A Survey of Fault Tolerant CORBA Systems

Muhammad Fahad[1], Aamer Nadeem[1], and Michael R. Lyu[2]

[1] Department of Computer Science
Mohammad Ali Jinnah University, Islamabad, Pakistan
mhd.fahad@gmail.com, aamern@acm.org
[2] Department of Computer Science and Engineering
Chinese University of Hong Kong, Hong Kong S.A.R., China
lyu@cse.cuhk.edu.hk

Abstract. CORBA is an OMG standard for distributed object computing; but despite being a standard and wide scale acceptance in the industry it lacks the ability to meet high demands of quality of service (QoS) required for building a reliable fault tolerant distributed system. To tackle these issues, in 2001, OMG incorporated fault tolerance mechanisms, QoS policies and services in its standard interfaces as mentioned in its Fault Tolerant CORBA (FT-CORBA) specification. FT-CORBA Architecture used the notion of object replication to provide reliable and fault tolerant services. In this paper, we surveyed the different approaches for building FT-CORBA based distributed systems with their merits and limitations. We gave an overview of FT-CORBA specification; its requirements and limitations, and FT-CORBA Architecture. We have also revised the existing categorization of FT-CORBA systems by incorporating a fourth approach, i.e., Reflective Approach, in the categorization taxonomy. A comparison between different types of replication and FT-CORBA based systems is conducted to achieve quick insight on their features.

Keywords: CORBA Middleware, Object Replication Styles, Fault Tolerant CORBA Specification, Fault Tolerant CORBA systems.

1 Introduction

Distributed systems are used in a variety of application domains in which services are provided by independent components working together as a single transparent system. In distributed systems, CORBA is accepted as a standard because of its inherent location transparency, portability, interoperability and language independence [1]. With these features, CORBA was made a standard for distributed object computing by the Object Management Group (OMG) [2]. In CORBA, Interface Definition Language (IDL) defines interfaces to objects. Clients have to implement IDL interfaces to access server functionality and this makes CORBA language independent. By location transparency, clients can invoke server objects without worrying about the location of the server objects. Portability makes CORBA independent of specific ORB and the system can be implemented and used on top of any CORBA-compliant ORB. This is achieved by the Portable Object Adaptor (POA), a component of CORBA, which is responsible for making server-side functionality appear as CORBA object to clients.

R. Meersman and Z. Tari et al. (Eds.): OTM 2007, Part I, LNCS 4803, pp. 505–521, 2007.

Interoperability of CORBA ensures the system to be used by the clients and servers, running on ORBs from different vendors. Despite these benefits, CORBA does not address partial failures and does not provide totally ordered multicast of messages while building distributed systems [2], which are the key factors of fault tolerance.

To provide fault tolerance in distributed CORBA based systems, Fault Tolerant (FT) CORBA specification defines interfaces, QoS policies, associated fault tolerance mechanisms and services to enhance the reliability of CORBA applications [3]. Existing fault tolerant CORBA systems provide fault tolerance through replication of CORBA objects. By replicated objects, fault tolerant services are provided even if one of individual entities fails. A replicated object is implemented by a set of distinct CORBA objects called an *object group,* i.e., an abstraction to provide replication transparency and failure transparency [4]. These systems differ mostly at the level at which the replication mechanism support is introduced. Felber and Narasimhan categorize the FT CORBA systems on this basis into three categorizes: *integration, interception and service* [1] and discuss the experiences and lessons learnt in building their two distinct FT CORBA systems.

This paper presents the overview of Fault Tolerant CORBA Specification; its architecture, requirements and limitations. We surveyed the FT-CORBA systems based on their FT properties and highlighted their prominent features and limitations. We analyzed the several different approaches that implement FT-CORBA and revised the existing categorization of FT-CORBA systems by incorporating a fourth approach called Reflective Approach in the existing categorization taxonomy. Moreover we compared the working of individual systems on different criteria and provide analysis matrix to achieve quick insight on their infrastructures.

The rest of this paper is organized as follows: Section 2 covers the approaches of building the fault tolerant CORBA system with their merits and limitations. Section 3 throws light on replication styles and comparison of these styles. Section 4 gives the overview of Fault Tolerant CORBA Specification; its requirement, architecture and limitations. Section 5 covers the Fault Tolerant CORBA systems with a critical look on their features. Section 6 shows our analysis about the FT-CORBA systems. Section 7 concludes the paper.

2 Fault Tolerant Approaches

According to the built-in support of replication logic, various fault tolerant systems are categorized into four approaches, which are termed as Integration approach, Interception approach, Service approach and Reflective approach. Taxonomies of the first three approaches can be found in [5,6,7]. With the passage of time new FT-CORBA Systems were built introducing fourth approach, i.e., Reflective approach. So here we incorporate the fourth approach in the existing (old) categorization taxonomy. Summarized features of these approaches are represented in Table 1.

2.1 Integration Approach

In this approach, support for replication is integrated transparently into the ORB. It integrates necessary fault tolerant replication by proprietary mechanisms in the ORB.

This is the most efficient approach but modification in the ORB makes this approach non-compliant with the CORBA standard, i.e., does not enable off-the-shelf ORBs to be used. In this approach, modified ORB gets a message from application objects, passes it to the adapter object that multicasts it by using underlying toolkit. Fault tolerance mechanisms and replication strategies are transparent to the client as these are integrated into the ORBs. Portability is not achieved but interoperability can be achieved depending on the support for *IIOP* invocations (Internet Inter-ORB Protocol, a CORBA Standard for invocations).

2.2 Interception Approach

In this approach, support for replication is provided underneath the ORB, which makes the replication logic transparent to the users. Messages from client and server are intercepted transparently, externally to the ORB by using low level (OS-level) interceptor and then multicast by the group communication toolkit. The use of low-level interceptor makes this approach non-portable. Moreover as there is no need of modification in ORB, thus systems built using this approach are ORB compliant. Interoperability can be achieved by writing an interception layer of each distinct OS.

2.3 Service Approach

In this approach, support for replication is provided through a collection of CORBA objects that reside above the ORB. As there is no need to modify the ORB, the systems built exploiting this approach are CORBA compliant, interoperable and portable. To use service objects that provide the policies and mechanisms for achieving fault tolerance, application objects require knowledge of these service objects and hence application code needs modifications to use their functionality. Each request from application objects to service objects passes through the underlying ORB, which increases performance overheads. Service objects are defined as IDL interfaces so they are independent of language constructs. Service objects can be made distributed by locating them on different hosts on the network.

Table 1. Comparison between Fault tolerant Approaches

System Features	Integration	Interception	Service	Reflective
ORB Compliance	No	Yes	Yes	Yes
Transparency	Yes	Yes	Depends on service implementation	Yes
OS dependence	No	Yes	No	No
Portability	No	Can be achieved	Yes	Yes
Interoperability	Depends on IIOP invocation	Yes	Yes	yes
Performance	Most efficient	Efficient	Good	Good
System Example	Electra, PPF, Orbix+Isis	Eternal, CARRIGE	OGS, DOORS, AQuA, Newtop, FTS, IRL, Aquarius	FRIENDS, FT-MOP

2.4 Reflective Approach

Reflection approach separates the concerns between the application and the fault tolerance mechanisms and enables off-the-shelf ORBs to be used. It employs metalevel architecture to integrate fault tolerance in CORBA systems and provides a means to develop transparent fault tolerance software as any CORBA software with different object-oriented languages. In this approach the replication necessarily involves creating a single point of failure outside the client's failure domain, thus partially defeating the purpose of the replication (no single failure is visible to the client). The use of *Metaobjects Protocol* (MOP) and restricted reflective features of some object-oriented languages makes this approach different from other approaches. Using this approach MOP can be implement as compile time or runtime. But the integration of runtime and compile-time MOPs enables more efficient functionality for fault tolerance. This MOP is CORBA compliant which enables the execution and the state evolution of CORBA objects to be controlled. Metaobjects are not only used for the purpose of fault tolerance but can also be used for security purposes. Interoperability can be achieved with the engagement of Metaobjects protocol.

All these approaches support different types of object replication styles in which they replicate their constituent objects. The need for object replication is to increase the reliability and performance of the system. Failure of a replica does not affect the services provided, as other replicas are there to give the required services. Performance issues arise when distributed systems need to scale in number and geographical area [4]. Fault tolerance benefits can be achieved only when object replication maintains strong replica consistency. Strong replica consistency means that all the replicated objects should have the same state and they perform the same behavior. There are many issues which should be analyzed while maintaining strong replica consistency [7,8]. First, all the replicas perform the same sequence of operations in the same order. Second, to perform a single invocation multi-replicated client objects initiate a request to replicated server objects, thus each of server objects receive multiple requests made by each of the client object. Therefore, the system should be capable enough to detect duplicate requests. Third, systems which support multithreading should carefully analyze different threads and the functions they perform. Fourth, in case of failure of replicated objects, recovery mechanisms should be transparently managed to provide the reliable fault tolerant services.

3 Replication Styles

The replication logic is a set of protocols, mechanisms and services that allow a CORBA system to handle object replication [9]. There are many styles of object replication but the main ones are Active replication and Passive replication [5,6,7]. Underlying mechanisms for both are the same but their role to provide strong replica consistency is different. Some of the FT-CORBA systems rely on proprietary group toolkit for replication logic implementation but others provide either centralized replication logic in its core or completely distributed above the ORBs [9]. A comparison between replication styles is represented in Table 2.

3.1 Active Replication

In Active Replication, all replicated objects are active and independently handle client requests and return the responses to the client. Duplicate responses should be detected and suppressed to provide client transparency. One of these active replica objects is called primary, while others act as backup. The crash failure of single primary is masked by the presence of other active replica by providing fault tolerant services; thus this style provides better fail-over time, and state transfer and recovery mechanisms are provided to regain the use of the crashed node. To ensure replica consistency, it consumes a lot of computational resources and totally ordered multicast of messages is needed to maintain the same state and to achieve same behavior by active replicas, i.e., it needs operations on the replicated objects to be deterministic. It shields fastest recovery from faults.

3.2 Passive Replication

In Passive Replication only one operational replica is active, termed as primary, to fulfill client request. It requires less memory and processing costs, and shields slower recovery from faults. On the basis of recovery mechanisms it has two variations:

Warm Passive. Only one server replica (primary) is active in each object group and remaining replicas are preloaded into the memory and are synchronized periodically to handle state transfer while crash faults. To achieve this state synchronization, totally ordered multicast as well as deterministic operations are needed. Only active replica is operational to fulfill client request, while backups are running for the sake of state storage and state transfer in case of primary failure. When primary fails, new primary is selected from the backup replicas.

Table 2. Comparison between Replication Styles

Analysis Parameters	Active	Warm Passive	Cold Passive
Number of operational replica	All	Only primaries	One
Fail-over time	Very low	Medium	Very high
Computational resources	High	Medium	Low
Duplicate message detection and suppression required	Yes	Yes	No
Totally ordered multicast required	Yes	Yes	No
Operations on replicated objects	Deterministic	Deterministic	Non-deterministic
Recovery from faults	Fastest, very Rapid	Rapid	Slower

Cold Passive. Only one server replica is active and the remaining replicas are not even preloaded into the memory. State of the primary is logged into the storage for recovery mechanisms. If the primary fails, new primary is created and state is transferred from logged storage to the new primary, which increases the fail-overtime. This approach uses less resources and non-deterministic operations, as only one replica is operational at a time.

4 Overview of FT-CORBA

In 1998, Object Management Group (OMG) felt the need of making fault tolerant standard properties for CORBA Architecture for adding availability and reliability in CORBA applications. Hence issued a Request For Proposal (RFP) that results the Fault Tolerant CORBA specifications in early 2000 [3]. FT-CORBA specification addressed the issues of entity redundancy, fault detection, and fault recovery. This section throws a light on FT-CORBA Specification.

4.1 FT-CORBA Architecture

The Fault Tolerant CORBA Architecture [3] is achieved by handling issues of object replication transparently, fault detection and recovery mechanisms in CORBA Architecture as shown in the Fig. 1. Major components with their functionality are:

Replication Manager. Replication Manager has three components; *Property Manager*, *Generic Factory*, and *Object Group Manager*. Property Manager allows application developer to choose and set object group properties i.e. replication style, consistency style, membership style etc according to requirements. Generic Factory creates objects and makes object groups. Object Group Manager adds or deletes members.

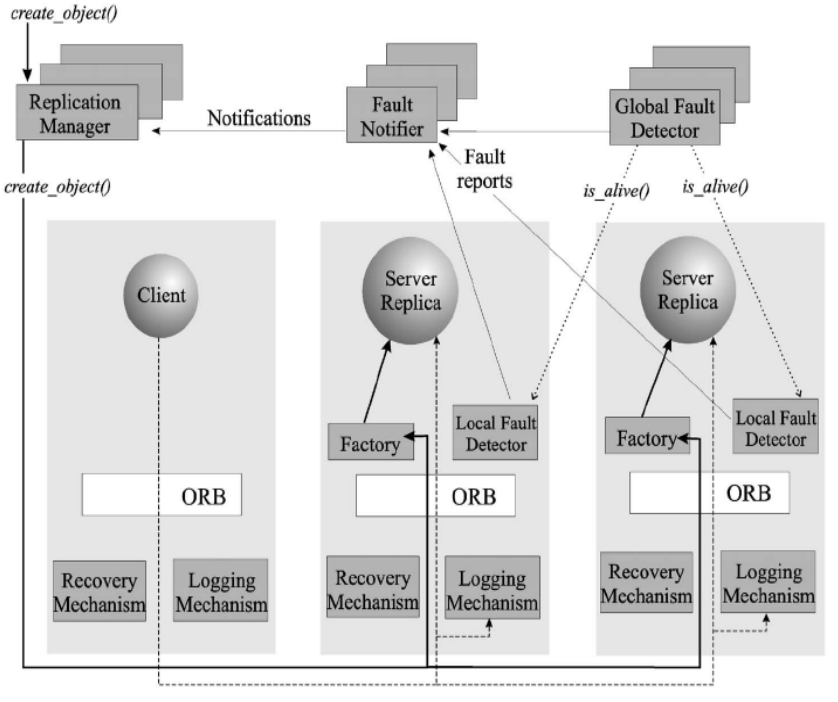

Fig. 1. The Architecture of Fault Tolerant CORBA [7]

Fault Detector and Fault Notifier. *Fault Detector* supports Pull Model and Push Model based fault monitoring [3]. In *Pull Monitoring*, crash faults are detected by invoking an isAlive() method of monitored object asking about its aliveness. If monitored object does not reply within some time interval then it is assumed that object has crashed. By this approach, application checks the status of objects when it is needed [10]. In *Push Monitoring*, crash faults are detected on the basis of I_am_Alive() messages sent by monitored object who tells about its aliveness. Crash fault of monitored object is assumed when it does not send message telling about its aliveness. By this approach fast detection of the crash failure is achieved [10]. Whenever a fault is detected, *Fault Detector* reports the fault to *Fault Notifier*, which diverts it to *Replication Manager* to take necessary actions. There should be separate *Fault Detector* and *Fault Notifier* components according to standard Fault tolerant CORBA specification.

Logging and Recovery. FT-CORBA defines a logging and recovery mechanisms by two IDL interfaces (*Checkpointable* and *Updateable*). The logging mechanism periodically stores object related information on the log, and recovery mechanism retrieves log information to restore valid state to the crashed replica.

4.2 Requirements of FT-CORBA Specification

According to FT-CORBA specification [3], system build on CORBA middleware should preserve CORBA object model for the infrastructure-controlled consistency style, and extended format of *Interoperable Object Reference* (IOR) should be used for the individual replicas so that legacy ORBs that does not support fault tolerance can invoked methods on ORBs that support fault tolerance and vice versa. Each component should be replicated to avoid single point of failure; moreover creation and deletion of objects, fault detection, and recovery mechanisms should be invisible to client to achieve transparency. In case of failure of replica, client's request should be transparently redirected to other available replica and client ORB systematically re-initiate the request until the request fulfills.

4.3 Limitations of FT-CORBA Specification

FT-CORBA specification [3] has many limitations that are: i) Clients running on non-FT-CORBA can invoke methods/operations on an object group, supported by the fault tolerant infrastructure without taking the benefits of its fault tolerant properties. ii) To achieve interoperability and full fault tolerance, the hosts with in a domain should use fault tolerant infrastructure and ORBs from the same vendor. iii) To achieve strong replica consistency, specification addressed that application objects should have deterministic behavior. iv) There is no support for partitioned systems, and *Network-Partitioning faults*, *Commission faults* (wrong results generated by the objects), and *Correlated faults* (Design Faults, and Programming Logic Errors) are not addressed in the specification.

5 Existing Fault Tolerant CORBA Systems

Many FT-CORBA systems were developed to address the issues of secure group based communication for embedding fault tolerance by the notion of object replication. The evolution of FT-CORBA systems starts with the integration of fault tolerant properties in the ORB, but later on different approaches were introduced to build replication for ease of use and customization purpose to provide fault tolerance in CORBA based distributed systems. The following sub-sections present the brief introduction to various Fault Tolerant Systems.

5.1 Electra

The Electra [4] is one of the earliest implementation of fault tolerant CORBA systems, developed at the University of Zurich which exploits the integration approach. It was the first time using the strengths of CORBA model and improving the weaknesses of CORBA model with group communication for consistent ordering of distributed events and transactions, handling of partial failures and support of asynchronous communication.

The first research based CORBA object request broker, Electra, combines the benefits of CORBA object model and virtual synchrony with reliable group communication as part of an ORB to achieve fault tolerance. As the replication logic is embedded into the ORB, it neither is ORB compliant nor maintains interoperability of CORBA architecture. Also we cannot achieve interoperability using Electra. The key focus of Electra is to enable ORB with build-in fault tolerant capabilities. All the special features of adding fault tolerance are enhanced by two C++ interfaces *Basic Adaptor Object* (BOA) Interface and *Environment Interface*, so C++ is the only target language for building fault tolerant CORBA based application using Electra prototype. Underlying toolkit, which is built on the model of virtual synchrony, provides reliable multicast. BOA provides active replication and Environment Interface is responsible for synchronous, asynchronous and deferred-synchronous communication. Adaptor object has the code specific to the toolkit so application developer can use another toolkit by simply relinking the application with the appropriate Adaptor Object. Basic Adaptor Object, which is hooked into the ORB, is responsible for replication services and mechanisms like creation and deletion of objects and object groups, and state transfer when primary replica fails. It also allows application developers to select the ordering protocol given by the toolkit according to requirements. Group communication is achieved by the subsystems (Horus, Isis) that are built on the model of virtual synchrony to maintain replica consistency.

5.2 Orbix+Isis

First commercially available Fault Tolerant CORBA system [11] developed by the IONA Technologies was Orbix+Isis, which exploits the integration approach. *Isis* developed by the Isis Distributed Systems was the first commercial toolkit built upon the model of virtual synchrony to provide high performance, totally ordered multicast and fault monitoring. *Orbix* is the C++ development environment to work on distributed CORBA objects.

It modifies ORB to use *Isis* toolkit which provides totally ordered multicast reliable communication, object groups and failure monitoring, whereas *Orbix* provides the object oriented environment to work on distributed objects and supports point-to-point communication. Fault tolerant replication mechanisms are implemented by using two base classes *ActiveReplica* and *Stream Event*. *ActiveReplica* provides transparent Active and hot-passive replication, and *Event Stream* (supports asynchronous requests using publish/subscribe paradigm) makes object groups and used for load balancing. *Orbix+Isis* allows application developers to select the object replication execution style. Transparent replications of server objects and filter mechanisms are provided by the Orbix specific smart proxies. Active replica execution style also gives an option to select the replication style. In Event Stream style, Event Streams are replicated which keep event history and Event Log. Servers registered to specific events are invoked by Event Stream when it receives the event from the client. Fault monitoring is based on two functions *_newMember()* and *_memberLeft()*. The former is called when an object joins group and latter one is called when the object leaves.

5.3 Eternal

Eternal [1,7], a FT CORBA standard, was developed at the University of California, Santa Barbara, which exploits the interception approach to provide transparent fault tolerance to ORB and application as well. It employs *Totem* toolkit for totally ordered multicast.

Eternal has an ORB compliant architecture but does not maintain interoperability of CORBA because when request came, it is captured by OS-level interceptor and then propagated to ORB, thus making Eternal OS dependant. Interoperability can still be achieved by writing a separate interception layer for every different ORB. It supports active and different types of passive replication (e.g. cold passive, warm passive) and logging-recovery mechanisms to provide reliable consistent replication. Active replication allows the Eternal to work, when single replica fails as this is masked by the presence of other active replicas and during recovery phase. For providing consistent replication it maintains three types of states; application level state, ORB/POA level state and Infrastructure state, and this distinguishes Eternal from other fault tolerant CORBA systems. It provides fault detection service based on user-defined timeouts to identify crash faults. It allows developers to select configuration management properties of fault tolerance, and employs mechanisms to overcome the non-determinism inherent in multithreaded CORBA applications.

5.4 DOORS

The Distributed Object Oriented Reliable Service (DOORS) [6] is an application-level framework developed at Lucent Technologies as an experimental middleware so that lessons learned during its implementation are integrated into the FT-CORBA standard. DOORS exploits the service approach to provide fault tolerance and follows an ORB compliant architecture which maintains interoperability of CORBA. The proposed architecture supports active and passive replication, but prototype implementation only provides passive replication. Both pull and push methods of fault monitoring are supported to provide fault detection and employs libraries for the transparent

checkpointing of applications. Fault detection and fault notification are merged into fault detector component. It provides transparently fault detection and fail-over to the client. The prototype does not support recovery and logging mechanisms, and duplicate detection and suppression of messages for reliable fault tolerance. Replication Manager is responsible for configuration management and replication mechanisms by allowing application developer to choose and set object group properties i.e. replication style and consistency style, according to requirements. Fault Detector detects the faults and reports them to super fault detector that diverts them to replication manager to take necessary actions. There is no separate Fault Notifier component thus it violates the standard fault tolerant CORBA specification. All these component services act as CORBA objects above the ORB.

5.5 AQuA

The BBN Technologies and University of Illinois, developed the AQuA's gateway architecture to provide adaptive fault tolerance to CORBA systems [12]. Its architecture consists of Quality Objects, Proteus, Maestro/Ensemble and gateways. It replaces the ORB IIOP implementation with proprietary gateway which propagates IIOP calls to other CORBA objects by using *Maestro/Ensemble* toolkit. The *gateway* and the group toolkit employ the replication logic. Due to replacement of *only IIOP module* of ORB with gateway, it is regarded to exploit integration approach [9]. But as the *gateway* captures the initial request by client object which acts as an OS-level interceptor, it is regarded to exploit interception approach [6]. We classified AQuA exploiting service approach, as it provides replication via a collection of CORBA objects above the ORB [1]. Nevertheless, interoperability is achieved by implementing gateway for each different OS and ORB. The configuration management regarding fault tolerance properties can be set during runtime. Push-based or heartbeat fault monitoring is supported for fault detection. Different types of active and passive replication schemes are supported to tolerate crash and value faults. The application developer can set the level of dependability by *Quality Objects* according to desired application requirements and state of the distributed system during execution of the system. *Proteus*, a flexible infrastructure, has replicated dependability manager, gateway handler and object factory. The replication dependability manager makes decisions on reported faults, manages configuration properties, and replicas are created and deleted by the object factories.

5.6 FTS

FTS [13] as a lightweight CORBA fault tolerance service was developed at Israel Institute of Technology that maintains the portability and interoperability of CORBA ORBs. It aims to support transparent client-side replication and embeds fault tolerance in CORBA by utilizing the standard CORBA's *Portable Object Adaptor* (POA).

It provides fail-over transparency and reliable transparent fault tolerance by redirecting a client's requests during processing. It supports two types of fault detection; process-based which is monitoring of *Group Object Adaptor* (GOA) and object-based in which all the objects are monitored which are connected with GOA by push fault monitoring model. Active replication of server objects is supported. A set of

components, which provide reliable functionality for fault tolerance, are merged into group object adaptor, which is built on the top of POA. *FTS Interceptors* detect faults during client-server replica communication and redirects a client's request to other replicas when they receive an indication of faults during request processing, thus adding reliable transparent fault tolerance to client applications. It partially supports network partitioning by imposing a primary component model.

5.7 IRL

IRL was developed by the University La Sapienza, Roma, Italy, which exploits the service approach [9]. It maintained the CORBA's interoperability and was built with supports of passive centralized replication logic. Later on, a distributed design was proposed to give more reliable fault tolerant properties [14].

With its replication logic implemented as CORBA objects above the ORB, IRL offers interoperable ORB compliant architecture in which all the components are deployed distributedly to avoid single point of failure, thus adding more reliable fault tolerant ways to handle client's request in a more transparent manner. Adding support of the client-side replication and the server-side replication to system makes IRL more reliable while achieving good performance. Object Replicas are distributed in different host domains for balancing loads and achieving high fault tolerance. To handle object creation and deletion, replication style and its management, *Object Group Handler* (OGH) and *Object Group* (OGs) Components were designed. Fault detector and fault notifier detect faults and provide fail-over transparency to clients. Host-specific IRL components as well as domain-specific IRL components handle failure management activities. Local failure detectors monitor crash faults by pull fault tolerant technique. Recovery mechanisms are carried out by Object Group component, which ensures strong replica consistency in a group.

5.8 OGS

Object Group Service (OGS) [5] developed at the Swiss Federal Institute of Technology, Lausanne, exploits the service approach as the first time in the history to provide fault tolerance in a CORBA system. It maintains interoperability and provides distributed replication support in building more reliable fault tolerance.

OGS supports a set of independent generic IDL specified interfaces, which provides transparent group invocations. It preserves portability of CORBA ORBs, and provides both reliable (for read-only client requests) and unreliable multicast of messages, and mechanisms for duplicate detection and suppression. Furthermore, it supports active and warm passive replication techniques, as well as fault monitoring by push and pull methods. *Group Service* component manages work related to objects and group membership and provides client transparency. The consensus service ensures the total ordered multicast and replica consistency, crash fault detection is done by monitoring service, and messaging service transmits client server invocations onto the transport layer. Replication service employs the user to select replication style and other fault tolerant properties. Clients implement IDL interfaces to use a set of services of known replicated server so it does not maintain replication transparency.

Furthermore, recovery services are used incase of failures of object replicas and for the transfer of application-level state.

5.9 Newtop

Newtop [15] was developed by the University of Newcastle, which exploits the service approach. It follows the similar approach as being implemented by OGS but it provides more group management facilities. It embeds the support for objects belonging to multiple groups and handling the failures due to partitioning. *Newtop Service Object* (NOS), provides the distributed mechanisms and handles client requests in a fault tolerant way. It achieves its functionality by three services implemented as an object, i.e., Group management service object, Invocation/multicast service object and Membership service object. Group management service is responsible for creation and deletion of objects from groups. Invocation/multicast service provides synchronous and asynchronous communication facilities and information about the object is kept by the Membership service. However it does not employ consistent remerging of the subgroups once communication is reestablished. Membership service is also held responsible for checking crash faults on the bases of a timeout protocol.

5.10 Aquarius

Aquarius was developed at the Hebrew University of Jerusalem, Israel [16]. It exploits the service approach and is based on Quorum Object Adaptor Architecture [17]. It provides the data-centric approach to build fault tolerance in CORBA.

Aquarius embeds server-side replication support by using the object adaptor approach like FTS. But it modifies the adaptor by adding an ordering protocol's algorithm. It employs proxies (stateless servers), which act as middle tier between client and server. These proxies propagate client requests to server and help to achieve efficient client-server invocation and transparency. It consists of two parallel threads of execution, one is responsible for propagating client requests to all replica servers and other is responsible for creating a total order of all client requests. Its architecture is similar to that of IRL but the middle tier of Aquarius uses independent entities that are not aware of each other and do not run any kind of distributed protocols among them. It applies the ordering protocol to maintain strong replica consistency. It utilizes RPC mechanisms that support asynchronous invocations for delivery of client requests to all replicas.

5.11 Pluggable Protocol Framework (PPF)

PPF was developed at the University of California, Santa Barbara, which utilizes the pluggable protocols framework to provide fault tolerance in CORBA [18]. It is an FT standard CORBA compliant infrastructure and achieves performance to maintain strong replica consistency, similar to DOORS or Eternal.

There is no need for any modification in CORBA ORB but PPF requires minimal modification in the application to run. It engages totem toolkit for totally ordered multicast of messages, fault detection and fault notification. FT protocol plug-in provides the fault tolerance on the server-side and client-side failover mechanisms. The Fault Detector, a component of FT protocol plug-in, detects the faults. Interoperability

is achieved by passively replicated gateways, which provide access of un-replicated clients to replicated servers. Active and semi-active replication styles are supported for strong replica consistency. Smart duplicate mechanisms are provided for duplicate message detection and suppression. This scheme is similar to the interception approach as it employs an underlying toolkit for message delivery but in fact it is closer to the integration as fault tolerant mechanisms are embedded inside the ORB. But it differs from the integration-based systems as no need modification in ORB is required and it can be ported from one ORB to another.

5.12 CARRIAGE

CARRIAGE [19] is a fault tolerant CORBA system developed at the Southeast University of China, which employs portable interceptors to integrate *ORBUS* and *EDEN* to achieve fault tolerant services in CORBA. ORBUS is a CORBA implementation, and EDEN is a fault tolerant framework provided by IRISA/INRIA, France. Both of these were combined together on the basis of standardized Portable Interceptor mechanisms.

EDEN uses active replication style to enhance fault tolerance services. It consists Replication Manager, which handles all activities related to object replication, and Total Order Component, which is responsible for totally ordered multicast of messages. ORBUS, an OMG CORBA specification implementation that supports C++ and JAVA programming environments to work with distributed CORBA objects. ORBUS supports *ClientRequestInterceptor* for client-side and *ServerRequest- Interceptor* for server-side request processing. The approach followed is similar to the integration approach, as interceptors are hooked into the ORB; but differs from it as CARRIGE maintains inherent features of CORBA (i.e., language transparency, location transparency, portability and interoperability), which plays a vital role in its success. Moreover, it fully follows the standard specification and application programs do not require any modification to use this framework.

5.13 Lightweight Fault Tolerance (LW-FT)

Felber introduced a lightweight approach of embedding fault tolerance in existing CORBA system [20]. It employs replicated gateways for client-server interactions and uses semantic repository for achieving fault tolerance in CORBA. Use of gateways enables two fault tolerant CORBA frameworks to bridge that are supported by different mechanisms and QoS.

The proposed architecture uses client-side FT mechanisms and keeps semantic repository about server objects for fault tolerant request processing. The client request is propagated through replicated *Gateway*, which uses semantic repository for request processing. *Semantic repository* helps to choose optimal protocols for component interactions, replica management, automatic request redirection in case of failure, cache management to avoid unnecessary invocations to the servers, and load balancing of client requests. However, this approach cannot be applied to passively replicated or non-deterministic servers, and does not address the issues of maintaining strong replica consistency.

5.14 FRIENDS

FRIENDS stand for Flexible and Reusable Implementation Environment for your Next Dependable System [21]. FRIENDS, a meta-object protocol developed at LAAS-CNRS, Toulouse, provides libraries of meta-objects for fault tolerance, secure communication and group-based distributed applications. It exploits the reflective approach as the first time to build fault tolerance in CORBA systems. It aims to provide flexibility through object-oriented libraries of meta-objects and enhance non-functional requirements such as security by using the meta-objects.

FRIENDS system engages separate meta-objects for providing fault tolerance in CORBA. The system is composed of three layers, *Kernel layer*, *System layer* and *User layer*. System layer is responsible for providing fault tolerance by detecting crash faults, stable storage, secure communication, and replication management. User layer is responsible for controlling application objects and remote object interactions. System layer is built on the top of the Kernel layer, which is either a *UNIX* kernel or a micro kernel. Due to being kernel specific, it does not maintain portability. It uses time-outs to detect crash faults and both replication styles (active and passive) to maintain strong replica consistency. By applying FRIENDS, non-fault tolerant applications do not invoke functions on fault tolerant applications. The drawback of FRIENDS is that it is not CORBA compliant and fault tolerance properties cannot be configured dynamically as the link between objects and meta-objects cannot be changed at runtime.

5.15 FT-MOP

A Reflective fault tolerant CORBA system was developed at LAAS-CNRS, which uses a *Fault Tolerant Meta-Object Protocol* (FT-MOP) to build fault tolerance in CORBA [22]. By FT-MOP, desirable fault tolerance properties can be attached to CORBA objects as CORBA Meta Objects and enables off-the-shelf ORBs to be used. Its architecture is an extension of FRIENDS with the elimination of its drawbacks, which is based on a general-purpose runtime meta-object protocol. FT-MOP provides more efficient functionality by using a general-purpose compile-time MOP to implement a runtime MOP, than by using only a runtime MOP as in the FRIENDS system.

FT-MOP controls the behavior and the state of application level CORBA objects. FT-MOP handles the creation, deletion and invocation of CORBA objects. The client sends a request to the server by using the stub to invoke the server's services, which are implemented as IDL interfaces. The request is propagated to Metaobjects through the Metastub. Metaobjects controls the behavior and state of the server. FT-MOP is ORB compliant and it maintains interoperability of ORBs. FT-MOP is C++ language dependant, but the reuse ability of this system in many application domains with different object-oriented languages distinguishes it from other systems.

6 Comparative Analysis

Table 3 shows a comparative analysis of the existing FT-CORBA systems, which provides a quick insight on the features of these systems. Analysis parameters are: Approach, Interoperability, ORB compliance, OS dependence, Fault detection and

notification, Replication transparency, Replication style, Replication implementation, Portability, Transparency to application, and FT-CORBA standard compliance. Values and meanings of these parameters are discussed above along with the systems.

Most of the FT-CORBA such as OGS, Eternal, DOORS, etc. provide fault monitoring based on non adaptive fault detectors [10], but their performance can be improved by using adaptive fault monitoring approaches, i.e., Discard Past Consider Present (DPCP) [23], or ADAPTATION [10] algorithms.

Table 3. Comparison among FT-CORBA Systems

Analysis Parameter	DOORS	FTS	IRL	OGS	Newtop	AQuA	Aquarius
Approach	service	service	service	service	service	service	service
Interoperability	yes	yes	yes	yes	yes	no	yes
ORB Compliance	yes	yes	yes	yes	yes	yes	yes
OS dependence	no	no	no	no	no	yes	no
Fault detection and notification	combined	separate	separate	combined	combined	separate	separate
Replication transparency	yes	yes	yes	no	no	yes	yes
Replication style	passive	both	passive	both	both	both	ordering protocol
Replication implementation	centralized	centralized	both	distributed	distributed	By Maestro /Ensemble	data-centric
Portability	yes	yes	yes	yes	yes	no	yes
Transparency to application	not always	not always	yes	no	no	not always	yes
FT-CORBA standard compliance	yes	No	no	no	no	no	no

Eternal	Isis+Orbix	Electra	CARRIGE	PPF	Friends	FT-MOP
interception	integration	integration	interception	integration	reflective	reflective
no	no	no	yes	yes	no	yes
yes	no	no	yes	yes	yes	yes
yes	yes	yes	no	no	yes	no
separate	combined	combined	separate	separate	separate	separate
yes	yes	yes	yes	yes	yes	Yes
both	both	passive	active	active, semi-active	metaobject protocol	metaobject protocol
by totem	by Isis	by Isis, Horus	EDEN	Totem	Open to programmer	open to programmer
no	no	no	yes	yes	no	yes
yes	yes	yes	yes	yes	yes	yes
yes	no	no	yes	yes	no	no

7 Conclusion

Traditional CORBA-based middleware cannot meet the demanding quality of service (QoS) for dependable systems, thus OMG fault tolerant CORBA specification addressed many of the QoS and fault tolerant mechanisms while maintaining CORBA's transparency, interoperability and simplicity of application programming. FT CORBA is not a replacement of fault tolerant infrastructure that were deployed before this specification, FT CORBA complements fault tolerant infrastructures by defining QoS policies, associated fault tolerance mechanisms and services to enhance the reliability of CORBA applications. This paper presents an overview of FT-CORBA specification; its architecture, requirements and limitations. We discussed the existing approaches for building CORBA based distributed systems, and evaluated the various fault tolerant CORBA systems by analyzing their prominent features and

limitations. We have discussed the various styles of replicating the objects of the application that provides fault tolerance for CORBA applications.

Acknowledgement. The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4150/07E).

References

1. Felber, P., Narasimhan, P.: Experiences, Strategies, and Challenges in Building Fault-Tolerant CORBA Systems. IEEE Transactions on Computers 53(5) (May 2004)
2. Object Management Group: The Common Object Request Broker: Architecture and Specification, 2.6(edn.) OMG Technical Committee Document, formal/02-01-02 (January 2002)
3. Object Management Group: Fault Tolerant CORBA (Final Adopted Specification), OMG Technical Committee Document, formal/01-12-29 (December 2001)
4. Maffeis, S.: Run-Time Support for Object-Oriented Distributed Programming, PhD thesis, Univ. of Zurich (February 1995)
5. Felber, P.: The CORBA Object Group Service: A Service Approach to Object Groups in CORBA, PhD thesis, Swiss Federal Inst. of Technology, Lausanne (1998)
6. Natarajan, B., Gokhale, A., Yajnik, S., Schmidt, D.C.: Doors: Towards High-Performance Fault Tolerant CORBA. In: DOA 2000. Proceedings of the Second International Symposium on Distributed Objects and Applications, pp. 39–48 (February 2000)
7. Narasimhan, P.: Transparent Fault Tolerance for CORBA, PhD thesis, Dept. of Electrical and Computer Engineering, University of California, Santa Barbara (December 1999)
8. Narasimhan, P., Moser, L.E., Smith, P.M.: State Synchronization and Recovery for Strongly Consistent Replicated CORBA Objects. In: Proceedings of the 2001 International Conference on Dependable Systems and Networks, Goteborg, Sweden, pp. 261–270 (2001)
9. Marchetti, C., Mecella, M., Virgillito, A., Baldoni, R.: An Interoperable Replication Logic for CORBA Systems. In: DOA. Proceedings of the Second International Symposium on Distributed Objects and Applications, Belgium, pp. 21–23 (September 2000)
10. Sotoma, I.: ADAPTATION - Algorithms to ADAPTive fAulT monItOriNg and Their Implementation on CORBA. In: DOA. Proceedings of the Third International Symposium on Distributed Object and Applications, pp. 219–228. IEEE, Los Alamitos (2001)
11. IONA and Isis: An Introduction to Orbix+Isis, IONA Technologies Ltd. and Isis Distributed Systems, Inc. (1994)
12. Cukier, M., Ren, J., Sabnis, C., Sanders, W.H., Bakken, D.E., Berman, M.E., Karr, D.A., Schantz, R.: AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In: Proceedings of the 17th IEEE International Symposium on Reliable Distributed Systems, pp. 245–253 (October 1998)
13. Friedman, R., Hadad, E.: FTS: A high performance CORBA fault tolerance service. In: Proceedings of the IEEE Workshop on Object-Oriented Real-Time Dependable Systems, pp. 61–68. IEEE Computer Society Press, Los Alamitos (2002)
14. Marchetti, C., Virgillito, A., Baldoni, R.: Design of an Interoperable FT-CORBA Compliant Infrastructure. In: ERSADS 2001. Proceedings of the 4th European Research Seminar on Advances in Distributed Systems Dependable Systems, Bertinoro, Italy, pp. 14–18 (May 2001)

15. Morgan, G., Shrivastava, S., Ezhilchelvan, P., Little, M.: Design and Implementation of a CORBA Fault-Tolerant Object Group Service In:Proceedings of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (June 1999)
16. Chockler, G., Malkhi, D., Merimovich, B., Rabinowitz, D.: Aquarius: A Data-Centric approach to CORBA Fault-Tolerance. In: DOA. The workshop on Reliable and Secure Middleware, in the 2003 International Conference on Distributed Objects and Applications, Sicily, Italy (November 2003)
17. Chockler, G., Malkhi, D., Dolev, D.: Quorum Based Approach to CORBA Fault-Tolerance. In: ERSADS 2001. University Residential Center of University of Bologna, Bertinoro (Forlì), Italy, pp. 14–18 (May 2001)
18. Zhao, W., Moser, L.E., Melliar-Smith, P.M.: Design and implementation of a pluggable fault tolerant CORBA infrastructure. In: Proceedings of the International Parallel and Distributed Processing Symposium, Fort Lauderdale, pp. 35–44 (April 2002)
19. Goncalves, F., Greve, P., Hurfin, M., Narzul, J.-P.L.: OPEN EDEN: a Portable Fault Tolerant CORBA Architecture. In: Proceedings of the Second International Symposium on Parallel and Distributed Computing, p. 88. IEEE Computer Society Press, Los Alamitos (2003)
20. Felber, P.: Lightweight Fault Tolerance in CORBA. In: DOA 2001. Proceedings of the International Symposium on Distributed Objects and Applications, pp. 239–247 (September 2001)
21. Fabre, J.C., Pérennou, T.: A Metaobject Architecture for Fault Tolerant Distributed Systems: The FRIENDS Approach. IEEE Transactions on Computers, Special Issue on Dependability of Computing Systems 47(1), 78–95 (1998)
22. Killijian, M.O., Fabre, J.C., Ruiz-García, J.-C., Chiba, S.: A Metaobject Protocol for Fault-Tolerant CORBA Applications. In: 17th IEEE Symp. on Reliable Distributed Systems, West Lafayette, Indiana, USA, pp. 127–134 (1998)
23. Sotoma, I.: DPCP (Discard Past Consider Present) - A Novel Approach to Adaptive fault Detection in Distributed Systems. In: FTDCS 2001. Proceedings of the Eight IEEE Workshop on Future Trends of Distributed Computing Systems, IEEE C.S, Los Alamitos (2001)