, 1);
    constrain (this, botPanel, 0, 1, 1, 2);
  }// end constructor

  public void constrain (Container container, Component component, int grid_x, int grid_y,
                         int grid_width, int grid_height) {
    constrain (container, component, grid_x, grid_y, grid_width,
      grid_height, GridBagConstraints.NONE);
  }// end method

  public void constrain (Container container, Component component, int grid_x, int grid_y,
                         int grid_width, int grid_height, int fill) {
    GridBagConstraints gbConstraints = new GridBagConstraints ();
    gbConstraints.gridx = grid_x;
    gbConstraints.gridy = grid_y;
    gbConstraints.gridwidth = grid_width;
    gbConstraints.gridheight = grid_height;
    gbConstraints.fill = fill;
    gbConstraints.anchor = GridBagConstraints.NORTHWEST;
    gbConstraints.weightx = 0.0;
    gbConstraints.weighty = 0.0;

    ((GridBagLayout) container.getLayout ()).setConstraints (component, gbConstraints);
    container.add (component);
  }// end method
}// end class

/*****************************************************************************/

package echoServiceApplet;

import java.awt.*;

public class MsgDialog extends Frame {
  protected Button button;
  protected Msg label;

  public MsgDialog(String title, String message) {
    super (title);

    Panel buttonPanel;

    // set layout manager
    this.setLayout(new BorderLayout(15, 15));

    // create message component and add to window
    label = new Msg (message, 20, 20);
    this.add ("Center", label);

    // create button and add to window
    button = new Button ("Dismiss");
    buttonPanel = new Panel ();
    buttonPanel.setLayout (new FlowLayout (FlowLayout.CENTER, 15, 15));
    buttonPanel.add (button);
    this.add ("South", buttonPanel);

    // resize the window to the preferred size of its components
    this.pack();
  }// end constructor

  public boolean action(Event e, Object arg) {
    if (e.target == button) {
      // pop down window
```

```
      this.hide();
      this.dispose();
      return true;
    }
    else return false;
  }// end method

  public boolean gotFocus(Event e, Object arg) {
    // give focus to button
    button.requestFocus();
    return true;
  }// end method
}// end class

/**************************************************************************/

package echoServiceApplet;

import java.awt.*;
import java.util.*;

public class Msg extends Canvas {
  // constants
  public static final int LEFT = 0;
  public static final int CENTER = 1;
  public static final int RIGHT = 2;

  // member variables
  protected String[] lines;         // the lines of text to display
  protected int num_lines;          // the number of lines
  protected int margin_width;       // left and right margins
  protected int margin_height;      // top and bottom margins
  protected int line_height;        // total height of the font
  protected int line_ascent;        // font height above baseline
  protected int[] line_widths;      // how wide each line is
  protected int max_width;          // the width of the widest line
  protected int alignment = LEFT;   // the alignment of the text.

  // constructor
  public Msg (String label, int margin_width, int margin_height, int alignment) {
    newLabel(label);
    this.margin_width = margin_width;
    this.margin_height = margin_height;
    this.alignment = alignment;
  }// end constructor

  public Msg(String label, int margin_width, int margin_height) {
    this(label, margin_width, margin_height, LEFT);
  }// end constructor

  // methods
  protected void newLabel(String label) {
    StringTokenizer t = new StringTokenizer(label, "\n");
    num_lines = t.countTokens();
    lines = new String[num_lines];
    line_widths = new int[num_lines];
    for(int i = 0; i < num_lines; i++) lines[i] = t.nextToken();
  }// end method

  protected void measure() {
    FontMetrics fm = this.getFontMetrics(this.getFont());
    if (fm == null) return;

    line_height = fm.getHeight();
    line_ascent = fm.getAscent();
    max_width = 0;
    for(int i = 0; i < num_lines; i++) {
      line_widths[i] = fm.stringWidth(lines[i]);
      if (line_widths[i] > max_width) max_width = line_widths[i];
    }
  }// end method

  public void setLabel(String label) {
    newLabel(label);
    measure();
    repaint();
  }// end method

  public void setFont(Font f) {
    super.setFont(f);
    measure();
    repaint();
```

```
  }// end method

  public void setForeground(Color c) {
    super.setForeground(c);
    repaint();
  }// end method

  public void setAlignment(int a) {
    alignment = a; repaint();
  }// end method

  public void setMarginWidth(int mw) {
    margin_width = mw; repaint();
  }// end method

  public void setMarginHeight(int mh) {
    margin_height = mh; repaint();
  }// end method

  public int getAlignment() {
    return alignment;
  }// end method

  public int getMarginWidth() {
    return margin_width;
  }// end method

  public int getMarginHeight() {
    return margin_height;
  }// end method

  public void addNotify() {
    super.addNotify(); measure();
  }// end method

  public Dimension preferredSize() {
    return new Dimension(max_width + 2*margin_width, num_lines * line_height + 2*margin_height);
  }// end method

  public Dimension minimumSize() {
    return new Dimension(max_width, num_lines * line_height);
  }// end method

  public void paint(Graphics g) {
    int x, y;
    Dimension d = this.size();
    y = line_ascent + (d.height - num_lines * line_height)/2;
    for(int i = 0; i < num_lines; i++, y += line_height) {
      switch(alignment) {
        case LEFT:
          x = margin_width; break;
        case CENTER:
        default:
          x = (d.width - line_widths[i])/2; break;
        case RIGHT:
          x = d.width - margin_width - line_widths[i]; break;
      }
      g.drawString(lines[i], x, y);
    }
  }// end method
}// end class
```

A3 – Echo Service Smart Proxy Application

```
interface echoServiceSP {
  string Service(in string Msg);
  long   Status();
  string GetName ();
};

interface echoServiceManager {
  // Get the least loaded echo service
  echoServiceSP GetEchoService ();
};

/************************************************************************/

package echoServiceSmartProxy;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.SystemException;

public class javaserver1 {

  public static void main(String args[]) {

    echoServiceManager myEM = null;
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
    System.out.println("Server Startup");

    try {
      myEM = new EchoServiceManagerImplementation("EchoServerName");

      _CORBA.Orbix.impl_is_ready("echoServiceManager",1000*60*1);//set server timeout

      System.out.println("Shutting down server...");
      orb.disconnect(myEM);

    }
    catch(SystemException se) {
      System.out.println("Exception raised during creation of EchoServiceManagerImplementation" +
se.toString());
      System.exit(1);
    }

    System.out.println ("Server exiting....");
  }// end main
}// end class

/************************************************************************/

class EchoServiceManagerImplementation extends _echoServiceManagerImplBase {
  int nextES;
  echoServiceSP aES[] = new echoServiceSP[3];

  /* The contructors create 3 server objects with different marker names. */
  EchoServiceManagerImplementation() {
    super();
    nextES = 2;
    aES[0] = new EchoServiceImplementation("Echo Service One",100);
    aES[1] = new EchoServiceImplementation("Echo Service Two",200);
    aES[2] = new EchoServiceImplementation("Echo Service Three",300);
    System.out.println("Finished constructer");
  }// end constructor

  EchoServiceManagerImplementation(String Marker) {
    super(Marker);
    nextES = 2;
    aES[0] = new EchoServiceImplementation("Echo Service One",100);
    aES[1] = new EchoServiceImplementation("Echo Service Two",200);
    aES[2] = new EchoServiceImplementation("Echo Service Three",300);
    System.out.println("Finished constructer");
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
  /* GetEchoService returns one of the three echo services. */
  public echoServiceSP GetEchoService() {
    nextES = (nextES+1) % 3;

    echoServiceSP es = aES[nextES];
    return es;
  }// end method
}// end class

/*************************************************************************/

class EchoServiceImplementation extends _echoServiceSPImplBase {
  public int mStatus = 0;

  EchoServiceImplementation(String Marker,int Status) {
    super(Marker);
    Status = Status;
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // implementation of the echo service operation
  public String Service(String Msg) {
    System.out.println("Server : echo service was called...");
    String Answer = new String(Msg + " - " + mStatus + " - Echo answer");
    return Answer;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // implementation of the service status operation
  public int Status() {
    ++mStatus;
    return mStatus;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // implementation of the GetName operation
  public String GetName() {
    String Name = _OrbixWeb.Object(this)._marker();
    return Name;
  }// end method
}// end class

/*************************************************************************/

package echoServiceSmartProxy;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class javaclient1 {
  public static void main(String args[]) {
    ORB.init(args,null);

    // read the command line parameter
    CCommandLine cln = new CCommandLine(args);

    // start the client application
    CEchoSPClient clt = new CEchoSPClient(cln.GetProtocol(),cln.GetHost());
    clt.Run();
  }// end main
}// end class

/*************************************************************************/

package echoServiceSmartProxy;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.SystemException;
import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb.Features.ProxyFactory;

public class CEchoSPClient {
  private echoServiceSP      es = null;
  private echoServiceManager theEchoManager = null;

  public CEchoSPClient(String Protocol, String Host) {
    new SmartESFactory();  // instanciate the smart proxy factory
```

```
    // do the binding
    if(Protocol.equals("IIOP"))
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(true));
    else
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(false));

    try {
      es = echoServiceSPHelper.bind(":echoServiceManager",Host);
    }
    catch (org.omg.CORBA.SystemException ex) {
      System.out.println("Exception during bind");
      System.out.println(ex.toString());
      ex.printStackTrace();
      System.exit(1);
    }

    try {
      theEchoManager = echoServiceManagerHelper.bind ("EchoServerName:echoServiceManager",
      Host);
    }
    catch (org.omg.CORBA.SystemException ex) {
      System.out.println("Exception during bind to echoManager");
      System.out.println(ex.toString());
      ex.printStackTrace();
      System.exit (1);
    }

  }//end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void Run() {
     for (int i=0; i<5; i++) {
       System.out.println("-----------------------------");
       DoService();
       DoName();
       DoStatus();
     }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Get the status of the remote echo object */
  public void DoStatus() {
    int ServiceStatus;
    try {
      ServiceStatus = es.Status();
    } catch (org.omg.CORBA.SystemException ex) {
      System.out.println("FAIL\tException during Status");
      System.out.println(ex.toString());
      return;
    }
    System.out.println("Client : Service status counter = " + ServiceStatus);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Call the echo service */
  public void DoService() {
    String Msg = new String("Echo message..");
    String Answer = null;
    try {
      Answer = new String(es.Service(Msg));
    } catch (org.omg.CORBA.SystemException ex) {
      System.out.println("FAIL\tException during service");
      System.out.println(ex.toString());
      System.out.println("Minor=" + ex.minor);
      System.exit(1);
    }

    System.out.println("Client : Service answer : " + Answer);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Get the name of the remote echo object */
  public void DoName() {
    String ServiceName;
    try {
      ServiceName = es.GetName();
    } catch (org.omg.CORBA.SystemException ex) {
      System.out.println("FAIL\tException during GetName");
```

```
      System.out.println(ex.toString());
      return;
    }
    System.out.println("Client : Service name : " + ServiceName);
  }// end method

}// end class

/************************************************************************/

/* Smart echo service proxy class. */

class SmartES extends _echoServiceSPStub {
  // The Smart proxy's echo manager object.
  private echoServiceManager theEchoManager = null;
  private echoServiceSP      actES = null;

  public java.lang.String getName () {
    return "dummy";
  }// end method

  /* This smart proxy has a number echo service object */
  public SmartES(){
    try {
      theEchoManager = echoServiceManagerHelper.bind ("EchoServerName:echoServiceManager",
                 _CORBA.Orbix.myHost());
    }
    catch (org.omg.CORBA.SystemException ex) {
      System.out.println("Exception during bind to echoManager");
      System.out.println(ex.toString());
      ex.printStackTrace();
      System.exit (1);
    }
  }// end constructor

  /*
   * Every time crunch() is called on the smart proxy, it asks the
   * number cruncher manger object for a number cruncher object.
   * The manger object can then balance the load on it's Number
   * Crunchers.
   */
  public String Service(String Msg) {
    actES = null;
    try {
       actES = theEchoManager.GetEchoService();
       if (actES == null) {
         System.out.println("actNC is null");
       }
       else {
        System.out.println("Using service object ...\n" +
_OrbixWeb.Object(actES)._object_to_string());
        System.out.println("Using service object ...\n" + ORB.init().object_to_string(actES));
        }
    }
    catch (SystemException ex) {
         System.out.println("Exception: Can't get service");
         System.exit (1);
    }
    String Answer = new String(actES.Service(Msg));
    return Answer;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String GetName() {
    String Name = new String(actES.GetName());
    return Name;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public int Status() {
    int Stat = actES.Status();
    return Stat;
  }// end method

}// end class

/************************************************************************/

/* Smart echo proxy factory class.
   Used to create smart proxies during _bind() to an object of type "echoSP".
```

```
*/

class SmartESFactory extends ProxyFactory {

  private static boolean createProxy = true;

  public SmartESFactory () {
    super (echoServiceSPHelper.id());
  }// end constructor

  /* Called during _bind() to an object of type echoES.
     Creates an returns a smart proxy object. */
  public org.omg.CORBA.Object New(org.omg.CORBA.portable.Delegate d) {

    if (false == createProxy)
         return null;

    System.out.println("Creating a Smart proxy...");
    createProxy = false; // We need only a single instance of SmartES!

    org.omg.CORBA.portable.ObjectImpl new_ref = null;
    try {
      new_ref = new SmartES ();
      new_ref._set_delegate(d);
    }
    catch (org.omg.CORBA.SystemException ex) {
      return null;
    }
    return new_ref;
  }// end method
}// end class
```

A4 – Echo Service DII Application

```
// echoServiceDII.idl

interface echoService {
  string Service(in string Msg);
  long Status();
};

/************************************************************************/

package echoServiceDII;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class javaclient1 {
  public static void main(String args[]) {
    ORB.init(args,null);

    // read the command line parameter
    CCommandLine cln = new CCommandLine(args);

    // start the client application
    CEchoClient clt = new CEchoClient(cln.GetProtocol(),cln.GetHost());
    clt.Run();
  }// end main
}// end class

/************************************************************************/

package echoServiceDII;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.Request;
import org.omg.CORBA.SystemException;
import IE.Iona.OrbixWeb._OrbixWeb;

public class CEchoClient {
  private org.omg.CORBA.Object refPtr = null;

  public CEchoClient(String Protocol, String Host) {
    // get the interface reference
    if(Protocol.equals("IIOP"))
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(true));
    else
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(false));

    try {
      refPtr =
      _OrbixWeb.ORB(ORB.init()).string_to_object(Host,null,"echoServer",null,"IR","echoService");
    } catch (SystemException ex) {
        System.out.println("Exception during destringify");
        System.out.println(ex.toString());
        return;
    }
    System.out.println("Found the echo factory.");
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void Run() {
    Request r = refPtr._request("");

    for (int i=0; i<5; i++) {
      DoStatus(r);
      DoService(r);
      DoStatus(r);
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Get the status of the remote echo object */
  public void DoStatus(Request r) {
```

```
    int     ServiceStatus;

    // Make a request
    try {
       _OrbixWeb.Request(r).reset("Status");
    } catch (SystemException ex) {
        System.out.println("Unexpected exception constructing a Request.");
        System.out.println(ex.toString());
        return;
    }

    try {
        r.set_return_type(IE.Iona.OrbixWeb._CORBA._tc_long);
    } catch (SystemException ex) {
        System.out.println("Unexpected exception setting return type.");
        System.out.println(ex.toString());
        return;
    }

    try {
        r.invoke();
    } catch (SystemException ex) {
        System.out.println("Unexpected exception invoking 'Status'.");
        System.out.println(ex.toString());
        return;
    }

    try {
        ServiceStatus = r.return_value().extract_long();
    } catch (SystemException ex) {
        System.out.println("Unexpected exception extracting status.");
        System.out.println(ex.toString());
        return;
    }
    System.out.println("Service status counter = " + ServiceStatus);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Call the echo service */
  public void DoService(Request r) {
    String Msg    = new String("This is the echo message... ");
    String Answer = null;

    // Make a request
    try {
       _OrbixWeb.Request(r).reset("Service");
    } catch (SystemException ex) {
        System.out.println("Unexpected exception constructing a Request.");
        System.out.println(ex.toString());
        return;
    }
    try {
        r.add_in_arg().insert_string(Msg);
        r.set_return_type(IE.Iona.OrbixWeb._CORBA._tc_string);
    } catch (SystemException ex) {
        System.out.println("Unexpected exception setting return type.");
        System.out.println(ex.toString());
        return;
    }

    try {
        r.invoke();
    } catch (SystemException ex) {
        System.out.println("Unexpected exception invoking 'Service'.");
        System.out.println(ex.toString());
        return;
    }

    try {
        Answer = new String(r.return_value().extract_string());
    } catch (SystemException ex) {
        System.out.println("Unexpected exception extracting service.");
        System.out.println(ex.toString());
        return;
    }
    System.out.println("Service answer : " + Answer);
  }// end method
}// end class
```

A5 - Echo Service Trading Application

```
// echoServiceTrader.idl

interface echoService_if {
  string Service(in string Msg);
  long Status();
};

/*************************************************************************/

package echoServiceTrader;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import CosTrading.*;
import org.omg.CORBA.*;

public class javaserver1 {

   public static void main(String args[]) {

      // ref to the trader
      String TRef = new
String("IOR:000000000000002249444c3a6f6d672e6f72672f436f7354726164696e672f4c6f6f6b75703a312e30000
000000000100000000000000055000100000000013776f6f64776172642e63732e7463642e69650000074e00000000000
313a5c776f6f64776172642e63732e7463642e69653a5444523a313a3a49523a436f7354726164696e675f4c6f6f6b757
000");

      // get initial reference of the trader and to the register interface
      ORB                 Orb        = null;
      org.omg.CORBA.Object InitTraRef = null;
      Lookup              TdrRef     = null;
      Register            RegRef     = null;
      Property            Props[]    = new Property[2];
      Any                 a          = null;
      Any                 b          = null;

      try {
         Orb                     = ORB.init();
         InitTraRef              = Orb.string_to_object(TRef);
         TdrRef                  = LookupHelper.narrow(InitTraRef);
         RegRef                  = TdrRef.register_if();
         a                       = Orb.create_any ();
         b                       = Orb.create_any ();
         echoService_if echoImpl = new _tie_echoService_if(new EchoServiceImplementation(100));

         // set the properties
         a.insert_string("Test1");
         b.insert_long(5);
         Props[0] = new Property("name", a);
         Props[1] = new Property("number", b);

         // export the service
         String offerId = RegRef.export(echoImpl,"IDL:echo:1.0",Props);
         System.out.println("Placed an offer in the trader and got offer_id ="
            + offerId);

         _CORBA.Orbix.impl_is_ready("echoServer");

         RegRef.withdraw(offerId);
         Orb.disconnect(echoImpl);

      }
      catch (Exception ex) {
         System.out.println("Exception during export service");
         ex.printStackTrace();
         System.exit(1);
      }

      System.out.println ("Server exiting....");
   }// end main

}// end class
```

```
/************************************************************************/

class EchoServiceImplementation implements _echoService_ifOperations {
  private int mStatus = 0;

  public EchoServiceImplementation(int Status) {
     mStatus = Status;
  }// end constructor

  // echo service operation
  public String Service(String Msg) {
     System.out.println("Server : Received message = " + Msg);
     String Answer = new String(Msg+"This is the echo answer");
     return Answer;
  }// end methode

  // status echo operation
  public int Status() {
     ++mStatus;
     return mStatus;
  }// end method
}// end class

/************************************************************************/

package echoServiceTrader;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;
import CosTrading.*;

public class javaclient2 {
   public static void main(String args[]) {
      // ref to the trader
      String TRef = new
String("IOR:000000000000002249444c3a6f6d672e6f72672f436f7354726164696e672f4c6f6f6b75703a312e30000
0000000001000000000000005500010000000000013776f6f64776172642e63732e7463642e69650000074e0000000000
313a5c776f6f64776172642e63732e7463642e69653a5444523a313a3a49523a436f7354726164696e675f4c6f6f6b757
000");

      // get initial reference of the trader
      ORB                  Orb        = null;
      org.omg.CORBA.Object InitTraRef = null;
      Lookup               TdrRef     = null;

      try {
         Orb        = ORB.init();
         InitTraRef = Orb.string_to_object(TRef);
         TdrRef     = LookupHelper.narrow(InitTraRef);
      }
      catch (Exception ex) {
         System.out.println("Exception during get initial trader reference");
         ex.printStackTrace();
         System.exit(1);
      }

      // start the client application
      CEchoClient2 clt = new CEchoClient2(TdrRef);
      clt.Run();
   }// end main

}// end class

/************************************************************************/

package echoServiceTrader;

import org.omg.CORBA.*;
import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;
import IE.Iona.OrbixWeb._OrbixWeb;
import CosTrading.*;
import CosTrading.Lookup.*;
import CosTrading.LookupPackage.*;

public class CEchoClient2 {
   static final String mEchoSTName = new String("IDL:echo:1.0");
   static final String mEchoIdName = new String("IDL:echo_if:1.0");

   private static Lookup  TdrLookupPtr = null;

   public CEchoClient2(Lookup TdrLookupPtr) {
```

```
      this.TdrLookupPtr = TdrLookupPtr;
   }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

   public void Run() {
      echoService_if TheService = QueryOffer();
      DoStatus(TheService);
      DoService(TheService);
      DoStatus(TheService);
   }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

public echoService_if QueryOffer() {
   // query
   System.out.println("--------------------------------");
   System.out.println("Query echo offer...");

   echoService_if      SerPtr              = null;
   Policy              default_policies[]  = new Policy[0];
   SpecifiedProps      desired_properties  = new SpecifiedProps();
                       desired_properties.___discriminator = HowManyProps.all;
   int                 how_many_offers     = 2 ;
   OfferSeqHolder      offers              = new OfferSeqHolder();
   OfferIteratorHolder offer_itr           = new OfferIteratorHolder();
   PolicyNameSeqHolder limits_applied      = new PolicyNameSeqHolder();

   try {
      TdrLookupPtr.query(mEchoSTName,"number > 0","random",
         default_policies,desired_properties,how_many_offers,offers,
         offer_itr,limits_applied);
   }
   catch (UserException ex) {
      System.out.println("User exception during query echo offer");
      ex.printStackTrace();
      System.exit(1);
   }
   catch (Exception ex) {
      System.out.println("Exception during query echo offer");
      ex.printStackTrace();
      System.exit(1);
   }

   if (offers.value.length > 0) {
      System.out.println("Offer echo found, size = " + offers.value.length);
      Offer offer = offers.value[0];
      System.out.println("Ref is " + offer.reference);

      // get ref to the echo service
      try {
         SerPtr = echoService_ifHelper.narrow(offer.reference);
      }
      catch (Exception ex) {
         System.out.println("Exception during get echo service reference");
         ex.printStackTrace();
         System.exit(1);
      }
   }
   else {
      System.out.println("No echo offer available!");
   }

   System.out.println("Offer echo queried!");
   return SerPtr;
}// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Get the status of the remote echo object */
  public void DoStatus(echoService_if s) {
    int ServiceStatus = 0;

    try {
       ServiceStatus = s.Status();
    }
    catch (org.omg.CORBA.SystemException ex) {
       System.out.println("Exception during Status");
       ex.printStackTrace();
       return;
    }
```

```
    System.out.println("Client : The service status counter has the value " + ServiceStatus);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  /* Call the echo service */
  public void DoService(echoService_if s) {
    String Msg    = new String("This is the echo message... ");
    String Answer = null;

    try {
      Answer = new String(s.Service(Msg));
    } catch (org.omg.CORBA.SystemException ex) {
      System.out.println("FAIL\tException during service");
      System.out.println(ex.toString());
      System.out.println("Minor=" + ex.minor);
      System.exit(1);
    }

    System.out.println("Client : The service answer is : " + Answer);
  }// end method
}// end class
```

12. APPENDIX B – SOURCE CODE FOR PROTOTYPE IMPLEMENTATION

This appendix contains the source code for the implementation of the fault tolerance prototype described in Chapter 5.

B1 – Package ft_Exceptions

```
/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : ConfigurationNotAvailable.java                                 */
/* Abstract : This exception package was developed as a part of my M.Sc.     */
/*            thesis at TCD.                                                 */
/*****************************************************************************/
/* Revision : 1.0                                                            */
/* Modified :                                                                */
/*****************************************************************************/

package ft_Exceptions;

public class ConfigurationNotAvailable extends Exception {
  public ConfigurationNotAvailable() { super(); }
}// end class

/*****************************************************************************/

public class TraderIORNotAvailable extends Exception {
  public TraderIORNotAvailable() { super(); }
}// end class

/*****************************************************************************/

public class RegisterOfferWithTraderFailed extends Exception {
  public RegisterOfferWithTraderFailed() { super(); }
}// end class

/*****************************************************************************/

public class ResolveOfferWithTraderFailed extends Exception {
  public ResolveOfferWithTraderFailed() { super(); }
}// end class
```

B2 – Package ft_Components

```
/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CInitialTraderRef.java                                         */
/* Abstract : This is basic implementation of a fault tolerant client server  */
/*            application using the OMG trading service.                     */
/*            This program was developed as a part of my M.Sc. thesis at     */
/*            TCD.                                                           */
/*            A client connects to a service. When the connected service gets */
/*            killed, the client smart proxy automatically reconnects to the  */
/*            backup service. As soon as the master service is back on line,  */
/*            the client smart proxy switches back to the master service.    */
/*            This class reads the trader's IOR from a file.                 */
/*****************************************************************************/
/* Revision : 1.0                                                            */
/* Modified :                                                                */
/*****************************************************************************/

package ft_Components;

import java.io.*;
import ft_Exceptions.*;

public class CInitialTraderRef {

  // instances
  private org.omg.CORBA.ORB mOrb         = null;
  private String            mTraderIOR   = null;
  private CDebug            mDebug       = null;// debug level

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CInitialTraderRef(org.omg.CORBA.ORB Orb, String CfgFileName) throws
ConfigurationNotAvailable, TraderIORNotAvailable {

    String          TraderIORFileName = null;
    DataInputStream TraderIORFile     = null;

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

    // class instances
    mOrb = Orb;

    // reads the trader IOR file name and location from the configuration file
    // throws exception
    CConfiguration Conf = new CConfiguration(CfgFileName);
    TraderIORFileName = Conf.GetTraderIORFileName();

    // open trader ior file
    try {
      TraderIORFile = new DataInputStream(new FileInputStream(TraderIORFileName));
    }
    catch (Exception e) {
      if (mDebug.mLevel>=mDebug.mHigh) {
        System.out.println("CInitialTraderRef - Cannot open Trader IOR file!");
      }
      throw new TraderIORNotAvailable();
    }

    // read the trader IOR from file
    try {
      mTraderIOR = new String(TraderIORFile.readLine());
    }
    catch (Exception e) {
      if (mDebug.mLevel>=mDebug.mHigh) {
        System.out.println("CInitialTraderRef - Cannot read Trader IOR file!");
      }
      throw new TraderIORNotAvailable();
    }

  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
  public org.omg.CORBA.Object GetTraderObj() {
    return mOrb.string_to_object(mTraderIOR);
  }// end method
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CTraderIF.java                                             */
/* Abstract : This class provides an application interface to the trader.*/
/*            It supports all client-trader and server-trader interfaces.*/
/*************************************************************************/

package ft_Components;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import CosTrading.*;
import CosTrading.Lookup.*;
import CosTrading.LookupPackage.*;
import CosTradingDynamic.*;
import ft_Exceptions.*;

public class CTraderIF {

  // trader properties
  private final String SerTpName = new String("ServiceTypeName");
  private final String SerName   = new String("ServerName");
  private final String SceName   = new String("ServiceName");
  private final String MList     = new String("MasterList");
  private final String PBList    = new String("PrimaryBackupList");
  private final String SBList    = new String("SecondaryBackupList");
  private final String Valid     = new String("OfferIsValid");
  private final String SerUtil   = new String("ServerUtilization");
  private final String NumUser   = new String("NumOfUsersOnServer");

  // trader interface parameters
  private String            mServiceType  = null;// service type name
  private String            mServiceOffer = null;// service offer name
  // trader interface references
  private Lookup            mLookupRef    = null;// trader interface
  private Register          mRegisterRef  = null;// trader interface
  private ImportAttributes  mImpAttRef    = null;// trader interface
  private SupportAttributes mSupAttRef    = null;// trader interface
  // class globals
  private org.omg.CORBA.ORB mOrb          = null;
  private CDebug            mDebug        = null;// debug level

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CTraderIF(org.omg.CORBA.ORB Orb, String ServiceTypeName, String ServiceOfferName, String
CfgFileName) throws ConfigurationNotAvailable, TraderIORNotAvailable {

    // initial trader interface reference
    org.omg.CORBA.Object InitTraRef = null;

    mOrb          = Orb;
    mServiceType  = new String(ServiceTypeName);
    mServiceOffer = new String(ServiceOfferName);

    // set debug level
    mDebug = new CDebug(mDebug.mLow);

    // get initial trader reference, throws exception
    CInitialTraderRef ITRef = new CInitialTraderRef(mOrb,CfgFileName);
    InitTraRef              = ITRef.GetTraderObj();
    // narrow initial trader reference into trader interface references
    mLookupRef              = LookupHelper.narrow(InitTraRef);
    mRegisterRef            = mLookupRef.register_if();
    mImpAttRef              = ImportAttributesHelper.narrow(InitTraRef);
    mSupAttRef              = SupportAttributesHelper.narrow(InitTraRef);
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // exports a service offer including all mandatory properties to the trader,
  // load balancing properties (dynamic) are set to default values
  public void Register(String ServerName, String ServiceName, String MasterList,
                       String PBackupList, String SBackupList,
```

```
                    boolean OfferIsValid, DynamicPropEval DP_SerUtil,
                    DynamicPropEval DP_NumUser, org.omg.CORBA.Object ObjRef) throws
RegisterOfferWithTraderFailed {
    // service offer properties
    Property      Props[]  = new Property[9];
    Any           na       = mOrb.create_any();// service type name
    Any           sr       = mOrb.create_any();// server name
    Any           sc       = mOrb.create_any();// service name
    Any           ml       = mOrb.create_any();// master list
    Any           bl1      = mOrb.create_any();// primary backup list
    Any           bl2      = mOrb.create_any();// secondary backup list
    Any           va       = mOrb.create_any();// valid offer
    Any           su       = mOrb.create_any();// dyn prop, no used yet
    Any           nu       = mOrb.create_any();// dyn prop, no used yet

    // set service offer properties
    na.insert_string(mServiceType);
    sr.insert_string(ServerName);
    sc.insert_string(ServiceName);
    ml.insert_string(MasterList);
    bl1.insert_string(PBackupList);
    bl2.insert_string(SBackupList);
    va.insert_boolean(OfferIsValid);
    su.insert_long(10);
    nu.insert_long(1);
    // set dynamic service offer properties, not used yet
    DynamicProp DProp_SU = new DynamicProp(DP_SerUtil,_CORBA._tc_long,na);
    DynamicProp DProp_NU = new DynamicProp(DP_NumUser,_CORBA._tc_long,na);
    //DynamicPropHelper.insert(su,DProp_SU);
    //DynamicPropHelper.insert(nu,DProp_NU);
    // set service offer property array
    Props[0] = new Property(SerTpName,na);
    Props[1] = new Property(SerName,sr);
    Props[2] = new Property(SceName,sc);
    Props[3] = new Property(MList,ml);
    Props[4] = new Property(PBList,bl1);
    Props[5] = new Property(SBList,bl2);
    Props[6] = new Property(Valid,va);
    Props[7] = new Property(SerUtil,su);
    Props[8] = new Property(NumUser,nu);

    // remove old service offer
    this.UnRegister(ServerName,ServiceName);

    // export the service offer
    try {
      String OfferId = mRegisterRef.export(ObjRef,mServiceOffer,Props);
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CTraderIF - placed an offer in the trader, id = " + OfferId);
        System.out.println("CTraderIF - reference is " + ObjRef);
     }
    }
    catch (Exception ex) {
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CTraderIF - register service offer failed.");
        System.out.println("CTraderIF - "+ex.toString());
      }
      throw new RegisterOfferWithTraderFailed();
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // withdraws a service offer from the trader
  public void UnRegister(String ServerName, String ServiceName) {

    // build the constraint string, 'search string' in trader property
    String Constraint = new String("'"+mServiceType+"' ~ "+SerTpName+
                            " and '"+ServerName+"' ~ "+SerName+
                            " and '"+ServiceName+"' ~ "+SceName);

    // withdraw the service offer
    try {
      mRegisterRef.withdraw_using_constraint(mServiceOffer,Constraint);
    }
    catch (Exception ex) {
      // tried to remove a non-existing offer
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CTraderIF - unregister service offer failed.");
        System.out.println("CTraderIF - "+ex.toString());
      }
    }
```

```
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // returns an service object reference that requires to be narrowed
  // into the correct type, flag is passed by reference
  public org.omg.CORBA.Object Resolve(int Which, String ClientGroupNumber, CInOutBoolean
OfferValidFlag) throws ResolveOfferWithTraderFailed {

    // build the contraint string
    String              Constraint = null;
    String              List       = null;
    CObjRefListConstants ObjList    = new CObjRefListConstants();

    // select the service list
    switch (Which) {
      case ObjList.mMaster:
             List = new String("MasterList");
             break;
      case ObjList.mPBackup:
             List = new String("PrimaryBackupList");
             break;
      case ObjList.mSBackup:
             List = new String("SecondaryBackupList");
             break;
      default:
             OfferValidFlag.SetValue(false);
             return null;
    }// end switch

    // build the constraint string, 'search string' in trader property
    Constraint = new String("'"+mServiceType+"' ~ "+SerTpName+
                    " and '"+ClientGroupNumber+"' ~ "+List);

    // resolve service offer, throws exception
    return DoResolve(Constraint, OfferValidFlag);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // returns an service object reference that requires to be narrowed
  // into the correct type
  public org.omg.CORBA.Object Resolve() throws ResolveOfferWithTraderFailed {

    // build the constraint string, 'search string' in trader property
    String Constraint = new String("'"+mServiceType+"' ~ "+SerTpName);

    // resolve service offer, throws exception
    CInOutBoolean Dummy = new CInOutBoolean();
    return DoResolve(Constraint, Dummy);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // returns an service object reference that requires to be narrowed
  // into the correct type
  private org.omg.CORBA.Object DoResolve(String Constraint, CInOutBoolean OfferValidFlag) throws
ResolveOfferWithTraderFailed {

    // service offer reference
    Offer               ObjRef           = null;
    // use the trader's default policy
    Policy              DefaultPolicies[] = new Policy[0];
    // specify all properties to be returned,
    // may be limited for performance reasons
    SpecifiedProps      DesiredProperties = new SpecifiedProps();
                        DesiredProperties.___discriminator = HowManyProps.all;
    // max number of service offers to be returned
    int                 HowManyOffers     = 2;
    // list of the returned service offers
    OfferSeqHolder      Offers            = new OfferSeqHolder();
    // offer iterator, not used
    OfferIteratorHolder OfferItr          = new OfferIteratorHolder();
    // policy sequence, not used
    PolicyNameSeqHolder LimitsApplied     = new PolicyNameSeqHolder();

    // display the trader's attributes
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CTraderIF - DefRetCard = "+mImpAttRef.def_return_card());
      System.out.println("CTraderIF - MaxRetCard = "+mImpAttRef.max_return_card());
      System.out.println("CTraderIF - DefMatCard = "+mImpAttRef.def_match_card());
      System.out.println("CTraderIF - MaxMatCard = "+mImpAttRef.max_match_card());
```

```
      System.out.println("CTraderIF - DefSerCard = "+mImpAttRef.def_search_card());
      System.out.println("CTraderIF - MaxSerCard = "+mImpAttRef.max_search_card());
      System.out.println("CTraderIF - ModifiableProps =
"+mSupAttRef.supports_modifiable_properties());
      System.out.println("CTraderIF - DynamicProps = "+mSupAttRef.supports_dynamic_properties());
    }// end if

    // query the trader for service offer
    try {
      mLookupRef.query(mServiceOffer, Constraint, "random", DefaultPolicies,
        DesiredProperties, HowManyOffers, Offers, OfferItr, LimitsApplied);
    }
    catch (Exception ex) {
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CTraderIF - resolve service offer failed.");
        System.out.println("CTraderIF - "+ex.toString());
      }
      throw new ResolveOfferWithTraderFailed();
    }

    // display the trader's attributes
    if ((mDebug.mLevel>=mDebug.mLow) && (Offers.value.length > 0)) {
      System.out.println("================================================");
      System.out.println("CTraderIF - service offer found, size = " + Offers.value.length);
      for (int i=0; i<Offers.value.length; i++) {
        ObjRef = Offers.value[i];
        System.out.println("Ref is " + ObjRef.reference);
        System.out.println("Property size is : " + ObjRef.properties.length);
        for (int k=0; k<ObjRef.properties.length; k++) {
          System.out.print(ObjRef.properties[k].name+" = ");
          // select the property value type
          if (ObjRef.properties[k].name.equals(Valid)) {
            System.out.println(ObjRef.properties[k].value.extract_boolean());
          }
          else if (ObjRef.properties[k].name.equals(SerUtil)) {
            System.out.println(ObjRef.properties[k].value.extract_long()+"%");
          }
          else if (ObjRef.properties[k].name.equals(NumUser)) {
            System.out.println(ObjRef.properties[k].value.extract_long());
          }
          else {
            System.out.println(ObjRef.properties[k].value.extract_string());
          }
        }// end for
        System.out.println("------------------------------------------------");
      }// end for
    }// end if

    // pre-init the service offer is valid flag
    OfferValidFlag.SetValue(false);

    // select the first service offer
    if (Offers.value.length > 0) {
      ObjRef = Offers.value[0];
    }
    else {
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CTraderIF - no service offer found.");
      }
      return null;
    }

    // extract the offer is valid value
    for (int k=0; k<ObjRef.properties.length; k++) {
      if (ObjRef.properties[k].name.equals(Valid)) {
        OfferValidFlag.SetValue(ObjRef.properties[k].value.extract_boolean());
      }
    }// end for

    return ObjRef.reference;
  }// end method
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CObjRefListConstants.java                                  */
/* Abstract : This class holds the constant parameters of the service object  */
/*            list.                                                      */
/*************************************************************************/

package ft_Components;
```

```
public class CObjRefListConstants {

  // constants
  public final int mNumEntry = 3;
  public final int mMaster   = 0;
  public final int mPBackup  = 1;
  public final int mSBackup  = 2;
  public final int mInvalid  = 3;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CObjRefListConstants() { } // end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String toString(int Which) {
    String Str;

    switch (Which) {
      case mMaster:
             Str = new String("Master");
             break;
      case mPBackup:
             Str = new String("PrimaryBackup");
             break;
      case mSBackup:
             Str = new String("SecondaryBackup");
             break;
      default:
             Str = new String(">Unknown<");
             break;
    }// end switch
    return Str;
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CConfiguration.java                                            */
/* Abstract : This class reads the client configuration from a text file.    */
/*****************************************************************************/

package ft_Components;

import java.io.*;
import ft_Exceptions.*;

public class CConfiguration {

  // constant
  private final String mClientGroupId  = new String("ClientGroupId");
  private final String mTdrIORFileName = new String("TraderIORFileName");
  private final String mTdrPollRate    = new String("TraderPollRate");
  private final char   mDelim          = '"';

  private String mConfigFileName = null;
  private CDebug mDebug          = null;// debug level

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CConfiguration (String ConfigFileName) {

    // set debug level
    mDebug = new CDebug(mDebug.mLow);

    // set class variables
    mConfigFileName = new String(ConfigFileName);

  }//end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // reads configuration parameter from file,
  // throws exception if value not available
  public String GetClientGroupId() throws ConfigurationNotAvailable {

    // instances
    DataInputStream File  = null;
    String          Line  = null;
    String          Value = null;
```

```
    // open configuration file
    File = OpenFile();
    if (File == null) {throw new ConfigurationNotAvailable();}

    // parse file for line including Keyword
    Line = ParseFileForLine(File, mClientGroupId);
    if (Line == null) {throw new ConfigurationNotAvailable();}

    // extract int value from line
    Value = new String(ExtractValueStringFromLine(Line));
    if (Value == null) {throw new ConfigurationNotAvailable();}

    if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CConfiguration - Client Group Id read : "+Value);
    }
    return Value;

  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // reads configuration parameter from file,
  // throws exception if value not available
  public String GetTraderIORFileName() throws ConfigurationNotAvailable {

    // instances
    DataInputStream File  = null;
    String          Line  = null;
    String          Value = null;

    // open configuration file
    File = OpenFile();
    if (File == null) {throw new ConfigurationNotAvailable();}

    // parse file for line including Keyword
    Line = ParseFileForLine(File, mTdrIORFileName);
    if (Line == null) {throw new ConfigurationNotAvailable();}

    // extract int value from line
    Value = new String(ExtractValueStringFromLine(Line));
    if (Value == null) {throw new ConfigurationNotAvailable();}

    if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CConfiguration - Trader IOR Filename read : "+Value);
    }
    return Value;

  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // reads configuration parameter from file,
  // returns default if value not available
  public int GetTraderPollRate() {

    // constants
    int DefaultDelay = 60;// minutes

    // instances
    DataInputStream File  = null;
    String          Line  = null;
    int             Value;

    // open configuration file
    File = OpenFile();
    if (File == null) {return DefaultDelay;}

    // parse file for line including Keyword
    Line = ParseFileForLine(File, mTdrPollRate);
    if (Line == null) {return DefaultDelay;}

    // extract int value from line
    Value = ParseLineForPosIntValue(Line);
    if (Value < 0) {return DefaultDelay;}

    if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CConfiguration - Trader Poll Rate read : "+Value);
    }
    return Value;

  }// end method
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private DataInputStream OpenFile() {

    // open configuration file
    DataInputStream File = null;
    try {
      File = new DataInputStream(new FileInputStream(mConfigFileName));
    }
    catch (Exception e) {
      if (mDebug.mLevel>=mDebug.mHigh) {
        System.out.println("CConfiguration - cannot open file "+mConfigFileName);
      }
      File = null;
    }
    return File;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private String ParseFileForLine(DataInputStream File, String KeyWord) {

    // instances
    String Line = null;

    // parse file
    do {
      // read next line
      try {
        Line = File.readLine();
        if (mDebug.mLevel>=mDebug.mHigh) {
          System.out.println("CConfiguration - Line : "+Line);
        }
        // search Keyword
        if (Line.regionMatches(true,0,KeyWord,0,KeyWord.length())) {
          // found KeyWord
          return Line;
        }
      }
      catch (Exception e) {
        if (mDebug.mLevel>=mDebug.mHigh) {
          System.out.println("CConfiguration - cannot read file.");
        }
        return null;
      }
    } while (!Line.equals(null));

    // EOF reached, KeyWord not found
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CConfiguration - parameter not found.");
    }
    return null;

 }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private int ParseLineForPosIntValue(String Line) {

    // instances
    String ValueStr = null;
    int    Value    = -1;

    // extract value string from line
    ValueStr = new String(ExtractValueStringFromLine(Line));

    // convert value string into int
    try {
      Value = Integer.parseInt(ValueStr);
    }
    catch (Exception e) {
      if (mDebug.mLevel>=mDebug.mHigh) {
        System.out.println("CConfiguration - invalid parameter found.");
      }
      Value = -1;
    }

    if ((Value >= 0) && (mDebug.mLevel>=mDebug.mHigh)) {
      System.out.println("CConfiguration - parameter found = "+Value);
    }
    return Value;
```

```
 }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private String ExtractValueStringFromLine(String Line) {

    // instances
    String ValueStr = null;

    // extract value string from line
    ValueStr = new
String(Line.substring(Line.indexOf((int)mDelim)+1,Line.lastIndexOf((int)mDelim)));

    return ValueStr;

 }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CDebug.java                                                    */
/* Abstract : This class provides a simple debug level holder.               */
/*****************************************************************************/

package ft_Components;

public class CDebug {

  // constants, what debug info will be displayed
  public final int mNone = 0;// no debug info
  public final int mLow  = 1;// important debug info only
  public final int mHigh = 2;// all debug info

  // selected debug level
  public int mLevel;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CDebug(int Level) {
    mLevel = Level;
  } // end constructor
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CInOutBoolean.java                                             */
/* Abstract : This class holds a boolean that will be passed by reference.   */
/*****************************************************************************/

package ft_Components;

public class CInOutBoolean {

  private boolean[] mBoolean = new boolean[1];//use array to pass by ref

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CInOutBoolean() {
    mBoolean[0] = false;
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CInOutBoolean(boolean Value) {
    mBoolean[0] = Value;
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void SetValue(boolean Value) {
    mBoolean[0] = Value;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public boolean GetValue() {
    return mBoolean[0];
  }// end method
}// end class
```

B3 – Package ft_echoService

```
/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : ft_echoService.idl                                         */
/* Abstract : This is basic implementation of a fault tolerant client server  */
/*            application using the OMG trading service.                 */
/*            This program was developed as a part of my M.Sc. thesis at     */
/*            TCD.                                                       */
/*            A client connects to a service. When the connected service gets */
/*            killed, the client smart proxy automatically reconnects to the  */
/*            backup service. As soon as the master service is back on line,  */
/*            the client smart proxy switches back to the master service.    */
/*************************************************************************/
/* Revision : 1.0                                                        */
/* Modified :                                                            */
/*************************************************************************/

interface ft_echoService_if {

  // fault tolerance exception
  exception NoValidObjectReference {};

  // basic service implemenation that receives a string, appends its answer
  // string and returns the new string.
  string DoService(in string Msg);// ret. null in case of NoValidObjectReference
  string DoService_ft(in string Msg) raises(NoValidObjectReference);

  // returns the service status that is the number of DoService calls.
  long   GetStatus();// returns 0 in case of NoValidObjectReference
  long   GetStatus_ft() raises(NoValidObjectReference);

  // returns its in value, used by the smart proxy class only to ping
  // a service object. This method has to be appended to every ft interface.
  long   Ping(in long Test);// returns 0 in case of NoValidObjectReference
  long   Ping_ft(in long Test) raises(NoValidObjectReference);
};

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CechoServiceSmartProxyFactory.java                         */
/* Abstract : This class creates and delegates an instance of the smart proxy */
/*            class, so the smart proxy will be called instead of the service */
/*            stub.                                                      */
/*            The necessary global settings, such as Trader instance, service */
/*            interface name, etc. are handled by this class as well.    */
/*************************************************************************/

package ft_echoService;

import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import IE.Iona.OrbixWeb.Features.ProxyFactory;
import ft_Components.*;
import ft_Exceptions.*;

public class CechoServiceSmartProxyFactory extends ProxyFactory {

  // constants
  // echoService (ES) type name
  private final String mESType=new String("ft_echoService_if");
  // echoService (ES) trader offer type name
  private final String mESOffer=new String("IDL:ft_echoServiceType:2.1");

  // reference to the service
  private ft_echoService_if mEchoService = null;
  // reference to the trader interface
  private CTraderIF         mTdrIF       = null;
  // name of the configuartion file
  private String            mCfgFileName = null;
  private CDebug            mDebug       = null;// debug level
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // creates an instance of the service and does the smart proxy delegation
  public CechoServiceSmartProxyFactory(org.omg.CORBA.ORB Orb, String CfgFileName) throws
ConfigurationNotAvailable, TraderIORNotAvailable, ResolveOfferWithTraderFailed, SystemException {

    // call super class
    super (ft_echoService_ifHelper.id());

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

    org.omg.CORBA.Object           SerObjRef = null;
    org.omg.CORBA.Object           SPRef     = null;
    org.omg.CORBA.portable.Delegate DelRef    = null;

    mCfgFileName = new String(CfgFileName);

    // get an instance of the trader interface, throws exception
    mTdrIF = new CTraderIF(Orb, mESType, mESOffer, mCfgFileName);
    // resolve a reference of a service object, throws exception
    SerObjRef = mTdrIF.Resolve();

    // get a delegation reference and do the delegation of the smart proxy
    DelRef = ((org.omg.CORBA.portable.ObjectImpl)SerObjRef)._get_delegate();
    SPRef  = this.ESNew(DelRef);// throws exception

    // narrow the service object reference into the correct service type
    mEchoService = ft_echoService_ifHelper.narrow(SPRef);

  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // creates a smart proxy object, does the delegation and returns the ref
  private org.omg.CORBA.Object ESNew(org.omg.CORBA.portable.Delegate d) throws
ConfigurationNotAvailable, ResolveOfferWithTraderFailed, SystemException {

    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServiceSmartProxyFactory - create smart proxy.");
    }

    // do the delegation
    org.omg.CORBA.portable.ObjectImpl new_ref = null;
    // throws exception
    new_ref = new CechoServiceSmartProxy(mTdrIF, mCfgFileName);
    // throws exception
    new_ref._set_delegate(d);

    return new_ref;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // default method, never called
  public org.omg.CORBA.Object New(org.omg.CORBA.portable.Delegate d) {

    if (mDebug.mLevel>=mDebug.mLow) {
      System.out.println("CechoServiceSmartProxyFactory - default New called!");
    }
    return null;

  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public ft_echoService_if GetEchoServiceObjRef() {
    return mEchoService;
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CechoServiceSmartProxy.java                                    */
/* Abstract : This smart proxy class overrides the client stub.              */
/*****************************************************************************/

package ft_echoService;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
```

```
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import ft_Components.*;
import ft_Exceptions.*;
import ft_echoService.ft_echoService_ifPackage.*;// IDL exceptions

public class CechoServiceSmartProxy extends _ft_echoService_ifStub {

  private CechoServiceManager mES    = null;// service object(s)
  private CDebug              mDebug = null;// debug level

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceSmartProxy(CTraderIF TdrIF, String CfgFileName) throws
ConfigurationNotAvailable, ResolveOfferWithTraderFailed {

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

    // instanciate the list of object references, throws exception
    mES = new CechoServiceManager(TdrIF, CfgFileName);

  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                             */
/* fault tolerant implementation of the client smart proxy overrides the      */
/* client stub.                                                                */
/*                                                                             */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String DoService(String Msg) {

    String            Default = null;
    ft_echoService_if ObjRef  = null;

    // get the actual service obj ref, throws exception
    try {
      ObjRef = mES.GetActObj();
    }
    catch (NoValidObjectReference ex) {
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - no DoService available!");
      }
      return Default;
    }

    // perform the service
    try {
      String Result = new String(ObjRef.DoService(Msg));
      return Result;
    }
    catch (Exception ex) {
      // service failed
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - reconnecting DoService!");
      }

      // try to reconnect to another service
      mES.SetObjRefToInvalid(ObjRef);// select another service object reference
      return this.DoService(Msg);// try next service
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String DoService_ft(String Msg) throws NoValidObjectReference {

    ft_echoService_if ObjRef = null;

    // get the actual service obj ref, throws exception
    ObjRef = mES.GetActObj();

    // perform the service
    try {
      String Result = new String(ObjRef.DoService(Msg));
      return Result;
    }
    catch (Exception ex) {
      // service failed
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - reconnecting DoService!");
```

```
      }

      // try to reconnect to another service
      mES.SetObjRefToInvalid(ObjRef);// select another service object reference
      return this.DoService_ft(Msg);// try next service, throws exception
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public int GetStatus() {

    int              Default = 0;
    ft_echoService_if ObjRef  = null;

    // get the actual service obj ref, throws exception
    try {
      ObjRef = mES.GetActObj();
    }
    catch (NoValidObjectReference ex) {
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - no GetStatus available!");
      }
      return Default;
    }

    // perform the service
    try {
      int Result = ObjRef.GetStatus();
      return Result;
    }
    catch (Exception ex) {
      // service failed
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - reconnecting GetStatus!");
      }

      // try to reconnect to another service
      mES.SetObjRefToInvalid(ObjRef);// select another service object reference
      return this.GetStatus();// try next service
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public int GetStatus_ft() throws NoValidObjectReference {

    ft_echoService_if ObjRef = null;

    // get the actual service obj ref, throws exception
    ObjRef = mES.GetActObj();

    // perform the service
    try {
      int Result = ObjRef.GetStatus();
      return Result;
    }
    catch (Exception ex) {
      // service failed
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - reconnecting GetStatus!");
      }

      // try to reconnect to another service
      mES.SetObjRefToInvalid(ObjRef);// select another service object reference
      return this.GetStatus_ft();// try next service, throws exception
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // this service is used by the service manager only
  public int Ping(int Test) {

    int              Default = 0;
    ft_echoService_if ObjRef  = null;

    // get the actual service obj ref, throws exception
    try {
      ObjRef = mES.GetActObj();
```

```
    }
    catch (NoValidObjectReference ex) {
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - no Ping available!");
      }
      return Default;
    }

    // perform the service
    try {
      int Result = ObjRef.Ping(Test);
      return Result;
    }
    catch (Exception ex) {
      // service failed
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - reconnecting Ping!");
      }

      // try to reconnect to another service
      mES.SetObjRefToInvalid(ObjRef);// select another service object reference
      return this.Ping(Test);// try next service
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // this service is used by the service manager only
  public int Ping_ft(int Test) throws NoValidObjectReference {

    ft_echoService_if ObjRef = null;

    // get the actual service obj ref, throws exception
    ObjRef = mES.GetActObj();

    // perform the service
    try {
      int Result = ObjRef.Ping(Test);
      return Result;
    }
    catch (Exception ex) {
      // service failed
      if (mDebug.mLevel>=mDebug.mLow) {
        System.out.println("CechoServiceSmartProxy - reconnecting Ping!");
      }

      // try to reconnect to another service
      mES.SetObjRefToInvalid(ObjRef);// select another service object reference
      return this.Ping_ft(Test);// try next service, throws exception
    }
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CechoServiceManager.java                                       */
/* Abstract : This class inits and holds the service object reference list.  */
/*****************************************************************************/

package ft_echoService;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import ft_Components.*;
import ft_Exceptions.*;
import ft_echoService.ft_echoService_ifPackage.*;// IDL exceptions

public class CechoServiceManager {

  private CTraderIF              mTdrIF    = null;// ref to the trader InterFace
  private String                 mCfgName  = null;// configuration file name
  private String                 mClGrpId  = null;// unique client group id
  private CechoServiceObjRefList mObjRef   = null;// service obj ref list
  private CDebug                 mDebug    = null;// debug level

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceManager(CTraderIF TdrIF, String CfgFileName) throws
ConfigurationNotAvailable, ResolveOfferWithTraderFailed {
```

```
    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

    // set the obj ref to the trader interface
    mTdrIF   = TdrIF;
    mCfgName = new String(CfgFileName);

    // read client group id from configuration file
    CConfiguration Conf = new CConfiguration(mCfgName);
    // set the client group id, clients using the same loation are likely to
    // have the same id, throws exception
    mClGrpId = new String(Conf.GetClientGroupId());

    // instantiante a service object reference list
    mObjRef = new CechoServiceObjRefList(mTdrIF, mClGrpId, mCfgName);

    // get an obj ref to master and backup service objects, throws exception
    Resolve(mObjRef.mSBackup);
    Resolve(mObjRef.mPBackup);
    Resolve(mObjRef.mMaster);

  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // queries the trader for a service object reference
  private void Resolve(int Which) throws ResolveOfferWithTraderFailed {

    // valid obj ref found, used to pass by reference
    CInOutBoolean OfferValidFlag = new CInOutBoolean(false);

    // query the trader for a service offer, throws exception
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServiceManager - resolve "+mObjRef.toString(Which));
    }
    org.omg.CORBA.Object SerObjRef = mTdrIF.Resolve(Which, mClGrpId, OfferValidFlag);

    // narrow the obj ref into the service type and update the obj ref list
    if (SerObjRef != null) {
      mObjRef.Update(Which, ft_echoService_ifHelper.narrow(SerObjRef),
OfferValidFlag.GetValue());
    }
    else {
      mObjRef.Update(Which, null, false);
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // returns the actual object reference to be invoked on
  public ft_echoService_if GetActObj() throws NoValidObjectReference {
    return mObjRef.GetActObj();
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // sets the given object reference to invalid
  public void SetObjRefToInvalid (ft_echoService_if ObjRef) {
    mObjRef.Update(ObjRef,false);
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CechoServiceObjRefList.java                                    */
/* Abstract : This class holds the list of the available service objects.    */
/*            Threads will be started to ping invalid object references.     */
/*****************************************************************************/

package ft_echoService;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import ft_Components.*;
import ft_Exceptions.*;
import ft_echoService.ft_echoService_ifPackage.*;// IDL exceptions

public class CechoServiceObjRefList extends CObjRefListConstants {
```

```
  // list of all the service object references.
  // in case no object reference is available, the entry is set to null
  // in case an object reference is on line, the flag is true
  private ft_echoService_if[] mESList  = new ft_echoService_if[mNumEntry];
  private boolean[]           mESFlag  = new boolean[mNumEntry];// on line refs
  private int                 mESAct;// obj ref currently in use
  private CDebug              mDebug   = null;// debug level
  private CTraderIF           mTdrIF   = null;// ref to the trader InterFace
  private String              mCfgName = null;// configuration file name
  private String              mClGrpId = null;// unique client group id

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceObjRefList(CTraderIF TdrIF, String ClientGroupId, String CfgFileName) {

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

    // set the obj ref to the trader interface
    mTdrIF   = TdrIF;
    mCfgName = new String(CfgFileName);

    // set the client group id, clients using the same loation are likely to
    // have the same id
    mClGrpId = new String(ClientGroupId);

    // pre-init the object list
    for (int i=0; i<mNumEntry; i++) {
      mESList[i] = null;
      mESFlag[i] = false;
    }
    mESAct = mInvalid;

  } // end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // returns the actual object reference
  public synchronized ft_echoService_if GetActObj() throws NoValidObjectReference {
    // check valid ref available
    if (mESAct>=mInvalid) {
      // no valid service object reference available
      throw new NoValidObjectReference();
    }
    return mESList[mESAct];
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // inserts a new object reference into the list
  public synchronized void Update(int Which, ft_echoService_if ObjRef, boolean Flag) {
    DoUpdate(Which, ObjRef, Flag);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // updates an existing object reference in the list
  public synchronized void Update(ft_echoService_if ObjRef, boolean Flag) {
    // search obj ref list
    for (int i=0; i<mNumEntry; i++) {
      if (ObjRef == mESList[i]) {
         DoUpdate(i,ObjRef,Flag);
      }
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // updates the object reference list
  // starts a ping thread for invalid refereces
  // selects the best available object reference and sets it active
  private void DoUpdate(int Which, ft_echoService_if ObjRef, boolean Flag) {
    // update the object reference list
    mESList[Which] = ObjRef;
    mESFlag[Which] = Flag;
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServiceObjRefList - "+this.toString(Which)+" offer updated. Ref =
"+ObjRef+" Flag = "+Flag);
    }

    // start up a thread to get a new offer of an invalid object ref
```

```
    if ((mESFlag[Which] == false) && (mESList[Which] != null)) {
      StartThread(Which);
    }

    // set the best available obj ref active
    SelectBestObjRef();
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // selects the best available obj ref and sets it active
  private void SelectBestObjRef() {
    mESAct = mInvalid;
    for (int i=(mNumEntry-1); i>=0; i--) {
      if ((mESFlag[i]) && (mESList[i]!=null)) {
        mESAct = i;
      }
    }
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServiceObjRefList - "+this.toString(mESAct)+" offer selected.");
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // starts a thread that periodically queries the trader for offers and pings
  // the offer's object reference to figure out whether it is a valid offer
  private void StartThread(int Which) {

    // create and start ping thread
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServiceObjRefList - create thread to ping "+this.toString(Which));
    }
    new CechoServicePingThread(Which, mTdrIF, mClGrpId, mCfgName, this);

  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Abstract : This class holds the threads to ping invalid object references. */
/*****************************************************************************/

class CechoServicePingThread extends Thread {

  private int                   mWhich;// which type to be pinged for
  private CechoServiceObjRefList mBackCallObjRef;// backcall obj to update objs
  private CDebug                mDebug   = null;// debug level
  private int                   mDelay;// ping delay (frequence) in minutes
  private CTraderIF             mTdrIF   = null;// ref to the trader InterFace
  private String                mCfgName = null;// configuration file name
  private String                mClGrpId = null;// unique client group id
  private int                   mPingTries = 0;// number of failed ping tries

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServicePingThread (int Which, CTraderIF TdrIF, String ClientGroupId, String
CfgFileName, CechoServiceObjRefList BackCallObjRef) {

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

    // set thread variables
    mWhich          = Which;
    mBackCallObjRef = BackCallObjRef;
    mTdrIF          = TdrIF;
    mCfgName        = new String(CfgFileName);
    mClGrpId        = new String(ClientGroupId);

    // reads the ping rate from the configuration file or set default value
    CConfiguration Conf = new CConfiguration(mCfgName);
    mDelay = Conf.GetTraderPollRate();

    // start thread
    this.start();
  }//end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // queries the trader for service offers and pings them
  public void run() {
```

```
    boolean Found = false;// loop exit condition
    int     Delay = mDelay;// delay in minutes

    // start querying the trader and pinging the service object
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServicePingThread - start pinging
"+mBackCallObjRef.toString(mWhich));
    }
    do {
      // delay
      try { sleep(Delay*60*1000); }
      catch (Exception ex) { }

      if (mDebug.mLevel>=mDebug.mHigh) {
        System.out.println("CechoServicePingThread - query trader for
"+mBackCallObjRef.toString(mWhich)+" offer.");
      }

      // query the trader for a service offer and ping reference
      Found = ResolveAndPing(mWhich);

      if ((Found) && (mDebug.mLevel>=mDebug.mLow)) {
          System.out.println("CechoServicePingThread - found valid
"+mBackCallObjRef.toString(mWhich)+" offer.");
      }
    } while (!Found);

  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // queries the trader for a service object reference
  private boolean ResolveAndPing(int Which) {

    // number of unsuccessful tries to contact the trader before killing thread
    final int MaxNumPingTries = 10;

    // valid obj ref found, used to pass by reference
    CInOutBoolean OfferValidFlag = new CInOutBoolean(false);

    // service object references are exported as CORBA objects
    org.omg.CORBA.Object SerObjRef = null;
    ft_echoService_if    mES       = null;

    // query the trader
    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServicePingThread - Resolve "+mBackCallObjRef.toString(Which));
    }
    // pass flag array by reference
    try {
      SerObjRef = mTdrIF.Resolve(Which, mClGrpId, OfferValidFlag);
    }
    catch (ResolveOfferWithTraderFailed e) {
      // check whether this thread has to be killed
      mPingTries++;
      if (mPingTries>=MaxNumPingTries) {
        if (mDebug.mLevel>=mDebug.mHigh) {
          System.out.println("CechoServicePingThread - killing thread due to trader access
failure.");
        }
        // kill thread
        this.stop();
      }
      return false;
    }

    // narrow the obj ref into the service type and ping it
    if ((OfferValidFlag.GetValue()) && (SerObjRef != null)) {
      // narrow and ping object reference
      mES = ft_echoService_ifHelper.narrow(SerObjRef);
      try {
        if (mES.Ping(5)==5) {
          // valid obj ref found, update object reference list
          mBackCallObjRef.Update(mWhich, mES, true);
          return true;
        }
      }
      catch (Exception ex) { }
    }// end if
    return false;
  }// end method
```

```
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : ft_echoService1.cfg                                            */
/* Abstract : This text file contains service configuration parameters.      */
/*            Parameter line syntax :                                        */
/*             - non-case sensitive Keyword                                  */
/*             - "value"                                                     */
/*****************************************************************************/
/* Revision : 1.0                                                            */
/* Modified :                                                                */
/*****************************************************************************/

// parameter used by clients only
ClientGroupId     = "0001";
TraderPollRate    = "1";

// parameter used by clients and servers
TraderIORFileName = "/dd/msc/meierr/dis_orbix/Trader/trader_data/IOR/trader.ior";

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : ft_echoService2.cfg                                            */
/* Abstract : This text file contains service configuration parameters.      */
/*            Parameter line syntax :                                        */
/*             - non-case sensitive Keyword                                  */
/*             - "value"                                                     */
/*****************************************************************************/
/* Revision : 1.0                                                            */
/* Modified :                                                                */
/*****************************************************************************/

// parameter used by clients only
ClientGroupId     = "0002";
TraderPollRate    = "1";

// parameter used by clients and servers
TraderIORFileName = "/dd/msc/meierr/dis_orbix/Trader/trader_data/IOR/trader.ior";
```

B4 – Package ft_echoServiceApplication, Client

```
/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : javaclient1.java                                           */
/* Abstract : This is basic implementation of a fault tolerant client server  */
/*            application using the OMG trading service.                 */
/*            This program was developed as a part of my M.Sc. thesis at     */
/*            TCD.                                                       */
/*            A client connects to a service. When the connected service gets */
/*            killed, the client smart proxy automatically reconnects to the  */
/*            backup service. As soon as the master service is back on line,  */
/*            the client smart proxy switches back to the master service.    */
/*            This class starts up an client application.                */
/*************************************************************************/
/* Revision : 1.0                                                        */
/* Modified :                                                            */
/*************************************************************************/

package ft_echoServiceApplication;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class javaclient1 {

  public static void main(String args[]) {

    // name of the configuartion file
    final String CfgFileName = new
String("/dd/msc/meierr/dis_orbix/Web/appl/ft_echoService/ft_echoService1.cfg");

    ORB Orb = null;
    // init orb
    try {
      Orb = ORB.init();
    }
    catch (Exception ex) {
      System.out.println("Exception during ORB init!");
      ex.printStackTrace();
      System.exit(1);
    }

    // start the client application
    CechoServiceClient clt =
      new CechoServiceClient(Orb, CfgFileName);
    clt.Run();
  }// end main
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : javaclient2.java                                           */
/* Abstract : This class starts up an client application.                */
/*************************************************************************/

package ft_echoServiceApplication;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class javaclient2 {

  public static void main(String args[]) {

  // name of the configuartion file
  final String CfgFileName = new
String("/dd/msc/meierr/dis_orbix/Web/appl/ft_echoService/ft_echoService2.cfg");

    ORB Orb = null;
    // init orb
    try {
      Orb = ORB.init();
    }
```

```
    catch (Exception ex) {
      System.out.println("Exception during ORB init!");
      ex.printStackTrace();
      System.exit(1);
    }

    // start the client application
    CechoServiceClient clt =
      new CechoServiceClient(Orb, CfgFileName);
    clt.Run();
  }// end main
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CechoServiceClient.java                                    */
/* Abstract : This class implements an client application.               */
/*************************************************************************/

package ft_echoServiceApplication;

import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import IE.Iona.OrbixWeb._CORBA;
import IE.Iona.OrbixWeb._OrbixWeb;
import ft_echoService.*;
import ft_Exceptions.*;
import ft_echoService.ft_echoService_ifPackage.*;// IDL exceptions

public class CechoServiceClient {
  // constants
  private final int     mNumServer      = 3;
  private final int     mNumService     = 2;
  private final boolean mShowInvokeTime = false;
  private final int     mNumCalls       = 1;

  // class instances
  private CechoServiceSmartProxyFactory mSPF = null;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceClient(ORB Orb, String CfgFileName) {

    // instanciate the client smart proxy and the trader
    try {
      mSPF = new CechoServiceSmartProxyFactory(Orb, CfgFileName);
    }
    catch (ConfigurationNotAvailable ex) {
      System.out.println("Exception during instanciating proxy factory - no config.!");
      ex.printStackTrace();
      System.exit(1);
    }
    catch (TraderIORNotAvailable ex) {
      System.out.println("Exception during instanciating proxy factory - no trader IOR!");
      ex.printStackTrace();
      System.exit(1);
    }
    catch (ResolveOfferWithTraderFailed ex) {
      System.out.println("Exception during instanciating proxy factory - no trader!");
      ex.printStackTrace();
      System.exit(1);
    }
    catch (SystemException ex) {
      System.out.println("System exception during instanciating proxy factory!");
      ex.printStackTrace();
      System.exit(1);
    }
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void Run() {

    final int NumServiceCalls = 1000;

    // get a service object reference
    ft_echoService_if TheService = mSPF.GetEchoServiceObjRef();

    // use the service object refernce
    System.out.println("");
    System.out.println("*******************************");
```

```
    System.out.println("*   Connecting echoServer(s)   *");
    System.out.println("********************************");
    for (int i=0; i<NumServiceCalls; i++) {
      PerformDoService(TheService);
      PerformGetStatus(TheService);
      System.out.println("----------------------------------");
      // delay
      try { Thread.sleep(500); }
      catch (Exception ex) { }
    }// end for
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private void PerformGetStatus(ft_echoService_if mES) {

    // perform service invokation
    if (!mShowInvokeTime) {
      try {
        int ESStatus = mES.GetStatus_ft();
        System.out.println("CechoServiceClient - GetStatus = " + ESStatus);
      }
      catch (NoValidObjectReference ex) {
        System.out.println("CechoServiceClient - GetStatus not available!");
      }
    }// end if
    else {
      // display invokation time
      try {
        long Time1 = 0;
        long Time2 = 0;

        Time1 = System.currentTimeMillis();
        for (int i=0; i<mNumCalls; i++) {
          int ESStatus = mES.GetStatus_ft();
        }
        Time2 = System.currentTimeMillis();
        System.out.println("CechoServiceClient - duration of "+mNumCalls+" GetStatus invokations
= "+(Time2-Time1)+" Millis.");
      }
      catch (NoValidObjectReference ex) {
        System.out.println("CechoServiceClient - GetStatus not available!");
      }
    }// end else
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private void PerformDoService(ft_echoService_if mES) {

    final String Msg    = new String("Client msg");

    // perform service invokation
    if (!mShowInvokeTime) {
      try {
        String Answer = new String(mES.DoService_ft(Msg));
        System.out.println("CechoServiceClient - DoService = " + Answer);
      }
      catch (NoValidObjectReference ex) {
        System.out.println("CechoServiceClient - DoService not available!");
      }
    }// end if
    else {
      // display invokation time
      try {
        long Time1 = 0;
        long Time2 = 0;

        Time1 = System.currentTimeMillis();
        for (int i=0; i<mNumCalls; i++) {
          String Answer = new String(mES.DoService_ft(Msg));
        }
        Time2 = System.currentTimeMillis();
        System.out.println("CechoServiceClient - duration of "+mNumCalls+" DoService invokations
= "+(Time2-Time1)+" Millis.");
      }
      catch (NoValidObjectReference ex) {
        System.out.println("CechoServiceClient - DoService not available!");
      }
    }// end else
  }// end method
}// end class
```

B5 – Package ft_echoServiceApplication, Client Startup

```
/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : javaclientStartUp.java                                     */
/* Abstract : This is basic implementation of a fault tolerant client server  */
/*            application using the OMG trading service.                 */
/*            This program was developed as a part of my M.Sc. thesis at     */
/*            TCD.                                                       */
/*            A client connects to a service. When the connected service gets */
/*            killed, the client smart proxy automatically reconnects to the  */
/*            backup service. As soon as the master service is back on line,  */
/*            the client smart proxy switches back to the master service.    */
/*            This class starts up an client application.                */
/*************************************************************************/
/* Revision : 1.0                                                        */
/* Modified :                                                            */
/*************************************************************************/

package ft_echoServiceApplication;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class javaclientStartUp {

  public static void main(String args[]) {

    ORB Orb = null;

    // init orb
    try {
      Orb = ORB.init();
    }
    catch (Exception ex) {
      System.out.println("Exception during ORB init!");
      ex.printStackTrace();
      System.exit(1);
    }

    // read the command line parameter
    CCommandLine cln = new CCommandLine(args);

    // start the client application
    CechoServiceStartUp clt =
      new CechoServiceStartUp(cln.GetProtocol(), cln.GetHost(), Orb);
    clt.Run();

  }// end main
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CCommandLine.java                                          */
/* Abstract : This class evaluates command line parameters.              */
/*************************************************************************/

package ft_echoServiceApplication;

import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;

public class CCommandLine {

  private String[] mProperties = new String[2];

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CCommandLine(String Args[]) {
  // Command line syntax is as follows...
  // ExecutableName [<IIOP>] [<hostname>]
  // If the "IIOP" parameter is given then the client will use the
  // IIOP protocol when taking to the orbixdj and the server. The
  // hostname parameter specifies where to find the orbixdj.
```

```
    // default, use IIOP and the local orbixdj
    if (Args.length < 1) {
      mProperties[0] = new String(_CORBA.Orbix.myHost());
      mProperties[1] = new String("IIOP");
    }
    // get parameters
    else if (Args[0].equals("IIOP")) {
        mProperties[1] = new String("IIOP");
      if (Args.length < 2)
        mProperties[0] = new String(_CORBA.Orbix.myHost());
      else
        mProperties[0] = new String(Args[1]);// hostname
    }
    else {
      mProperties[0] = new String(Args[0]);// hostname
      mProperties[1] = new String("POOP");
    }
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String GetProtocol() {
    return mProperties[1];
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String GetHost() {
    return mProperties[0];
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CechoServiceStartUp.java                                       */
/* Abstract : This class starts up the servers for the very first time to    */
/*            make them exporting service offers to the trader.              */
/*****************************************************************************/

package ft_echoServiceApplication;

import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import IE.Iona.OrbixWeb._CORBA;
import IE.Iona.OrbixWeb._OrbixWeb;
import ft_echoService.*;

public class CechoServiceStartUp {

  // constants
  private final int     mNumServer      = 3;
  private final int     mNumService     = 2;
  private final boolean mShowInvokeTime = false;
  private final int     mNumCalls       = 1000;

  // server and service name
  private final String mServerName   = new String("echoServer");
  private final String mServiceName1 = new String("Echo Service One");
  private final String mServiceName2 = new String("Echo Service Two");

  // class instances
  private ft_echoService_if[][] mES = new ft_echoService_if[mNumServer][mNumService];

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceStartUp(String Protocol, String Host, ORB Orb) {

    // select Orbix protocol IIOP
    if(Protocol.equals("IIOP"))
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(true));
    else
      _CORBA.Orbix.setConfigItem ("IT_BIND_USING_IIOP", String.valueOf(false));

    // do binding
    try {
      for (int Server=0; Server<mNumServer; Server++) {
        mES[Server][0] =
          ft_echoService_ifHelper.bind((mServiceName1+" on
"+mServerName+(Server+1)+":echoServer"+(Server+1)), Host);
        mES[Server][1] =
```

```
          ft_echoService_ifHelper.bind((mServiceName2+" on
"+mServerName+(Server+1)+":echoServer"+(Server+1)), Host);
      }
    }
    catch (Exception ex) {
      System.out.println("CechoServiceStartUp - exception during bind");
      ex.printStackTrace();
      System.exit(1);
    }
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void Run() {

    // use the service object refernce, call each service once
    for (int Server=0; Server<mNumServer; Server++) {
      System.out.println("");
      System.out.println("************************************");
      System.out.println("*  Bind : Connecting echoServer"+(Server+1)+" :  *");
      System.out.println("************************************");
      for (int Service=0; Service<mNumService; Service++) {
        PerformDoService(mES[Server][Service]);
        PerformGetStatus(mES[Server][Service]);
      }
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private void PerformGetStatus(ft_echoService_if mES) {

    // perform service invokation
    if (!mShowInvokeTime) {
      int ESStatus = mES.GetStatus();
      System.out.println("CechoServiceStartUp - GetStatus = " + ESStatus);
    }
    else {
      // display invokation time
      long Time1 = 0;
      long Time2 = 0;

      Time1 = System.currentTimeMillis();
      for (int i=0; i<mNumCalls; i++) {
         int ESStatus = mES.GetStatus();
      }
      Time2 = System.currentTimeMillis();
      System.out.println("CechoServiceStartUp - duration of "+mNumCalls+" GetStatus invokations =
"+(Time2-Time1)+" Millis.");
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  private void PerformDoService(ft_echoService_if mES) {

    final String Msg    = new String("Client msg");

    // perform service invokation
    if (!mShowInvokeTime) {
      String Answer = new String(mES.DoService(Msg));
      System.out.println("CechoServiceStartUp - DoService = " + Answer);
    }
    else {
      // display invokation time
      long Time1 = 0;
      long Time2 = 0;

      Time1 = System.currentTimeMillis();
      for (int i=0; i<mNumCalls; i++) {
        String Answer = new String(mES.DoService(Msg));
      }
      Time2 = System.currentTimeMillis();
      System.out.println("CechoServiceStartUp - duration of "+mNumCalls+" DoService invokations =
"+(Time2-Time1)+" Millis.");
    }
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

}// end class
```

B6 – Package ft_echoServiceApplication, Server

```
/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : javaserver1.java                                           */
/* Abstract : This is basic implementation of a fault tolerant client server  */
/*            application using the OMG trading service.                 */
/*            This program was developed as a part of my M.Sc. thesis at     */
/*            TCD.                                                       */
/*            A client connects to a service. When the connected service gets */
/*            killed, the client smart proxy automatically reconnects to the  */
/*            backup service. As soon as the master service is back on line,  */
/*            the client smart proxy switches back to the master service.    */
/*            This starts up a server application.                       */
/*************************************************************************/
/* Revision : 1.0                                                        */
/* Modified :                                                            */
/*************************************************************************/

package ft_echoServiceApplication;

public class javaserver1 {

  public static void main(String args[]) {

    // constants
    // echoService (ES) type name
    final String mESType=new String("ft_echoService_if");
    // echoService (ES) trader offer type name
    final String mESOffer=new String("IDL:ft_echoServiceType:2.1");
    // name of the configuartion file
    final String mCfgFileName  = new
String("/dd/msc/meierr/dis_orbix/Web/appl/ft_echoService/ft_echoService1.cfg");
    // server and service name
    final String mServerName   = new String("echoServer1");
    final String mServiceName1 = new String("Echo Service One");
    final String mServiceName2 = new String("Echo Service Two");
    // master and backup lists
    final String mMList1       = new String("0001");// master
    final String mMList2       = new String("0000");// master
    final String mPBList1      = new String("0000");// primary backup
    final String mPBList2      = new String("0002");// primary backup
    final String mSBList1      = new String("0000");// secondary backup
    final String mSBList2      = new String("0000");// secondary backup

    // start the server
    CechoServiceServer TheServer = new CechoServiceServer(mESType, mESOffer, mCfgFileName,
mServerName, mServiceName1 ,mServiceName2, mMList1, mPBList1, mSBList1, mMList2, mPBList2,
mSBList2);

    TheServer.run();

  }//end main
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : javaserver2.java                                           */
/* Abstract : This starts up a server application.                       */
/*************************************************************************/

package ft_echoServiceApplication;

public class javaserver2 {

  public static void main(String args[]) {

    // constants
    // echoService (ES) type name
    final String mESType=new String("ft_echoService_if");
    // echoService (ES) trader offer type name
    final String mESOffer=new String("IDL:ft_echoServiceType:2.1");
    // name of the configuartion file
```

```
    final String mCfgFileName  = new
String("/dd/msc/meierr/dis_orbix/Web/appl/ft_echoService/ft_echoService1.cfg");
    // server and service name
    final String mServerName   = new String("echoServer2");
    final String mServiceName1 = new String("Echo Service One");
    final String mServiceName2 = new String("Echo Service Two");
    // master and backup lists
    final String mMList1       = new String("0000");// master
    final String mMList2       = new String("0002");// master
    final String mPBList1      = new String("0001");// primary backup
    final String mPBList2      = new String("0000");// primary backup
    final String mSBList1      = new String("0000");// secondary backup
    final String mSBList2      = new String("0000");// secondary backup

    // start the server
    CechoServiceServer TheServer = new CechoServiceServer(mESType, mESOffer, mCfgFileName,
mServerName, mServiceName1 ,mServiceName2, mMList1, mPBList1, mSBList1, mMList2, mPBList2,
mSBList2);

    TheServer.run();

  }//end main
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : javaserver3.java                                           */
/* Abstract : This starts up a server application.                       */
/*************************************************************************/

package ft_echoServiceApplication;

public class javaserver3 {

  public static void main(String args[]) {

    // constants
    // echoService (ES) type name
    final String mESType=new String("ft_echoService_if");
    // echoService (ES) trader offer type name
    final String mESOffer=new String("IDL:ft_echoServiceType:2.1");
    // name of the configuartion file
    final String mCfgFileName  = new
String("/dd/msc/meierr/dis_orbix/Web/appl/ft_echoService/ft_echoService1.cfg");
    // server and service name
    final String mServerName   = new String("echoServer3");
    final String mServiceName1 = new String("Echo Service One");
    final String mServiceName2 = new String("Echo Service Two");
    // master and backup lists
    final String mMList1       = new String("0000");// master
    final String mMList2       = new String("0000");// master
    final String mPBList1      = new String("0000");// primary backup
    final String mPBList2      = new String("0000");// primary backup
    final String mSBList1      = new String("0001");// secondary backup
    final String mSBList2      = new String("0002");// secondary backup

    // start the server
    CechoServiceServer TheServer = new CechoServiceServer(mESType, mESOffer, mCfgFileName,
mServerName, mServiceName1 ,mServiceName2, mMList1, mPBList1, mSBList1, mMList2, mPBList2,
mSBList2);

    TheServer.run();

  }//end main
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CechoServiceServer.java                                    */
/* Abstract : This is an implementation of a server.                     */
/*************************************************************************/

package ft_echoServiceApplication;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import CosTradingDynamic.*;
import ft_echoService.*;
```

```
import ft_Components.*;
import ft_Exceptions.*;

public class CechoServiceServer {

  // echoService (ES) type name
  private String mESType       = null;
  // echoService (ES) trader offer type name
  private String mESOffer      = null;
  // name of the configuartion file
  private String mCfgFileName  = null;
  // server and service name
  private String mServerName   = null;
  private String mServiceName1 = null;
  private String mServiceName2 = null;
  // master and backup lists
  private String mMList1       = null;// master
  private String mMList2       = null;// master
  private String mPBList1      = null;// primary backup
  private String mPBList2      = null;// primary backup
  private String mSBList1      = null;// secondary backup
  private String mSBList2      = null;// secondary backup

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceServer(String ESType, String ESOffer, String CfgFileName,
                            String SrName, String ScName1, String ScName2,
                            String MList1, String PBList1, String SBList1,
                            String MList2, String PBList2, String SBList2) {
    // init local instances
    mESType       = new String(ESType);
    mESOffer      = new String(ESOffer);
    mCfgFileName  = new String(CfgFileName);
    mServerName   = new String(SrName);
    mServiceName1 = new String(ScName1);
    mServiceName2 = new String(ScName2);
    mMList1       = new String(MList1);
    mPBList1      = new String(PBList1);
    mSBList1      = new String(SBList1);
    mMList2       = new String(MList2);
    mPBList2      = new String(PBList2);
    mSBList2      = new String(SBList2);
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public void run() {

    // server shuts down after not receiving any client requests for this time
    final int ServerTimeout = 60*60*1000;

    // local instances
    ORB              Orb             = null;
    ft_echoService_if echoServiceImpl1 = null;// service object
    ft_echoService_if echoServiceImpl2 = null;// service object
    CTraderIF         TdrIF           = null;

    // init orb
    try {
      Orb = ORB.init();
    }
    catch (Exception ex) {
      System.out.println("Exception during ORB init!");
      ex.printStackTrace();
      System.exit(1);
    }

    // instanciate the services
    echoServiceImpl1 =
      new CechoServiceImplementation((mServiceName1+" on "+mServerName), 0);
    echoServiceImpl2 =
      new CechoServiceImplementation((mServiceName2+" on "+mServerName), 0);

    // get an instance of the trader interface
    try {
      TdrIF = new CTraderIF(Orb, mESType, mESOffer, mCfgFileName);
    }
    catch (ConfigurationNotAvailable ex) {
      System.out.println("Exception during instanciating trader - no config.!");
      ex.printStackTrace();
      System.exit(1);
    }
```

```
    catch (TraderIORNotAvailable ex) {
      System.out.println("Exception during instanciating trader - no IOR!");
      ex.printStackTrace();
      System.exit(1);
    }

    // instanciate dynamic property objects, not in use
    DynamicPropEval DP_SerUtil = new CServerUtilization();
    DynamicPropEval DP_NumUser = new CNumOfUsersOnServer();

    // export service offers
    try {
      TdrIF.Register(mServerName,mServiceName1,
                     mMList1,mPBList1,mSBList1,true,
                     DP_SerUtil,DP_NumUser,echoServiceImpl1);
      TdrIF.Register(mServerName,mServiceName2,
                     mMList2,mPBList2,mSBList2,true,
                     DP_SerUtil,DP_NumUser,echoServiceImpl2);
    }
    catch (RegisterOfferWithTraderFailed ex) {
      System.out.println("Exception during register offer with trader!");
      ex.printStackTrace();
      System.exit(1);
    }

    // wait for service requests
    System.out.println(mServerName+" is waiting for client requests...");
    _CORBA.Orbix.impl_is_ready(mServerName, ServerTimeout);

    // shutting down server after timeout
    System.out.println("Shutting down "+mServerName);
    // disconnect the services
    Orb.disconnect(echoServiceImpl1);
    Orb.disconnect(echoServiceImpl2);
    System.out.println (mServerName+" exiting...");

  }//end method
}// end class

/*************************************************************************/
/* Author   : Meier, R.                                                  */
/* Date     : August 1998                                                */
/* Filename : CechoServiceImplementation.java                            */
/* Abstract : This is an implementation of a service.                    */
/*************************************************************************/

package ft_echoServiceApplication;

import IE.Iona.OrbixWeb._OrbixWeb;
import ft_echoService.*;
import ft_Components.*;
import ft_echoService.ft_echoService_ifPackage.*;// IDL exceptions

public class CechoServiceImplementation extends _ft_echoService_ifImplBase {

  private int    mStatus = 0;   // counts the number of service calls
  private CDebug mDebug  = null;// debug level

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceImplementation(String Marker, int Status) {
    super(Marker);
    mStatus = Status;

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CechoServiceImplementation() {
    super("NoName service");
    mStatus = 0;

    // set debug level
    mDebug = new CDebug(mDebug.mHigh);

  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                       */
```

```
/* implementation of the services.                                          */
/* The ServiceName_ft methodes will never be called, these calls are        */
/* handled in the clients smart proxy class.                                */
/*                                                                          */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String DoService(String Msg) {
    ++mStatus;// inc number od service calls
    String ServiceName = _OrbixWeb.Object(this)._marker();

    if (mDebug.mLevel>=mDebug.mHigh) {
      System.out.println("CechoServiceImplementation - DoService called.");
    }

    //
    String Answer = new String(Msg+" : "+ServiceName+" : service answer!");
    return Answer;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public String DoService_ft(String Msg) throws NoValidObjectReference {
    return DoService(Msg);
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public int GetStatus() {
    return mStatus;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public int GetStatus_ft() throws NoValidObjectReference {
    return GetStatus();
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // implementation of the echo service ping operation, used by client SP only
  public int Ping(int Test) {
    return Test;
  }// end method

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  // implementation of the echo service ping operation, used by client SP only
  public int Ping_ft(int Test) throws NoValidObjectReference {
    return  Ping(Test);
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CServerUtilization.java                                        */
/* Abstract : This is an implementation of a dynamic property class used     */
/*            for load balancing.                                            */
/*****************************************************************************/

package ft_echoServiceApplication;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import CosTradingDynamic.*;

public class CServerUtilization extends _DynamicPropEvalImplBase {

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CServerUtilization() {
    super();
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public Any evalDP(String name, TypeCode returned_type, Any extra_info) {
    System.out.println("CServerUtilization - Dynamic property implementation called");
    Any a = ORB.init().create_any();
    a.insert_long(22);
```

```
    return a;
  }// end method
}// end class

/*****************************************************************************/
/* Author   : Meier, R.                                                      */
/* Date     : August 1998                                                    */
/* Filename : CNumOfUsersOnServer.java                                       */
/* Abstract : This is an implementation of a dynamic property class used     */
/*            for load balancing.                                            */
/*****************************************************************************/

package ft_echoServiceApplication;

import IE.Iona.OrbixWeb._OrbixWeb;
import IE.Iona.OrbixWeb._CORBA;
import org.omg.CORBA.ORB;
import org.omg.CORBA.*;
import CosTradingDynamic.*;

public class CNumOfUsersOnServer extends _DynamicPropEvalImplBase {

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public CNumOfUsersOnServer() {
    super();
  }// end constructor

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  public Any evalDP(String name, TypeCode returned_type, Any extra_info) {
    System.out.println("CNumOfUsersOnServer - Dynamic property implementation called");
    Any a = ORB.init().create_any();
    a.insert_long(99);
    return a;
  }// end method
}// end class
```