

1

Date: ~~September~~June 200911

2

Lightweight Fault Tolerance for

3

Distributed RT Systems

4

Version 1.0 Beta ~~#1~~#2

5

OMG Document Number: ptc/2011~~09~~-~~10~~06-~~08~~06

6

Standard document URL: <http://www.omg.org/spec/LWFT/1.0/PDF>

7

Associated File(s)*: <http://www.omg.org/spec/LWFT/2009080220110508>

8

9

Source document: Lightweight Fault Tolerance for Distributed RT Systems version [Version 1.0 alphaBeta](#)

10

[1](#) ([ptcmars/200911-108-018](#))

11

* Original file(s): Lightweight Fault Tolerance for Distributed RT Systems machine-readable files ([ptcmars/](#)

12

[200911-085-028](#))

1 Copyright © 2009, Object Management Group, Inc.
2 Copyright © 2006, 2009, Thales.
3 Copyright © 2006, 2009, PrismTech Group Ltd.

4 USE OF SPECIFICATION – TERMS, CONDITIONS & NOTICES

5 The material in this document details an Object Management Group specification in accordance with the terms,
6 conditions and notices set forth below. This document does not represent a commitment to implement any portion of
7 this specification in any company's products. The information contained in this document is subject to change
8 without notice.

9 LICENSES

10 The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-
11 free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute
12 copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed
13 to have infringed the copyright in the included material of any such copyright holder by reason of having used the
14 specification set forth herein or having conformed any computer software to the specification.

15 Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a
16 fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use
17 this specification to create and distribute software and special purpose specifications that are based upon this
18 specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that:
19 (1) both the copyright notice identified above and this permission notice appear on any copies of this specification;
20 (2) the use of the specifications is for informational purposes and will not be copied or posted on any network
21 computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3)
22 no modifications are made to this specification. This limited permission automatically terminates without notice if
23 you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the
24 specifications in your possession or control.

25 PATENTS

26 The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may
27 require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which
28 a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope
29 of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective
30 users are responsible for protecting themselves against liability for infringement of patents.

31 GENERAL USE RESTRICTIONS

32 Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications
33 regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved.
34 No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic,

- 1 electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--
- 2 without permission of the copyright owner.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c) (1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

MDA®, Model Driven Architecture®, UML®, UML Cube logo®, OMG Logo®, CORBA® and XMI® are registered trademarks of the Object Management Group, Inc., and Object Management Group™, OMG™, Unified Modeling Language™, Model Driven Architecture Logo™, Model Driven Architecture Diagram™, CORBA logos™, XMI Logo™, CWM™, CWM Logo™, IIOP™, MOF™, OMG Interface Definition Language (IDL)™, and OMG SysML™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

1 Software developed under the terms of this license may claim compliance or conformance with this specification if
2 and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the
3 specification. Software developed only partially matching the applicable compliance points may claim only that the
4 software was based on this specification, but may not claim compliance or conformance with this specification. In
5 the event that testing suites are implemented or approved by Object Management Group, Inc., software developed
6 using this specification may claim compliance or conformance with the specification only if the software
7 satisfactorily completes the testing suites.

1

OMG's Issue Reporting Procedure

2 All OMG specifications are subject to continuous review and improvement. As part of this process we encourage
3 readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting
4 Form listed on the main web page <http://www.omg.org>, under Documents, Report a Bug/Issue ([http://www.omg.org/
5 technology/agreement](http://www.omg.org/technology/agreement).)

Table of Contents

| | | |
|----|---------|---|
| 1 | | |
| 2 | 1 | Scope.....1 |
| 3 | 2 | Conformance.....1 |
| 4 | 3 | Normative References.....3 |
| 5 | 4 | Terms and Definitions.....3 |
| 6 | 5 | Symbols (and abbreviated terms).....5 |
| 7 | 6 | Additional Information.....7 |
| 8 | 6.1 | Changes to Adopted OMG Specifications.....7 |
| 9 | 6.1.1 | Extensions.....7 |
| 10 | 6.1.2 | Changes.....7 |
| 11 | 6.2 | Acknowledgments.....7 |
| 12 | 7 | Overall design rationale.....7 |
| 13 | 7.1 | General Architecture.....8 |
| 14 | 7.1.1 | FT-enabled middleware.....9 |
| 15 | 7.1.2 | Middleware-independent FT infrastructure.....9 |
| 16 | 7.1.3 | Server/Process Replication Management Independent from Access to Service.....10 |
| 17 | 7.1.3.1 | Client-side view / Object groups.....10 |
| 18 | 7.1.3.2 | Server-side view / Process Groups.....10 |
| 19 | 7.1.4 | Fault Tolerance Current.....10 |
| 20 | 7.1.5 | Invocation Timeouts.....10 |
| 21 | 7.1.6 | Location Forwarding.....10 |
| 22 | 7.1.7 | Naming Groups.....11 |
| 23 | 8 | Conceptual Model.....12 |
| 24 | 8.1 | Fault Tolerance Domain.....13 |
| 25 | 8.2 | Redundancy Management.....13 |
| 26 | 8.2.1 | Process Group.....14 |
| 27 | 8.2.2 | FTProcess (Process Replica).....15 |
| 28 | 8.2.3 | Replication Management.....17 |
| 29 | 8.2.4 | Passive Replication.....17 |
| 30 | 8.2.4.1 | Backup Insertion.....17 |
| 31 | 8.2.4.2 | Recovery.....17 |
| 32 | 8.2.5 | Active Replication.....18 |
| 33 | 8.3 | Fault Management.....18 |

| | | | |
|----|---------|--|----|
| 1 | 8.3.1 | Fault Containment..... | 18 |
| 2 | 8.3.2 | Fault Detection & Notification..... | 18 |
| 3 | 8.3.2.1 | FaultNotifier..... | 18 |
| 4 | 8.3.3 | Fault Analysis/Diagnosis..... | 18 |
| 5 | 8.3.4 | Connection Management..... | 19 |
| 6 | 8.4 | QoS Policies..... | 19 |
| 7 | 8.4.1 | FTProcess Recovery time..... | 19 |
| 8 | 8.4.2 | Maximum Service Localization Time..... | 19 |
| 9 | 8.4.3 | Service Access Properties..... | 19 |
| 10 | 8.4.4 | Invocation Maximum Blocking Time..... | 19 |
| 11 | 8.4.5 | Transport/Connection Timeouts..... | 19 |
| 12 | 8.5 | Transparent Failover..... | 19 |
| 13 | 8.5.1 | At Most Once Semantics..... | 20 |
| 14 | 8.5.2 | FT Context..... | 20 |
| 15 | 8.5.3 | Service Locator..... | 20 |
| 16 | 9 | OMG CORBA/IDL Platform Specific Model..... | 20 |
| 17 | 9.1 | Server Group..... | 20 |
| 18 | 9.2 | Server Replica Identity..... | 21 |
| 19 | 9.2.1 | Server Status..... | 21 |
| 20 | 9.2.1.1 | SRV_STATUS_RECOVERING..... | 21 |
| 21 | 9.2.1.2 | SRV_STATUS_READY..... | 22 |
| 22 | 9.2.2 | Server Replication Role..... | 22 |
| 23 | 9.2.2.1 | SRV_REPL_PRIMARY..... | 22 |
| 24 | 9.2.2.2 | SRV_REPL_BACKUP..... | 22 |
| 25 | 9.2.2.3 | SRV_REPL_ACTIVE..... | 22 |
| 26 | 9.3 | Naming Object Groups | 23 |
| 27 | 9.3.1 | Support for Aliases..... | 23 |
| 28 | 9.4 | ClientFailoverSemanticsPolicy..... | 23 |
| 29 | 9.5 | ObjectGroupMembershipPolicy..... | 24 |
| 30 | 9.5.1 | AUTO_ORB_CTRL | 25 |
| 31 | 9.5.2 | AUTO_OBJ_GRP_MNGR | 25 |
| 32 | 9.5.3 | USER_CTRL_OBJ_GRP_MNGR | 26 |
| 33 | 9.6 | ObjectGroupNameResolutionPolicy..... | 26 |
| 34 | 9.7 | FT_REQUEST Service Context..... | 26 |
| 35 | 9.8 | FTRequestDurationPolicy..... | 27 |
| 36 | 9.9 | Transport Heartbeats..... | 28 |
| 37 | 9.9.1 | TAG_FT_HEARTBEAT_ENABLED Component..... | 28 |
| 38 | 9.9.2 | Heartbeat Policy..... | 28 |
| 39 | 9.9.3 | Heartbeat Enabled Policy..... | 29 |
| 40 | 9.10 | Interoperable Object Group References..... | 29 |

| | | |
|----|--|-----------|
| 1 | 9.10.1 Support for Location Agents..... | 29 |
| 2 | 9.10.2 FT_GROUP_ID Service Context | 30 |
| 3 | 9.10.3 Persistent vs. Transient IOR..... | 31 |
| 4 | 9.10.4 Gateway..... | 31 |
| 5 | 9.11 GenericFactory..... | 31 |
| 6 | 9.11.1 Alias property..... | 32 |
| 7 | 9.12 ServerCallback interface..... | 32 |
| 8 | 9.13 GroupUpdateObserver interface..... | 32 |
| 9 | 9.13.1 on_update_group operation..... | 33 |
| 10 | 9.14 ServerGroupNotFound exception..... | 33 |
| 11 | 9.15 ServerGroupNotFound exception..... | 33 |
| 12 | 9.16 WrongStatus exception..... | 34 |
| 13 | 9.17 AlreadyRegistered exception..... | 34 |
| 14 | 9.18 UnsupportedCallback exception..... | 34 |
| 15 | 9.19 Server Group Manager..... | 34 |
| 16 | 9.19.1 register_server operation..... | 35 |
| 17 | 9.19.2 unregister_server operation..... | 36 |
| 18 | 9.19.3 server_ready operation..... | 36 |
| 19 | 9.20 ServerGroupObserver | 36 |
| 20 | 9.20.1 on_register_server operation..... | 37 |
| 21 | 9.20.2 on_unregister_server operation..... | 37 |
| 22 | 9.20.3 on_server_ready operation..... | 37 |
| 23 | 9.21 Server Group Manager Ext..... | 38 |
| 24 | 9.21.1 Implicit operation register_server..... | 38 |
| 25 | 9.21.2 Implicit operation unregister_server..... | 38 |
| 26 | 9.21.3 Implicit operation server_ready..... | 39 |
| 27 | 9.22 RecoverableServer Interface..... | 39 |
| 28 | 9.22.1 start_recovery operation..... | 39 |
| 29 | 9.23 ServerRecoveryManager interface..... | 39 |
| 30 | 9.24 ServerManager | 40 |
| 31 | 9.24.1 the_location attribute..... | 40 |
| 32 | 9.24.2 register_server operation..... | 40 |
| 33 | 9.24.3 server_ready operation..... | 40 |
| 34 | 9.24.4 unregister_server operation..... | 41 |
| 35 | 9.25 FT Current..... | 41 |
| 36 | 9.25.1 NoContext exception..... | 41 |
| 37 | 9.25.2 get_client_id operation..... | 42 |
| 38 | 9.25.3 get_retention_id operation..... | 42 |
| 39 | 9.25.4 get_expiration_time operation..... | 42 |
| 40 | 9.26 FaultNotifier..... | 42 |

| | | |
|----|--|----|
| 1 | 9.27 ServerCrashFault..... | 42 |
| 2 | 9.28 RecoveryObserver..... | 43 |
| 3 | 9.29 Fault Detection..... | 43 |
| 4 | 9.29.1 PullMonitorable interface..... | 43 |
| 5 | 9.29.1.1 is_alive operation..... | 44 |
| 6 | 9.29.2 PullMonitorableServer interface..... | 44 |
| 7 | 9.30 ObjectGroupManager..... | 44 |
| 8 | 9.31 Group Object Adapter..... | 45 |
| 9 | 9.31.1 create_id_for_reference operation..... | 46 |
| 10 | 9.31.2 reference_to_ids operation..... | 46 |
| 11 | 9.31.3 associate_reference_with_id operation..... | 46 |
| 12 | 9.31.4 disassociate_reference_with_id operation..... | 47 |
| 13 | 9.32 FT CCM Component..... | 47 |
| 14 | 9.32.1 Navigation/Introspection | 47 |
| 15 | 9.32.2 Component Activation..... | 48 |
| 16 | 9.33 At most once semantics..... | 48 |
| 17 | 9.34 TransportProperties..... | 48 |
| 18 | 9.35 ServiceLocator interface..... | 48 |
| 19 | 9.35.1.1 locate operation..... | 49 |
| 20 | 9.35.1.2 fallback attribute..... | 49 |
| 21 | 9.36 ForwardingServiceLocator..... | 49 |
| 22 | 9.37 RequestDecoder..... | 51 |
| 23 | 9.38 Necessary ORB modifications..... | 51 |
| 24 | 9.39 Providing LWFT to unmodified ORBs..... | 52 |
| 25 | | |

1 Preface

2 OMG

3 Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer
4 industry standards consortium that produces and maintains computer industry specifications for interoperable,
5 portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes
6 Information Technology vendors, end users, government agencies, and academia.

7 OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's
8 specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle
9 approach to enterprise integration that covers multiple operating systems, programming languages, middleware and
10 networking infrastructures, and software development environments. OMG's specifications include: UML®
11 (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common
12 Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

13 More information on the OMG is available at <http://www.omg.org/>.

14 OMG Specifications

15 As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications
16 Catalog is available from the OMG website at:

17 http://www.omg.org/technology/documents/spec_catalog.htm

18 Specifications within the Catalog are organized by the following categories:

19 Business Modeling Specifications

- 20 • Business Strategy, Business Rules and Business Process Management Specifications

21 Middleware Specifications

- 22 • CORBA/IIOP Specifications
- 23 • Minimum CORBA
- 24 • CORBA Component Model (CCM) Specification
- 25 • Data Distribution Service (DDS) Specifications

26 Specialized CORBA Specifications

- 27 • Includes CORBA/e and Realtime and Embedded Systems

1 **Language Mappings**

- 2 • IDL / Language Mapping Specifications
- 3 • Other Language Mapping Specifications

4 **Modeling and Metadata Specifications**

- 5 • UML®, MOF, XMI, and CWM Specifications
- 6 • UML Profiles

7 **Modernization Specifications**

- 8 • KDM

9 **Platform Independent Model (PIM), Platform Specific Model (PSM) and** 10 **Interface Specifications**

- 11 • OMG Domain Specifications
- 12 • CORBAservices Specifications
- 13 • CORBAfacilities Specifications
- 14 • OMG Embedded Intelligence Specifications
- 15 • OMG Security Specifications

16 All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing
17 OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and
18 PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management
19 Group, Inc. at:

20 OMG Headquarters
21 140 Kendrick Street
22 Building A, Suite 300
23 Needham, MA 02494
24 USA
25 Tel: +1-781-444-0404
26 Fax: +1-781-444-0320
27 Email: pubs@omg.org

28 Certain OMG specifications are also available as ISO standards. Please consult <http://www.iso.org>

29 **Typographical Conventions**

- 1 The type styles shown below are used in this document to distinguish programming statements from ordinary
- 2 English. However, these conventions are not used in tables or section headings where no distinction is necessary.
- 3 Times/Times New Roman - 10 pt.: Standard body text
- 4 **Arial - 9 pt. Bold:** OMG Interface Definition Language (OMG IDL) and syntax elements.
- 5 Arial – 9 pt: Examples
- 6 NOTE: Terms that appear in *italics* are defined in the glossary. *Italic* text also represents the name of a document,
- 7 specification, or other publication.

1 Scope

This specification extends the Fault Tolerance CORBA specification with capabilities to support real-time applications by providing predictable recovery times and support for application-defined consistency management.

Admitting that there is ‘no one size fits all’ solution for building highly fault tolerant solutions, applications can now provide their own fault detectors, fault analyzers and recovery mechanisms. Application state consistency will be decided and managed by application-supplied mechanisms that shall be integrated with FT infrastructures. The FT CORBA specification provided no means for applications to provide their own consistency management or to cooperate with middleware ReplicationManagement in order to synthesize a workable status for server groups.

Extensions to CORBA, FT CORBA are specified to enable interoperability between client ORBs and server ORBs in a fault tolerant enabled infrastructure.

2 Conformance

This specification defines 3 conformance levels:

- Server Replica Management:** Support for both cold-passive and warn-passive replication styles regardless of the interaction style (C/S, Pub/Sub).
- Basic Object Group Management:** Server Replica Management with support for client/server CORBA interactions based on standard IORs.
- Extended Object Group Management:** Basic Object Group Management with support for PortableGroups, FTCORBA IOGR's for enforcing at most once semantics, and support for Fault Detection and Notification.

An implementation claiming compliance to a given level shall also comply with lower levels (if any).

Table 1 provides a detailed service to conformance level matrix.

| Services/Interfaces | Server Replica Management | Basic Object Group Management | Extended Object Group Management |
|-----------------------|---------------------------|-------------------------------|----------------------------------|
| ServerManager | X | X | X |
| ServerRecoveryManager | X | X | X |
| RecoveryObserver | X | X | X |

| | | | |
|---------------------------------------|---|---|-----------|
| -ORBFTLocation | X | X | X |
| Object Group Name | | X | X |
| Object Group Name Alias | | | X |
| ClientFailoverSemanticsPolicy | | X | X |
| ObjectGroupMembershipPolicy | | X | X |
| ObjectGroupNameResolutionPolicy | | | X |
| ServiceLocator | | X | X |
| ForwardServiceLocator | | X | X |
| Interoperable Object Group References | | | X |
| FT_REQUEST ServiceContext | | | X |
| FT_GROUP_ID ServiceContext | | | X |
| FTRequestDurationPolicy | | | X |
| FT Current | | | X |
| ServerCallback | | X | X |
| ServerUpdateObserver | | X | X |
| ServerGroupManager | | X | X |
| ServerGroupManagerExt | | X | X |
| RecoverableServer | | X | X |
| Transport Heartbeats | | | X |
| FaultNotifier | | | X |
| ServerCrashFault | | | X |
| PullMonitorable | | | X |
| PullMonitorableServer | | | X |
| TCPProtocolProperties | | | If RT ORB |

| | | | |
|----------------------|--|---|---|
| -ORBKeepAlive | | X | X |
| GenericFactory | | | X |
| ObjectGroupManager | | | X |
| Group Object Adapter | | | X |

Table 1: Service to Conformance level matrix

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- [CORBA] Common Object Request Broker Architecture: Core Specification, March 2004 Version 3.0.3 (formal/04-03-01)
- [FTCORBA] Fault Tolerant CORBA - Chapter 23 from the CORBA Core Specification
- [CCM] CORBA Components, June 2002, Version 3.0 (formal/02-06-65)
- [DDS] Data Distribution Service for Real-Time Systems Specification, December 2005, Version 1.1 (formal/05-12-04)
- [RTCORBA] Real Time - CORBA Specification, November 2003, Version 2.0 (formal/03-11-01)
- [UMLPROFILE] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Version 1.0, (formal/06-05-02)
- [DATAPARALLEL] Data Parallel CORBA Specification, Version 1.0 (formal/06-01-03)
- [MIOP] Unreliable Multicast Inter-ORB Protocol Specification

4 Terms and Definitions

Fault Tolerance Domain

For scalability, large applications are divided into multiple fault tolerance domains each managed by a single ReplicationManager. All replicas of process groups and all the members of objects groups hosted by these processes are located within a single fault tolerance domain but can invoke, or can be invoked by, objects of other fault tolerance domains. A host can support objects from multiple fault tolerance domains.

FT-Enabled Middleware

1 FT-enabled middleware is responsible for providing the group abstraction needed for a reliable architecture
2 through middleware Replication Management and transparent failover.

3 The FT middleware may also rely on some middleware fault detectors for monitoring the middleware
4 infrastructure.

5 **FT Infrastructure**

6 The set of hardware and software components and services provided by the FT-enabled middleware and
7 external mechanisms in support for applications in a reliable and fault tolerant system. These include
8 replication management, fault management, consistency management and recovery management.

9 **Process Group**

10 A process group is a grouping of FT Process replicas managed by the ReplicationManager. Process groups
11 host members of object groups in order to ensure replication and failure transparencies to object group
12 users.

13 **FT Process**

14 A node-level POSIX process part of a process group (also referred to as replica) that is considered as a unit
15 of redundancy by the ReplicationManager.

16 **Object Group Manager**

17 The Object Group Manager is responsible for handling object group membership operation requests from
18 applications. It contains operations for creating a member of an object group at a particular location, adding
19 a member to an object group at a particular location, removing a member from an object group at a
20 particular location, getting the locations of the members of an object group.

21 **Forwarder**

22 A component of the FT-enabled middleware responsible for locating the latest object group reference and
23 'forwarding' it to the client. In order to act as a fallback object when all known object group members are
24 not reachable by a client, an object group reference must contain a profile to a Forwarder.

25 **Object Group Reference**

26 In order to provide replication transparency (client objects not aware that there are several server objects)
27 and failure transparency (client objects not aware of faults in the FT Process replicas), server object replicas
28 are managed as an object group.

29 An object group reference is an object reference for the object group as a whole.

30 An object group reference is standard Interoperable Object Group reference (IOR) that may or may not be
31 an Interoperable Object Group reference (IOGR).

32 **Server**

1 “A server is a computational context in which the implementation of an object exists. Generally, a server
2 corresponds to a process.” In LWFT a Server will always correspond to a Process

3 **Server ID**

4 “A Server ID must uniquely identify a server to an IMR. This specification only requires unique
5 identification using a string of some kind and does not intend to make more specific requirements for the
6 structure of a server ID. The server ID may be specified by an ORB_init argument of the form –
7 ORBServerId. The value assigned to this property is a string. All templates created in this ORB will return
8 this server ID in the server_id attribute. It is required that all ORBs in the same server share the same
9 server ID. Specific environments may choose to implement -ORBServerId in ways that automatically
10 enforce this requirement.”

11 **5 Symbols (and abbreviated terms)**

| | |
|-----------|---|
| AMSM | Application Management and System Monitoring for CMS Systems |
| CCM | CORBA Component Model |
| CIF | Component Implementation Framework |
| CIM | Common Information Model; a standard for system administration developed by DMTF |
| CORBA | Common Object Request Broker Architecture |
| DCE ESIOP | Environment-Specific Inter-ORB Protocol (ESIOP) for the OSF DCE environment |
| DCE-CIOP | DCE Common Inter-ORB Protocol |
| DCPS | Data-Centric Publish-Subscribe (part of DDS) |
| DDS | Data Distribution Service |
| DLRL | Data Local Reconstruction Layer (part of DDS) |
| DMTF | Distributed Management Task Force (cf. www.dmtf.org) |
| GIOP | General Inter-ORB Protocol |
| GOA | Group Object Adapter |
| IDL | Interface Definition Language |

| | |
|---------|---|
| IEEE | Institute of Electrical and Electronics Engineers |
| IMR | Implementation Repository |
| IOGR | Interoperable Object Group reference |
| IOR | Interoperable Object Reference |
| MIOP | Unreliable Multicast Inter-ORB Protocol |
| OMG | Object Management Group (cf. www.omg.org) |
| ORB | Object Request Broker |
| OSF | Open Software Foundation |
| PIM | Platform Independent Model |
| POA | Portable Object Adapter |
| POSIX | Portable Operating System Interface for Unix; name of a family of related standards specified by the IEEE |
| PSM | Platform Specific Model |
| QoS | Quality of Service |
| RFP | Request For Proposal |
| RO-MIOP | Reliable Ordered Multicast Inter-ORB Protocol |
| UML | Unified Modelling Language |
| W3C | World Wide Web Consortium (cf. www.w3c.org) |
| WBEM | Web-Based Enterprise Management |
| XML | eXtensible Mark-up Language |

6 Additional Information

6.1 Changes to Adopted OMG Specifications

6.1.1 Extensions

This specification adds the following to CORBA 3.1 specification:

- A new policy object: ClientFailoverSemanticsPolicy (§9.4)
- Two new POA Policies: ObjectGroupMembershipPolicy (§9.5), and ObjectGroupNameResolutionPolicy (§9.6)
- One Current object : FTCurrent (§9.25)
- Five new ObjectIds for resolve_initial_references: “FTCurrent” (see §9.25), “FTServiceLocator” (§9.35), “FTObjectGroupManager” (§9.31), “FTServerGroupManager” (§9.19), and “FTServerManager” (§9.24).
- Two new ORB Configuration parameters: -ORBKeepAlive (§9.34), and -ORBFTLocation (§9.24)
- One pseudo operation: FT_Locate (§9.36).

6.1.2 Changes

This specification adds the following to FT CORBA specification:

- Add support for location agents in Interoperable Object Group References (§9.10.1)
- A new service context: FT_GROUP_ID (§9.10.2)

6.2 Acknowledgments

The following companies submitted this specification:

- Thales
- PrismTech Group Ltd

7 Overall design rationale

The proposed design overcomes shortcomings and complexities of fault tolerant CORBA by introducing the Server Group concept (generally a POSIX process group) making a Server a unit of redundancy managed by an FT-enabled middleware while maintaining failure and replication transparencies for clients provided by Fault Tolerant CORBA object groups.

Figure 1: Overall design rationale

1
2
3 The consistency of server state is not managed by the FT-enabled middleware but is rather managed by middleware
4 independent mechanisms provided/supported by the application. The server state may thus rely on a persistent state service, a
5 replicated database, a Data Distribution Service ... etc.

6 Applications may now perform fault analysis, fault containment and recovery and also provide application-specific fault
7 detectors to complement the FT-middleware enabled fault detector.

8 To provide failure and replication transparencies to clients, the proposed design makes use of location forwarding techniques
9 to locate a suitable Server replica that is capable of dealing with their requests.

10 Servers register themselves with a central registry service (Server Group Manager) which maintains records of the endpoints
11 of all Server replicas. The reference advertised to clients contains endpoint information of one or more forwarding
12 components or a location service. When a Client attempts to use a Server's object reference it will therefore be directed to a
13 forwarding service locator instead, which will return the reference for a suitable Server replica that the Client should use. A
14 simplified diagram of this process can be seen below in Figure 2:

Figure 2: Diagram of the basic forwarding mechanism

7.1 General Architecture

15
16
17
18

19 **7.1 General Architecture**

20 CORBA Fault Tolerance architecture tackled most of the architectural elements required for a reliable architecture: entity
21 redundancy, fault detection, and recovery.

22 By externalizing fault detection and recovery to enable applications use more efficient mechanisms, the proposed architecture
23 focuses on Replication Management and Transparent Failover.

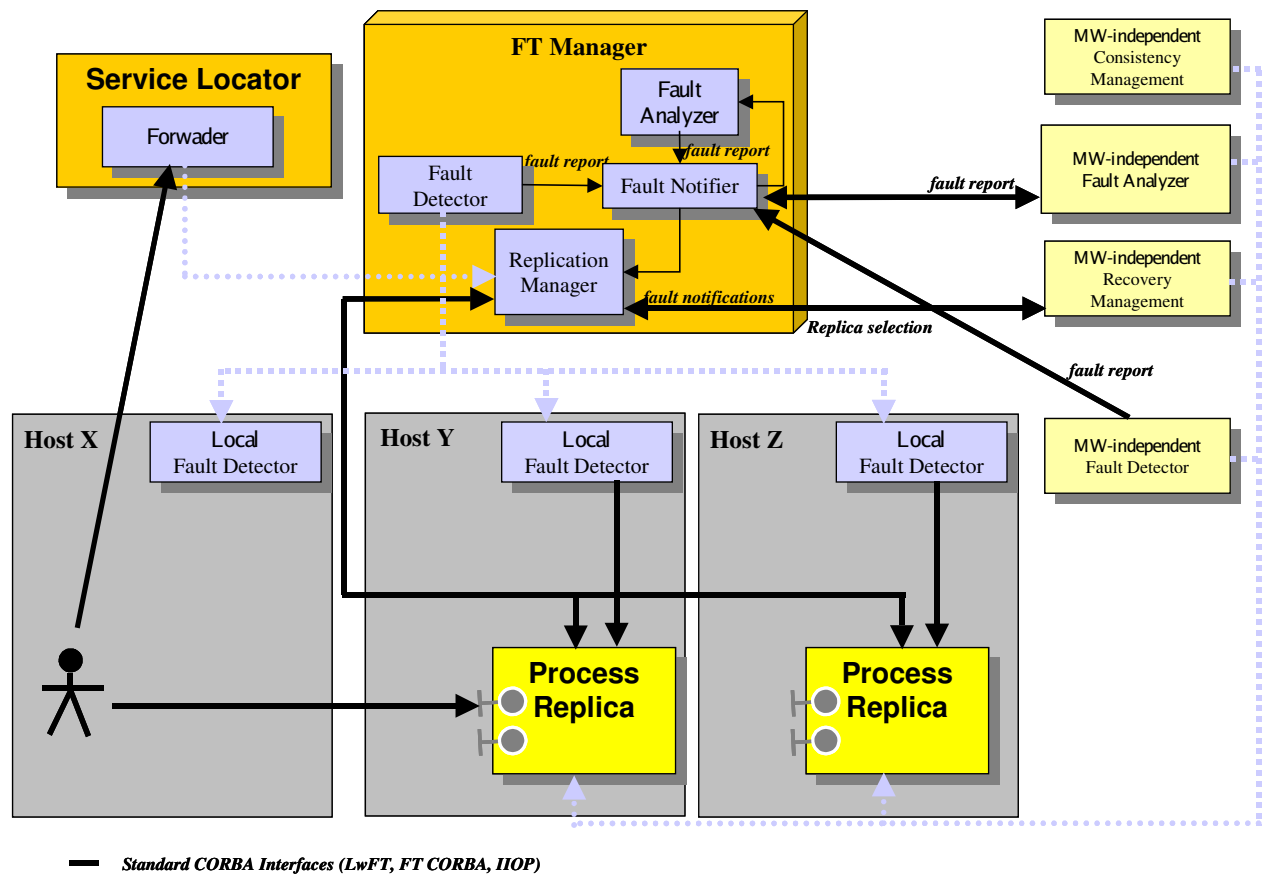


Figure 3: General Architecture

7.1.1 FT-enabled middleware

FT-enabled middleware is responsible for providing the group abstraction needed for our reliable architecture through middleware Replication Management and transparent failover. On primary replica failure, the middleware ReplicationManager may either elect a new one via some predefined strategies, or involve the middleware-independent FT infrastructure or the application for some specific/dedicated election strategy.

The FT middleware may also rely on some middleware fault detectors for monitoring the middleware infrastructure.

A Service Locator architectural component is also introduced to overcome a current FT CORBA limitation when all member replicas have failed (7.1.6).

7.1.2 Middleware-independent FT infrastructure

The FT-enabled middleware relies on external mechanisms for fault detection, fault analysis, and consistency and recovery management.

1 **7.1.3 Server/Process Replication Management Independent from Access to Service**

2 Server process replication done through the **ServerGroupManager** interface does not make any assumption on the way
3 application offer/use services (Client/Server, Data Publish/Subscribe ... etc.).

4 This specification focuses on the CORBA Client/Server interaction model.

5 **7.1.3.1 Client-side view / Object groups**

6 The **ClientFailoverSemanticsPolicy** provides a means for application developers to explicitly control how and if failover
7 occurs. This Policy will be effective in the context of any object reference.

8 Use of **object groups** when failure and replication transparencies are required based on standard forwarding mechanisms on
9 either standard IORs or Interoperable Object Group references (IOGRs).

10 **7.1.3.2 Server-side view / Process Groups**

11 The ServerGroupManager manages **process groups** in a fault tolerance domain. Since all objects and components within a
12 process group share computing resources, memory ... etc. the unit for ensuring consistency is the server process. In passive
13 replication, a server is either primary or backup and as a side effect, all hosted CORBA objects and components are assumed
14 to be primaries or backups.

15 At start-up, the server registers with the ServerGroupManager of its fault tolerance domain and joins its server group.
16 Middleware-independent mechanisms may be used for consistency management within the server but these should then
17 advise the ServerGroupManager of the completion/readiness of the server.

18 The ServerGroupManager then, advises the FT-enabled middleware/FT-ORB of the FT status of the server in order to
19 allow/deny request delivery to the server replica depending on whether it is primary or backup.

20 **7.1.4 Fault Tolerance Current**

21 It is necessary to retry request invocation in the presence of failure to provide failure transparency to applications with the
22 risk of performing a request more than once at the server side. Applications wishing to enforce the at most semantics of
23 CORBA invocation can use the **FTCurrent** object to retrieve invocation identification and detect repetitive request
24 invocations.

25 **7.1.5 Invocation Timeouts**

26 Invocation timeouts introduced by FTCORBA and CORBA Messaging can still be used with this specification
27 (RelativeRoundtripPolicy, RequestDurationPolicy ... etc.).

28 **7.1.6 Location Forwarding**

29 A Service Locator architectural component is responsible for locating/retrieving latest group object reference when all
30 previously known profiles have failed. This submission extends the Interoperable Object Group Reference to support location
31 mechanisms as in DCE-CIOP Location Mechanism ([CORBA] §16.6.1).

1 **7.1.7 Naming Groups**

2 To ease application bootstrapping and service localisation (Location/Forwarding) object groups are named using human
3 readable schemes.

4

8 Conceptual Model

This submission is based on proven concepts from the CORBA Fault Tolerance specification and basic forwarding techniques used in many ORB vendors' Implementation Repositories (IMR).

CORBA Fault Tolerance defines *fault tolerance domains* containing several hosts (nodes) and many object groups.

A reliable application within a fault tolerance domain will be made of many local applications running on physical nodes. An application offers some *services* for use by other applications or users. *ServiceAccessPoint* models the client view/access to the service (see AM SM, DMTF/CIM).

QosPolicy is used to capture main real time properties attached to the application or service access point.

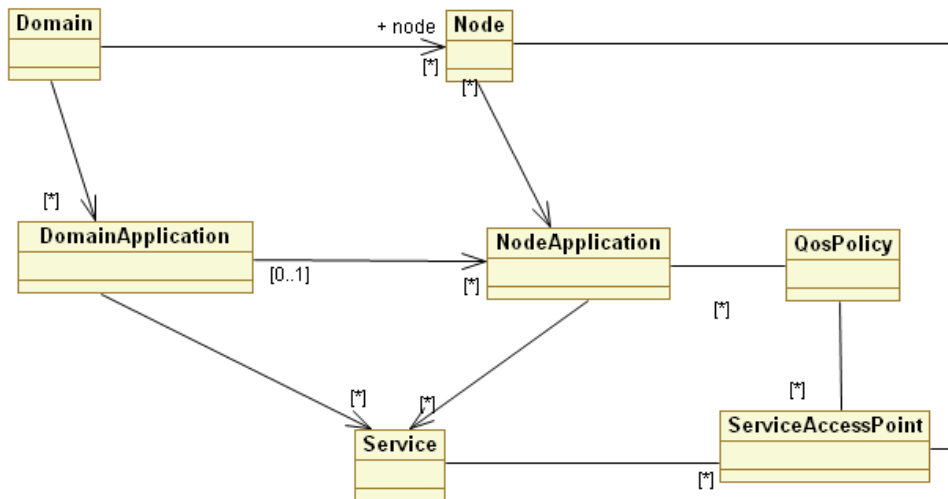
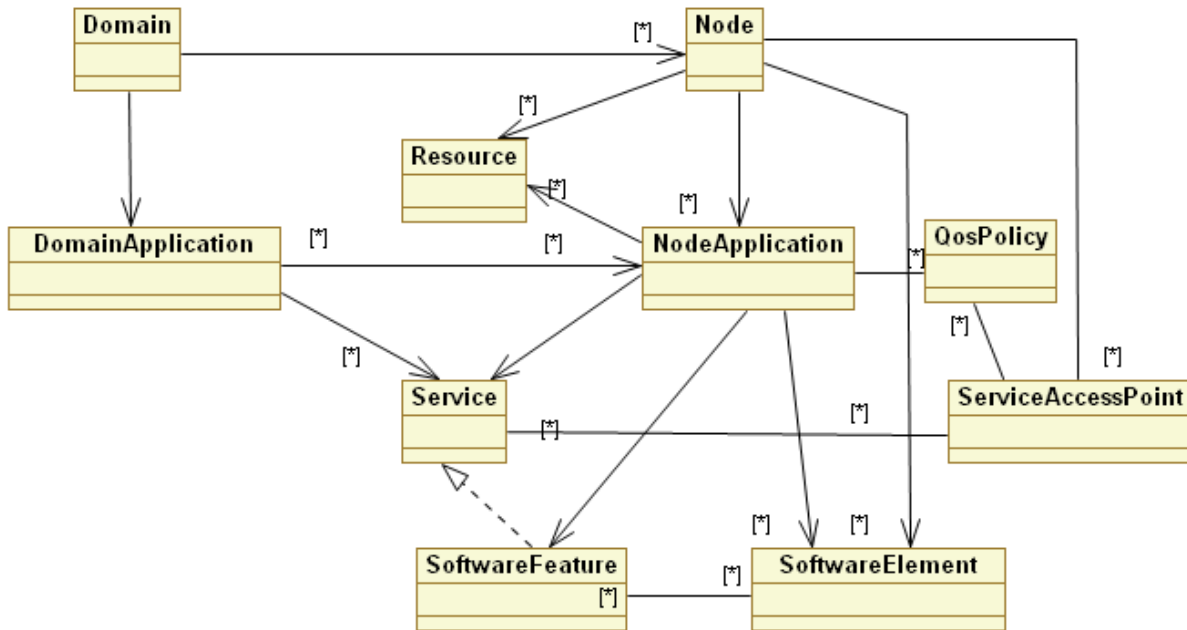


Figure 4: General overview of a Domain Application

The following figure (Figure 5) reuses concepts from DMTF CIM and models a local application in a platform independent way whether services are CORBA based, data centric or another. Only the *ServiceAccessPoint* is to be refined for the specific technology.



2 **Figure 5: General overview of a Node Application**

3 Class *SoftwareElement* (class CIM_SoftwareElement in DMTF/CIM) is a unit of deployment and represents a collection of
 4 more files are individually deployed and managed on a particular platform. This may be useful for tackling fault tolerance for
 5 CORBA Components.

6 Class *SoftwareFeature* (class CIM_SoftwareFeature in DMTF/CIM) is a unit of component management and represents a
 7 collection of software elements that perform a particular function (realizes one or more services).

8 **8.1 Fault Tolerance Domain**

9 CORBA Fault Tolerance [FTCORBA] introduced a fault tolerance domain for improving scalability of large applications
 10 where a single Replication Manager manages each fault tolerance domain.

11 **8.2 Redundancy Management**

12 This specification addresses replication at the POSIX process level. So the *ProcessGroup* concept models a group of process
 13 replicas (*FTProcess*). Each *FTProcess* runs on a Node and hosts local *FTApplication* local applications (see Figure 6).

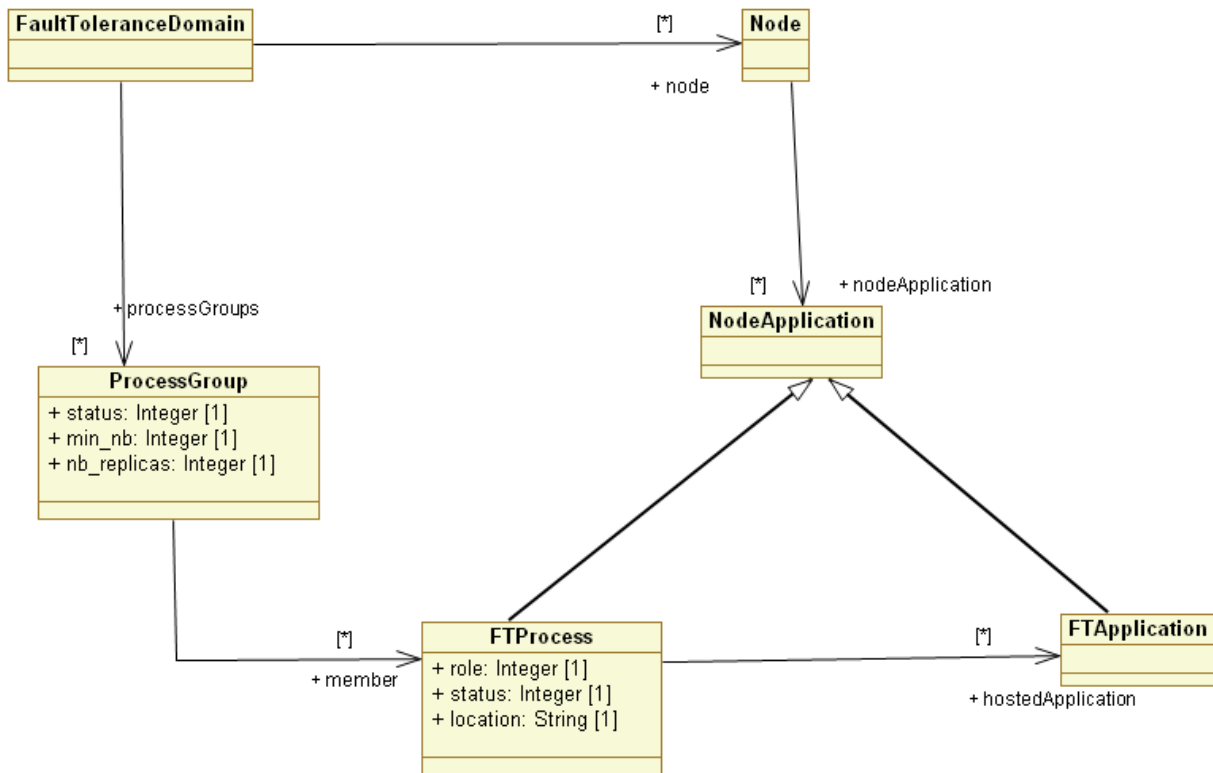


Figure 6: Process Groups and Replicas

2
3
4 A process group is a grouping of node-level POSIX process replicas. At Software Management level, *CIM_RedundancySet*
5 collection from DMTF/CIM can be used to model this process group concept.

6 8.2.1 Process Group

7 Class **ProcessGroup** has the following two attributes:

- 8 • *min_number_of_replicas*: The minimum number of FTProcess objects desired by the application.
- 9 • *nb_replicas*: the current number of FTProcess objects.

10
11 Class specialisation for Passive Replication may provide an additional:

- 12 • *primary_location*: Location of the primary FTProcess.

13
14 The ReplicationManager synthesizes the status of a ProcessGroup.

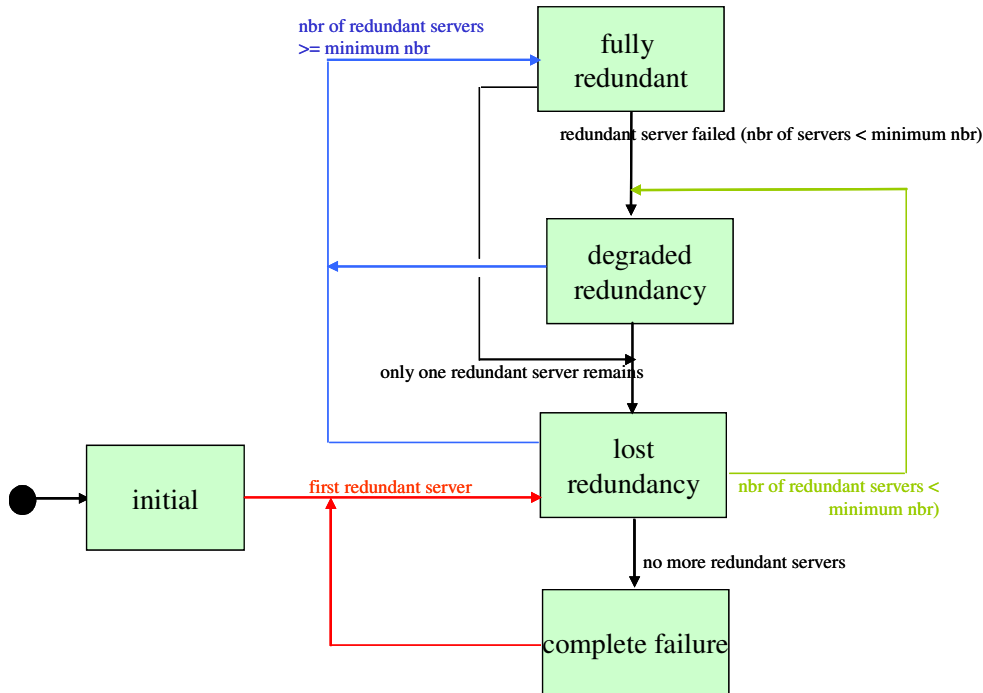
1 Figure 7: ProcessGroup status lists the status of a ProcessGroup depending on the current number of replicas and the
2 minimum number of replicas.

3 Example:

- 4 • **Fully Redundant** : number of replicas \geq minimum number of replicas
- 5 • **Degraded Redundancy** : $1 <$ number of replicas $<$ minimum number of replicas

6

7



9

Figure 7: ProcessGroup status

10 Monitoring of *ProcessGroup* objects and possible notifications on the transitions is left to Application Management
11 applications and is thus not in the scope of the FT-enabled middleware.

12 8.2.2 FTProcess (Process Replica)

13 The *FTProcess* class contains the following attribute:

- 14 • **location**: Location of the replica.

15

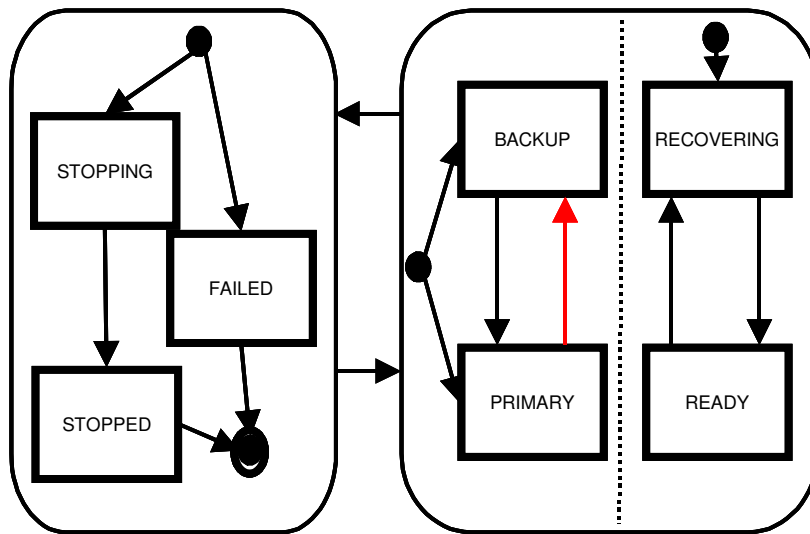
16 **FTProcess Status:**

- 1 • **RECOVERING:** The server process is performing its recovery/initialisation to get ready to service request in case of a
2 primary service, or else to become ready to take over should the primary fail in case of a backup server. It is not yet
3 ready to service requests so the FT-enabled middleware will not deliver to it group requests. Non-group requests such
4 as administrative requests will however be delivered to the server in order to allow it to perform any required
5 initializations.
- 6 • **READY:** The server process has now completed its recovery and it is ready to service requests in case of a primary
7 service, or else to become ready to take over should the primary fail in case of a backup server.

8 **FTProcess Replication Role:**

- 9 • **BACKUP:** The server process is either backup or recovering to become a backup. Once its status has become ready,
10 the server will be eligible to be become a primary once current primary server fails.
- 11 • **PRIMARY:** The server process is either primary or recovering to become a primary. Once its status has become ready,
12 the server will be serving group requests.
- 13 • **ACTIVE:** In the case of active replication, the server has an active role and will be allowed to service requests once
14 its status becomes ready.

15 Figure 8: FTProcess status in PassiveReplication provides the status of an FTProcess class in the case of PassiveReplication.



17 **Figure 8: FTProcess status in PassiveReplication**

18 The Process Replica registers with the Replication Manager in order to join its group and is told whether it shall be
19 initializing as primary or backup. At the end of its initialization/insertion, it shall advise the ReplicationManager of its
20 readiness to be allowed to receive requests on the fault tolerant group. See 8.2.4.1 for more details on backup insertion.

1 **8.2.3 Replication Management**

2 The ReplicationManager is an important component of the Fault Tolerance Infrastructure that interacts with other
3 components of the infrastructure. It is responsible for the creation of process groups, object groups for accessing application
4 services and the handling of all group membership requests.

5 The use of object group identifiers defined in CORBA Fault Tolerance allows for efficient retrieval of object group
6 information. But both process groups and object groups may be named for efficient initialization/bootstrap of middleware
7 independent FT infrastructure and clients and servers.

8 The FT Server registers with the Replication Manager in order to join its process group and then gets its replication role for
9 performing its recovery (initializing as primary or backup for example). At the end of its recovery, it shall advise the
10 ReplicationManager of its readiness to be allowed to receive requests for the fault tolerant object groups. See 8.2.4.1 for more
11 details on backup insertion.

12

13 **8.2.4 Passive Replication**

14 In the Warm-passive replication style, only the primary member is executing the methods invoked on the object group by the
15 client objects. The state of the primary is transferred to the backups to be ready for recovery from the primary replica failure.

16 **8.2.4.1 Backup Insertion**

17 Backup insertion refers to the start-up sequence of a replica and its acquisition of the existing primary replica's state in order
18 to be considered as a backup to the primary. Once a backup has been 'inserted', it is eligible for promotion to primary in case
19 of a primary replica failure.

20 The insertion of a replica may or may not disrupt the functioning of the primary.

21 The application or any consistency management shall advise the FT-middleware ReplicationManager of the completion of the
22 backup insertion/initialization.

23 Though not part of this specification, middleware independent consistency management mechanisms are advised to perform
24 the following:

- 25 • Failure of a primary replica while backups are inserting should trigger cancellation of their insertion.
- 26 • During a backup insertion, the state transfer should not start until the primary has completed its initialization.

27

28 **8.2.4.2 Recovery**

29 The Backup server is advised by the ReplicationManager of its promotion to Primary in order to start its recovery. Once all
30 state is recovered (responsibility of middleware-independent infrastructure) it shall advise the ReplicationManager of
31 recovery completion in order to generate Server transition to Primary and allow request deliveries.

1 **8.2.5 Active Replication**

2 Basic Redundancy Management is intended for both Passive and Active replication styles. However this specification does
3 not define some group communication protocol for accessing services offered by the FTApplications.

4 Only a TCP/IP based protocol suitable for passive replication only is addressed by this submission.

5 **8.3 Fault Management**

6 Fault Management provides the following functions:

- 7 • Fault detection: detecting the presence of a fault in the system and generating a fault report.
- 8 • Fault notification: propagating fault reports to entities that have registered for such notifications.
- 9 • Fault analysis/diagnosis: analyzing a (potentially large) number of related fault reports and generating condensed or
10 summary reports or any induced fault reports.
- 11 • Fault Containment: preventing the propagation of faults from their origin at one point in a system to a point where it
12 can have an effect on the service to the user.

13 **8.3.1 Fault Containment**

14 Middleware-independent FT infrastructure is responsible for enforcing fail-stop semantics by aborting/killing any server
15 process (FTProcess) that is suspected of being faulty.

16 Terminating any suspected process will prevent a process considered faulty from sending undesired messages.

17 **8.3.2 Fault Detection & Notification**

18 A FaultNotifier is provided by the FT-enabled middleware for notifying applications of any failures reported to it.

19 Server process may support a PullMonitorable interface to be used by local fault detectors (middleware independent or FT-
20 enabled middleware) for monitoring the state of the server replica.

21 Fault Reports conveyed to the Fault Notifier by the Fault Detectors shall conform to CosNotification::StructuredEvent with
22 event type as specified in CORBA Fault Tolerance specification.

23 **8.3.2.1 FaultNotifier**

24 The FaultNotifier [FTCORBA] defines operations for use by the fault detectors and fault analysers to push fault reports for
25 the FaultNotifier. The FaultNotifier propagates such fault reports to registered entities such the ReplicationManager, and any
26 other fault analyser or application objects.

27 **8.3.3 Fault Analysis/Diagnosis**

28 Once a server process (FTProcess) has failed, all embedded/hosted applications shall be considered faulty.

1 **8.3.4 Connection Management**

2 Support for transport heartbeats may be provided by the FT-enabled middleware to close connections to failed server hosts or
3 with failed communication links.

4 **8.4 QoS Policies**

5 To enforce some real-time properties a set of QoS policies are needed. These apply to different steps/phases of the
6 application.

7 **8.4.1 FTProcess Recovery time**

8 FTProcess recovery (for both Passive and Active replication styles) may only be achieved within this time limit. A failure to
9 comply with this time duration may be reported to applications by the middleware independent FT infrastructure.

10 It is the responsibility of middleware independent fault analyzer to decide whether this is an application failure or not, in
11 which case it should enforce the crash/stop model by terminating the FTProcess.

12

13 **8.4.2 Maximum Service Localization Time**

14 Locating the service shall not exceed this time duration. FT-enabled middleware shall abort localization of the service when
15 this time duration is exceeded.

16 **8.4.3 Service Access Properties**

17 Some QoS properties may be needed for accessing the service (such as CORBA priorities in RT CORBA).

18 **8.4.4 Invocation Maximum Blocking Time**

19 A client accessing the service cannot be blocked for more than the maximum blocking time.

20 **8.4.5 Transport/Connection Timeouts**

21 Transport-level timeouts for (heartbeating, keepalive) may be provided by the FT-enabled middleware to close connections to
22 failed server hosts or with failed communication links.

23 **8.5 Transparent Failover**

24 Failover semantics as defined in the CORBA Fault Tolerance specification proved very efficient and workable. This
25 specification relies on currently defined semantics from [FTCORBA].

1 The Service Locator (cf. 8.5.3) is responsible for supplying the latest working object reference.

2 **8.5.1 At Most Once Semantics**

3 Enforcing at most once semantics by the means of request and reply logging defined in CORBA Fault Tolerance specification
4 is not mandatory for the FT-enabled middleware as applications have better knowledge of operation semantics and can thus
5 handle this in an efficient manner.

6 Should a unique request identifier be required by the application in order to detect repetitive requests, the FT Context object
7 (cf. 8.5.2) can provide the necessary information.

8 **8.5.2 FT Context**

9 The FT Context object provides access to invocation information for applications requiring this knowledge for consistency
10 management or enforcing at most once semantics.

11 **8.5.3 Service Locator**

12 The Service Locator is responsible for providing access to replicated objects/components when direct connection between
13 client and server processes is broken. Depending on the kind of reference is used by the FT-enabled middleware; the Service
14 Locator may also be used at startup to open initial connections between client and server processes.

15 Implementers of the Service Locator are free to access the ReplicationManager, and run any implementation-specific
16 protocol as long as they can assure a predictable time for providing a valid reference.

17 **9 OMG CORBA/IDL Platform Specific Model**

18 **9.1 Server Group**

19 A **ServerId**, in the context of LWFT identifies a **ServerGroup**. All servers with a common ServerId are replicas of each
20 other and members of the same ServerGroup.

```
21  
22 module LWFT  
23 {  
24     // PortableInterceptor::ServerId is a string  
25     typedef PortableInterceptor::ServerId ServerId;  
26 };
```

27

28 Server ID Is defined in the CORBA specification section **8.5.1.1 (08-01-04) Server ID**. **This definition is reproduced**
29 **below:**

1 “A Server ID must uniquely identify a server to an IMR. This specification only requires unique identification using a string
2 of some kind and does not intend to make more specific requirements for the structure of a server ID. The server ID may be
3 specified by an **ORB_init** argument of the form **-ORBServerId**. The value assigned to this property is a string. All templates
4 created in this ORB will return this server ID in the server_id attribute. It is required that all ORBs in the same server share
5 the same server ID. Specific environments may choose to implement -ORBServerId in ways that automatically enforce this
6 requirement.”

7 Section **15.2.1 Model Components** of the CORBA specification (**08-01-04**) contains the following definition:
8 “*Server*—A server is a computational context in which the implementation of an object exists. Generally, a server
9 corresponds to a process.” **In LWFT a Server will always correspond to a Process.**

10 **9.2 Server Replica Identity**

11 As in the Fault Tolerant CORBA specification, the Location is used to identify a process instance within a ServerGroup and
12 globally. The FT CORBA specification defines a FT::Location as follows:

13 “*The name for a fault containment region, host, device, cluster of hosts, etc., which may be hierarchical.... For each object*
14 *group and each location, only one member of that object group may exist at that location.*”

15 However, in LWFT Server process based replication following additional constraint applies:

16 Each LWFT server process must have only one Location and the Location must uniquely identify the server process. For
17 example, a LWFT server process Location might well include both the DNS host name or IP address and the process id. This
18 way the Location will uniquely identify the server process instance within the fault tolerance domain.

```
19  
20 module LWFT  
21 {  
22     typedef PortableGroup::Name Location;  
23 };
```

24 **9.2.1 Server Status**

```
25  
26 typedef long ServerStatusValue;  
27 const ServerStatusValue SRV_STATUS_UNKNOWN = 0;
```

28 **9.2.1.1 SRV_STATUS_RECOVERING**

```
29  
30 const ServerStatusValue SRV_STATUS_RECOVERING = 1;
```

31
32 The server process is performing its recovery/initialisation to get ready to service requests in the case that it is a primary
33 service, or else to become ready to take over should the primary fail in case where it is a backup server.

1 The server is not yet ready to service requests so the FT-enabled middleware will not deliver group requests to it.
2 Non-group requests such as administrative requests will however be delivered to the server in order to allow it to perform any
3 required initialisations.

4 **9.2.1.2 SRV_STATUS_READY**

```
5  
6     const ServerStatusValue SRV_STATUS_READY = 2;
```

7
8 The server process has now completed its recovery and it is ready to service requests in the case that it is a primary service,
9 or else to become ready to take over should the primary fail in case that it is a backup server.

10 **9.2.2 Server Replication Role**

```
11  
12     typedef long ServerReplicationRoleValue;  
13     const ServerReplicationRoleValue SRV_REPL_UNKNOWN = 0;
```

15 **9.2.2.1 SRV_REPL_PRIMARY**

```
16  
17     const ServerReplicationRoleValue SRV_REPL_PRIMARY = 1;
```

18
19 The server process is either primary or recovering to become a primary.
20 Once its status has become ready, the server will be serving group requests.

21 **9.2.2.2 SRV_REPL_BACKUP**

```
22  
23     const ServerReplicationRoleValue SRV_REPL_BACKUP = 2;
```

24
25 The server process is either backup or recovering to become a backup.
26 Once its status has become ready, the server will be eligible to become a primary once current primary server fails.

27 **9.2.2.3 SRV_REPL_ACTIVE**

28

1 **const ServerReplicationRoleValue SRV_REPL_ACTIVE = 3;**

2

3 In the case of active replication, the server has an active role and will be allowed to service requests once its status becomes
4 ready.

5 **9.3 Naming Object Groups**

6 The **ObjectGroupName** is a hierarchic **CosNaming::Name**. Name Components within the name must be ordered according
7 to the following schema:

8 **<server_id>.SRV/<orb_id>.ORB/<poa_id>.POA[<sub_poa_id>.POA*]/<object_id>.OBJ**

9 where poa_id is the first named POA below RootPOA and sub_poa values are in order of occurrence beneath that POA.

10 A shorthand notation may be supported when there is no ambiguity:

11 **<server_id>/<object_id>**

12 **9.3.1 Support for Aliases**

13 Applications should be able to provide aliases for object group names to allow for more application-oriented naming
14 schemes. The object group name (alias) is unique within the context of the Fault Tolerance domain.

15 One way to define this alias when the object group is created via a **PortableGroup::GenericFactory** interface is to define a
16 new ft property à la '**org.omg.ft.alias**' to be provided in the criteria parameter of **create_object** operation. The following
17 default alias should be made available within the fault tolerance domain with an **ObjectGroupManager**:

18 **<object_group_id>.GID**

19 Administrative tools may also be used for naming the object groups via implementation-specific interfaces.

20

```
21     module LWFT {  
22         const string FT_ALIAS = "org.omg.ft.Alias";  
23     };
```

24

25 **9.4 ClientFailoverSemanticsPolicy**

26 The **ClientFailoverSemanticsPolicy** provides a means for application developers to explicitly control how and if failover
27 occurs. This Policy will be effective in the context of any object reference.

28

```

1  module LWFT {
2
3      enum ClientFailoverSemanticsPolicyValue {
4          ORB_DEFAULT,
5          BEST_EFFORT,
6          AT_LEAST_ONCE,
7          AT_MOST_ONCE
8      };
9      local interface ClientFailoverSemanticsPolicy : CORBA::Policy {
10         readonly attribute ClientFailoverSemanticsPolicyValue value;
11     };
12 };

```

13

14 The four values of this Policy modify the client ORB behaviour as follows:

- 15 • **ORB_DEFAULT** - The ORB behaviour is unmodified from its default.
- 16 • **BEST_EFFORT** - This value of the Policy provides the client application with a guaranteed 'best effort' delivery
17 attempt. Normal CORBA invocation semantics do not require that a compliant ORB **must** reinvoke requests on
18 COMPLETED_NO retryable exception conditions or exhaust all available addresses in an IOR. Compliant client
19 ORBs must provide the following behaviour under the influence of this Policy value: Each time a client ORB attempts
20 to invoke a method, it must not abandon the invocation and raise an exception to the client application until it has tried
21 to invoke the server using all of the alternative addresses in the IOR, or has received a “non-failover” condition, or the
22 request duration has expired.
- 23 • **AT_LEAST_ONCE** - This value behaves as BEST_EFFORT however retryable exception conditions with status
24 COMPLETED_MAYBE must also be reinvoked.
- 25 • **AT_MOST_ONCE** - This value behaves as AT_LEAST_ONCE but the client ORB must transmit an FT_REQUEST
26 service context (*FTRequestServiceContext*) with the request.

27 9.5 ObjectGroupMembershipPolicy

28 The **ObjectGroupMembershipPolicy** POA policy is to be used on the server side to control object group management in a
29 portable way.

30 Application developers can specify this policy on POA creation to indicate to the FT enabled ORB that Objects created on
31 that POA should automatically be members of replicated Object Groups and that the references returned from POA
32 operations should be ObjectGroup references.

33

```

1  module LWFT {
2
3      enum ObjectGroupMembershipPolicyValue
4      {
5          AUTO_ORB_CTRL,
6          AUTO_OBJ_GRP_MNGR,
7          USER_CTRL_OBJ_GRP_MNGR
8      };
9
10     local interface ObjectGroupMembershipPolicy : CORBA::Policy
11     {
12         readonly attribute ObjectGroupMembershipPolicyValue value;
13     };
14 };
15

```

16 The three ordered values of ObjectGroupMembershipPolicy represent increasing levels of user control over the mechanisms
17 used to provide fault tolerance and corresponding decreasing user application transparency.

18 **ObjectGroupMembershipPolicy** values are discussed in the following sections.

19 **9.5.1 AUTO_ORB_CTRL**

20 The objects references created on the POA will be automatically FT replicated across all similar server processes (i.e. that
21 have a matching FT POA hierarchy). Every object reference created on such a POA will be made a member of the
22 ObjectGroup named as per the naming scheme defined in Section 2.4.3 Naming Object Groups. i.e.:

23 `<server_id>.SRV/<orb_id>.ORB/<poa_id>.POA[/<sub_poa_id>.POA*]/<object_id>.OBJ`

24 The behaviour from the users perspective is ' **AUTO_OBJ_GRP_MNGR** (see below) but the mechanism used by the server
25 side ORB to ensure this is left to the ORB implementation.

26 **N.B.:** This policy value is only valid on POAs with LifespanPolicy 'PERSISTENT' and IdAssignmentPolicy 'USER_ID'.

27 All POA operations that create object references will return references that refer to the whole object group rather than to this
28 individual member.

29 **9.5.2 AUTO_OBJ_GRP_MNGR**

30 The objects created on the POA will be automatically FT replicated as above and as per the behaviour below. The POA will
31 transparently use the relevant ObjectGroupManager operations to ensure that this is achieved. At the point that a reference is
32 created on a POA with this Policy value (and before the reference is returned to the user code) the POA should make use of
33 appropriate GOA / ObjectGroupManager operations to:

- 34 1. Get the group to which this object should belong
- 35 2. Add the member to it.

1 3. Return the new IOGR to the caller of the POA operation.

2 **N.B.:** Only valid on POAs with IdAssignmentPolicy USER_ID. All POA operations that create object references will return
3 references that refer to the whole object group rather than to this individual member.

4 To retrieve the object group reference, compliant POAs should first use the object group ServiceLocator, and if no
5 ServiceLocator is present, it shall attempt to use the NameService.

6 Compliant ORBs / FT implementations should make the ServiceLocator available to the POA by
7 **ORB::resolve_initial_references ("FTServiceLocator").**

8 **9.5.3 USER_CTRL_OBJ_GRP_MNGR**

9 The POA created with this policy will be a GOA. This will make the GOA object group management operations available to
10 the application developer. The interaction with the FT infrastructure will be via the ObjectGroupManager as per the already
11 existing FT CORBA spec.

12 Compliant ORBs / FT implementations should make the ObjectGroupManager available to the Object Adapter by
13 **ORB::resolve_initial_references ("FTObjectGroupManager").**

14 **9.6 ObjectGroupNameResolutionPolicy**

15 The **ObjectGroupNameResolutionPolicy** POA policy is needed in conjunction with AUTO_OBJ_GRP_MNGR value of
16 ObjectGroupMembershipPolicy policy to select a fully qualified naming scheme or a shorthand notation.

```
17  
18    module LWFT {  
19       enum ObjectGroupNameResolutionPolicyValue  
20       {  
21            USE_FULLY_QUALIFIED_NAME,  
22            USE_SHORTHAND_NOTATION  
23       };  
24       local interface ObjectGroupNameResolutionPolicy : CORBA::Policy  
25       {  
26            readonly attribute ObjectGroupNameResolutionPolicyValue value;  
27       };  
28    };  
29
```

30 In the absence of a specified Policy value being defined on the POA the LWFT ORB will assume a default of
31 **USE_FULLY_QUALIFIED_NAME.**

32 **9.7 FT_REQUEST Service Context**

33 The FT_REQUEST service context (FTRequestServiceContext) is defined in CORBA Fault Tolerance to ensure that a request
34 is not executed more than once under fault conditions.

1 When **AT_MOST_ONCE** is used for the **ClientFailoverSemanticsPolicy**, the FT-enabled middleware shall encode a CDR
2 encapsulation of the **FTRequestServiceContext** struct in the **context_data** of the **ServiceContext** struct of a request message
3 header.

```
4  
5     module IOP {  
6         const Serviced FT_REQUEST = 13;  
7     };  
8     module FT {  
9         struct FTRequestServiceContext { // context_id = FT_REQUEST;  
10            string client_id;  
11            long retention_id;  
12            TimeBase::TimeT expiration_time;  
13        };  
14    };
```

15 The **FT_REQUEST** service context contains a unique **client_id** for the client, a **retention_id** for the request, and an
16 **expiration_time** for the request. The *client_id* and *retention_id* serve as a unique identifier for the client's request and allow
17 the server application to recognize that the request is a repetition of a previous request. A server application willing to enforce
18 at most once semantics on its incoming requests may choose to return a previously generated reply in case of repetition of
19 previous requests.

20 The *expiration_time* serves as a garbage collection mechanism. It provides a lower bound on the time until which the server
21 application must honor the request and, therefore, retain the request and corresponding reply (if any) in its log.

22 9.8 FTRequestDurationPolicy

23 The Request Duration Policy is defined in CORBA Fault Tolerance. It determines how long a request, and the corresponding
24 reply, should be retained by a server to handle reinvocation of the request under fault conditions.

```
25  
26     module FT {  
27         const CORBA::PolicyType REQUEST_DURATION_POLICY = 47;  
28         interface RequestDurationPolicy : CORBA::Policy {  
29             readonly attribute TimeBase::TimeT request_duration_policy_value;  
30         };  
31     };
```

32 The Request Duration Policy, applied at the client, defines the time interval over which a client's request to a server remains
33 valid and must be retained by the server application to detect repeated requests.

34 The *request_duration_policy_value* is added to the client ORB's current clock value to obtain the *expiration_time* that is
35 included in the **FT_REQUEST** service context for the request.

9.9 Transport Heartbeats

'23.2.9 Transport Heartbeats' of the CORBA specification defines TAG_FT_HEARTBEAT_ENABLED component of the TAG_INTERNET_IOP profile, and two policies: Heartbeat and HeartbeatEnabled to solve the problem of host and/or link failures in IOP (TCP/IP).

This specification reuses those mechanisms. Fully compliant Server ORBs should be able to respond to the FT_HB pseudo operation as defined in that section. In the case of Server ORBs that are not compliant, users may choose to implement this operation at the application level. Alternatively, components of the LWFT middleware can identify Servers that are not fully compliant by a BAD_OPERATION.

9.9.1 TAG_FT_HEARTBEAT_ENABLED Component

“The TAG_FT_HEARTBEAT_ENABLED component in a TAG_INTERNET_IOP profile indicates that the addressed endpoint supports heartbeating.”

```
module IOP {
    const ComponentId TAG_FT_HEARTBEAT_ENABLED = 29;
};
module FT {
    struct TagFTHeartbeatEnabledTaggedComponent {
        // tag =TAG_FT_HEARTBEAT_ENABLED
        boolean heartbeat_enabled;
    };
};
```

9.9.2 Heartbeat Policy

“The Heartbeat Policy, applied at the client, allows the client to request heartbeating of its connections to servers, using the heartbeat_interval and heartbeat_timeout.”

```
module FT {
    const CORBA::PolicyType HEARTBEAT_POLICY = 48;
    struct HeartbeatPolicyValue {
        boolean heartbeat;
        TimeBase::TimeT heartbeat_interval;
        TimeBase::TimeT heartbeat_timeout;
    };
    interface HeartbeatPolicy : CORBA::Policy {
        readonly attribute HeartbeatPolicyValue heartbeat_policy_value;
    };
};
```

“When the Heartbeat Policy is applied at a client ORB, the ORB is responsible for taking the following steps. While a

1 *connection exists to a remote server, the ORB sends a request message over the connection at least as often as was requested*
2 *by the heartbeat_interval of the Heartbeat Policy of any client connected to a server over that connection. The request*
3 *message is equivalent to an invocation of the method:*

4
5 **void FT_HB ();**

6 *on any one of the server objects accessed by the connection...*

8 **9.9.3 Heartbeat Enabled Policy**

9 *“Because heartbeating can generate significant network traffic, and can use significant server resources, the heartbeating*
10 *capability is explicitly enabled or disabled using the Heartbeat Enabled Policy.”*

11
12 **module FT {**
13 **const CORBA::PolicyType HEARTBEAT_ENABLED_POLICY = 49;**
14 **interface HeartbeatEnabledPolicy : Policy {**
15 **readonly attribute boolean heartbeat_enabled_policy_value;**
16 **};**
17 **};**

18 *“The Heartbeat Enabled Policy allows the heartbeating of a server endpoint. If the Heartbeat Enabled Policy is enabled for a*
19 *server endpoint, the TAG_INTERNET_IOP profile for that endpoint contains the TAG_FT_HEARTBEAT_ENABLED*
20 *component to indicate to the client that the server endpoint is heartbeat_enabled.”*

21 **9.10 Interoperable Object Group References**

22 The object group abstraction is very powerful and allows for the provision of both failure and replication transparencies for
23 clients. This specification extends the existing FT CORBA definition of Interoperable Object Groups with the support for
24 Location agents.

25 Use of FT_REQUEST service context allows applications that require enforcing at most once semantics to access (via FT
26 Current interface) request information.

27 Use of FT_GROUP_VERSION service context allows a server to determine whether a client is using an obsolete object
28 group reference and, if so, to respond with the most recent object group reference to update the client’s object reference.

29 This specification introduces a new FT_GROUP_ID service context to allow for more efficient lookup of latest IOGR in case
30 of requests with older versions coming to the process as the object group identifier may not necessarily be obtained from the
31 object key. This avoids comparison of IORs as it is more likely to have more than one object group with a local member.

32 **9.10.1 Support for Location Agents.**

33 Current FT CORBA states the following in §23.2.4 Modes of Profile Addressing:

1 “The interoperable object group references contain profiles that address server object groups. This section illustrates the use
2 of these profiles according to one of two modes:

- 3 • Profiles that address object group members.
- 4 • Profiles that address gateways (technically generic in-line bridges of the type described in the Building Inter-ORB
5 Bridges chapter of the CORBA specification).”

6 To allow for the support of Location services as in FT CORBA §16.6.1 Location Mechanism Overview, the following shall be
7 added:

- 8 • Profiles for agents that may not be able to accept direct requests for any objects, but acts instead as a location service.
9 Any request messages sent to the agent would result in either exceptions or replies with LOCATION_FORWARD
10 status, providing new addresses to which requests may be sent. Such agents would also respond to locate request
11 messages with appropriate locate response messages.
- 12 • Profiles for agents that may directly respond to some requests (for certain objects) and provide forwarding locations
13 for other objects.

14 9.10.2 FT_GROUP_ID Service Context

15 The FTGroupIdServiceContext struct contains the version of the identifier of the server object group, to assist the server in
16 getting the object group identifier when not encoded within the object key. When encoded in a request or reply message
17 header, the context_data component of the ServiceContext struct shall contain a CDR encapsulation of the
18 FTGroupIdServiceContext struct, which is defined below.

```
19  
20 module IOP {  
21     const Serviced FT_GROUP_ID = 17;  
22 };  
23 module FT {  
24     struct FTGroupIdServiceContext { //context_id = FT_GROUP_ID;  
25         ObjectGroupId object_group_id;  
26     };  
27 };
```

28 If the server determines from the FT_GROUP_VERSION service context that the client is using an obsolete object group
29 reference, the server may use the object_group_id value to get the most recent object group reference for the server group and
30 then return it in a LOCATION_FORWARD_PERM response.

```
31     const Serviced FT_GROUP_ID = 17;
```

32 A constant that designates the FT_GROUP_ID service context.

```
33  
34 struct FTGroupIdServiceContext { //context_id = FT_GROUP_ID;  
35     ObjectGroupId object_group_id;  
36 };  
37
```

38 A structure that contains the same object_group_id that is in the TAG_FT_GROUP component of each of the

1 TAG_INTERNET_IOP profiles of the object group reference for the server object group, to be used, if necessary, by the
2 server ORB to obtain the most recent object group reference of the server object group.

3 A server ORB that is capable of obtaining the the most recent object group reference of the server object group without the
4 help of the FT_GROUP_ID service context should ignore this context. On the other hand, should the FT_GROUP_ID service
5 context be required by the server ORB but not present in the client's request, the server ORB shall return an INV_OBJREF
6 exception to the client.

7 **9.10.3 Persistent vs. Transient IOR**

8 Depending on the technique used by the FT-enabled middleware for providing middleware transparent failover (between the
9 ReplicationManager and its Gateways), server-side FT-enabled middleware may choose to create transient references only to
10 avoid backup replicas (previously started as primary then restarted) receiving client requests in case that only a primary
11 object reference is used within the object group reference.

12 Object Group membership operations initiated by the Group Object Adapter (§9.31) will rely on the LifeSpan policy used at
13 the object adapter's creation.

14 **9.10.4 Gateway**

15 CORBA Fault Tolerance specification defines a Gateway for non-FT aware ORBs. This gateway may be extended to provide
16 Service Location capabilities as defined in 8.5.3.

17 The gateway may rely on LOCATION FORWARDING techniques such as those used in IMRs.

18 Within a Fault Tolerance domain (managed by a single Replication Manager), the Gateway shall not make any assumption
19 about the client's ORB's ability to assure portability and keep any constraint that may be imposed by the FT-enabled
20 middleware within the 'Server-scope'.

21 **9.11 GenericFactory**

22 FT-enabled middleware relying on **PortableGroup::GenericFactory** interface for the creation of object groups may support
23 naming (also named aliasing in this specification) of object groups in order to ease bootstrapping to applications.

24

```

1  module PortableGroup
2  {
3      interface GenericFactory
4      {
5          typedef any FactoryCreationId;
6          Object create_object
7              (in _TypeId type_id,
8               in Criteria the_criteria,
9               out FactoryCreationId factory_creation_id)
10             raises (NoFactory, ObjectNotCreated, InvalidCriteria,
11                   InvalidProperty, CannotMeetCriteria);
12         void delete_object
13             (in FactoryCreationId factory_creation_id)
14             raises (ObjectNotFound);
15     }; // end GenericFactory
16 };

```

17 Add a new criteria for use by the **GenericFactory::create_object()** operation to hold the Alias property.

18 9.11.1 Alias property

19 **FT_ALIAS** property name = *'org.omg.ft.Alias'*. When no Alias property is provided at the object group creation a default
20 name will be supplied by the **GenericFactory** : “<<object group id>>.gid”.

21 9.12 ServerCallback interface

```

22
23 interface ServerCallback {};
24

```

25 The **ServerCallback** interface is the base for specific callback specialisations. Servers can pass callback references to the FT
26 Middleware to enable the middleware, and actors external to the middleware which are observing the ServerGroup, to interact
27 with the individual ServerGroup process members.

28 9.13 GroupUpdateObserver interface

29 **GroupUpdateObserver** is a specialisation of **ServerCallback** interface for use when it is necessary to be updated whenever
30 the list of process group members changes. Each process group entry contains the location of the member, its callback object
31 and the list of process-specific properties passed to the **ServerGroupAdmin** object during the process registration.

32

```

1  struct ServerGroupMember
2  {
3      Location      server_location;
4      ServerCallback server_callback;
5      Properties    props;
6  };
7  typedef sequence<ServerGroupMember> ServerGroupMembers;
8  interface GroupUpdateObserver : ServerCallback
9  {
10     void on_update_group(in ServerGroupMembers members);
11 };
12

```

13 9.13.1 on_update_group operation

```

14
15     void on_update_group(in ServerGroupMembers members);
16

```

17 The ServerGroupManager (or other part of the FT infrastructure) can notify the process of a change of the list of process
18 group members.

19 9.14 ServerGroupNotFound exception

```

20
21     exception ServerGroupNotFound {};
22

```

23 The server group with the given identifier is not found by the ServerGroupManager. This exception may be raised if the
24 ServerGroupManager can only register known ServerGroups.

25 9.15 ServerGroupNotFound exception

```

26
27     exception ServerNotFound {};
28

```

29 No server process is known by the ServerGroupManager exists for the given location value.

1 **9.16 WrongStatus exception**

2

3 **exception WrongStatus {};**

4

5 Operation incompatible with current process status.

6 **9.17 AlreadyRegistered exception**

7

8 **exception AlreadyRegistered {};**

9

10 The Server Location value has already been registered..

11 **9.18 UnsupportedCallback exception**

12

13 **exception UnsupportedCallback {};**

14

15 The ServerGroupManager does not support provided ServerCallback type.

16 **9.19 Server Group Manager**

17 Compliant FT implementations should register an instance to be accessible to the server ORB via

18 **ORB::resolve_initial_references ("FTServerGroupManager").**

19


```

1  interface ServerGroupManager
2  {
3      void register_server(inout Location  the_location,
4                          in ServerId    the_server_id,
5                          in Properties   props,
6                          in ServerCallback callback)
7          raises (ServerGroupNotFound, AlreadyRegistered,
8                UnsupportedCallback);
9
10     void unregister_server(in Location location)
11         raises (ServerNotFound);
12
13     void server_ready(in Location the_location)
14         raises (ServerNotFound,
15               WrongStatus);
16 };

```

17

18 9.19.1 register_server operation

19

```

20     void register_server(inout Location  the_location,
21                         in ServerId    the_server_id,
22                         in Properties   props,
23                         in ServerCallback callback)
24         raises (ServerGroupNotFound, AlreadyRegistered, UnsupportedCallback);

```

25

26 The *register_server* operation is used to notify the LWFT infrastructure about the existence of a Server and the ServerGroup
27 that it belongs to.

28 The *the_location* argument denotes the location of the server replica. It is expected that the location will contain host and
29 process or process identification. If (and only if) the process provides an empty location then the infrastructure could provide
30 a value.

31 The *the_server_id* argument identifies the Server (Process) group. It is expected to contain “-*ORBServerId*” value of
32 **ORB_init**.

33 The *props* parameter allows for infrastructure-specific properties that may be required for correct functioning of the
34 application. Some implementations may require, for example, process endpoints as provided by by *-ORBListenEndpoints*.

35 The *callback* argument allows the LWFT infrastructure to optionally provide additional control or monitoring over the
36 process. The infrastructure can narrow it to specific subcategories.

37 The ServerGroupManager will raise **ServerGroupNotFound** exception when *the_server_id* does not denote an existing

1 process group. When *location* has already been registered, **AlreadyRegistered** exception will be raised.

2 **9.19.2 unregister_server operation**

```
3  
4 void unregister_server(in Location the_location)  
5     raises (ServerNotFound);  
6
```

7 The *unregister_server* operation is used to notify the LWFT of deliberate deregistration of a server from its server group.
8 This is may occur during graceful stop of the application.

9 The *location* argument identifies the location of the server that is to be unregistered.

10 **ServerNotFound** exception is raised when *location* does not denote an already registered server.

11 **9.19.3 server_ready operation**

```
12  
13 void server_ready(in Location the_location)  
14     raises (ServerNotFound,  
15           WrongStatus);  
16
```

17 The *server_ready* operation is used to notify the LWFT infrastructure that server recovery has completed and that the
18 application is ready to service incoming requests.

19 The *location* argument identifies the location of the server ready to service FT requests.

20 **ServerNotFound** exception is raised when *location* does not denote an already registered server.

21 **WrongStatus** exception is raised when the server is in a status that does not expect calls to the server ready operation (this
22 may occur when the server calls *server_ready* multiple times).

23

24 **9.20 ServerGroupObserver**

25 **ServerGroupObserver** interface is an interface actors external to the middleware should implement to be made aware of
26 server registration (i.e. creation); unregistration (i.e. destruction / failure); and server readiness and un-readiness

27

```

1  interface ServerGroupObserver
2  {
3      void on_register_server (in Location    the_location,
4                              in ServerId    the_server_id,
5                              in Properties   props,
6                              in ServerCallback callback);
7      void on_unregister_server (in Location the_location);
8      void on_server_ready (in Location the_location);
9  };

```

10 9.20.1 on_register_server operation

```

11
12  void on_register_server (in Location    the_location,
13                          in ServerId    the_server_id,
14                          in Properties   props,
15                          in ServerCallback callback);
16

```

17 This operation is called whenever a server at location ‘**the_location**’ is registering at the ServerGroupManager joining the
18 server group ‘the_server_id’.

19 9.20.2 on_unregister_server operation

```

20
21  void on_unregister_server (in Location the_location);
22

```

23 This operation is called whenever a server at location ‘**the_location**’ is unregistering from its server group at the
24 ServerGroupManager.

25 9.20.3 on_server_ready operation

```

26
27  void on_server_ready (in Location the_location);
28

```

29 This operation is called whenever a server at location ‘**the_location**’ is calling server_ready operation of the
30 ServerGroupManager.

31

1 **9.21 Server Group Manager Ext**

2 This extension to ServerGroupManager allows actors external to the middleware to be made aware of server registration (i.e.
3 creation); unregistration (i.e. destruction / failure); and server readiness and un-readiness. By registering a reference to an
4 implementation of ServerGroupObserver

5 The FT middleware ServerGroupManagerExt will notify all registered observer instances of the ServerGroupObserver
6 interface, in order of registration, when one of its own ServerGroupManager operations have been invoked, by calling the
7 same method on all listeners and passing on the values that it received itself.

```
8  
9 module LWFT  
10 {  
11 interface ServerGroupManagerExt : ServerGroupManager  
12 {  
13 // Empty sequence means all  
14 void register_observer (in ServerIdSeq server_groups,  
15 in ServerGroupObserver observer);  
16  
17 void unregister_observer (in ServerIdSeq server_groups);  
18 };  
19 };
```

20

21 **9.21.1 Implicit operation register_server**

22

23 Upon an invocation register_server from an client the ServerGroupManagerExt should call on_register_server on all
24 registered observers. This should be done prior to returning a response to register_server to the client.

25 If an exception is encountered on any listener register_server invocation then the ServerGroupManagerExt should call
26 on_unregister_server on all observers that the call succeeded on (ignoring any exceptions generated from these
27 on_unregister_server calls) before propagating the exception back to the client.

28

29 **9.21.2 Implicit operation unregister_server**

30

31 Upon an invocation of unregister_server from a client the ServerGroupManagerExt should call on_unregister_server on all
32 registered ServerGroupObserver observers. This should be done prior to returning a response to register_server to the client.

33 If an exception is encountered on any observer on_unregister_server invocation then the ServerGroupManagerExt should call
34 on_unregister_server on all remaining observers (ignoring any further exceptions generated from those on_unregister_server

1 calls) before propagating the original exception back to the client.

2 **9.21.3 Implicit operation server_ready**

3 Upon an invocation of server_ready the ServerGroupManagerExt should call the corresponding call to all registered
4 observers.

5 **9.22 RecoverableServer Interface**

6 Observer interface for being notified whenever recovery is in progress in the case of primary/backup passive replication, i.e.,
7 a backup is being promoted to become a primary.

```
8  
9 module LWFT  
10 {  
11 interface RecoverableServer : ServerCallback  
12 {  
13 void start_recovery (in Properties props);  
14 };  
15 };
```

16 **9.22.1 start_recovery operation**

17 This callback operation instructs the backup replica that recovery is starting. The infrastructure can call this method to notify
18 the server that it should perform the actions needed to make it ready to accept requests as a primary, for example.

19 When the server is ready to accept requests it should signal the infrastructure by calling **ServerManager::server_ready()**.

20 **9.23 ServerRecoveryManager interface**

```
21  
22 module LWFT  
23 {  
24 local interface ServerRecoveryManager : ServerManager  
25 {  
26 boolean register_recovery_observer (in RecoveryObserver rec);  
27 boolean unregister_recovery_observer (in RecoveryObserver rec);  
28  
29 readonly attribute ServerStatusValue replica_status;  
30 readonly attribute ServerReplicationRoleValue replica_role;  
31 };  
32 };
```

33

1

2 **9.24 ServerManager**

3 **ServerManager** is a local interface that lets the application manage its ServerGroup registration, de-registration and notify
4 the FT infrastructure of its readiness to receive requests.

5 Compliant ORBs / FT implementations should make this available to the application developer by
6 **ORB::resolve_initial_references ("FTServerManager")**.

7

```
8 local interface ServerManager  
9 {  
10 attribute Location the_location;  
11 void register_server ();  
12 void server_ready ();  
13 void unregister_server ();  
14 };
```

15 **9.24.1 the_location attribute**

16

```
17 attribute Location the_location;
```

18

19 This value will default to -ORBFTLocation if specified. Attempting to set a an alternate-ORBFTLocation value on more than
20 one ORB initialisation in process is a user error and the behaviour is undefined. The value of the the_location attribut may
21 not be changed after the operation register_server has ben called.

22 **9.24.2 register_server operation**

23

```
24 void register_server ();
```

25

26 This call prompts the ServerManager to register the server with the LWFT ServerGroupManager

27 **9.24.3 server_ready operation**

28

```
29 void server_ready ();
```

30

1 This call will notify the LWFT ServerGroupManager that this server is now fully initialised and ready to receive requests.

2 **9.24.4 unregister_server operation**

```
3  
4 void unregister_server ();
```

6 The application may call this to prompt the call to unregister_server on the ServerGroupManager.

8 **9.25 FT Current**

9 This local interface provides access to request context information (*client_id*, *retention_id* and *expiration_time*). Applications
10 can retrieve its object reference by calling *ORB::resolve_initial_references("FTCurrent")*.

11 Server implementations can use this information to identify a re-invocation of a request when at most once semantics need to
12 be ensured.

```
13  
14 module FT {  
15     // resolve_initial_references("FTCurrent");  
16     local interface Current : CORBA::Current {  
17  
18         exception NoContext { };  
19  
20         string get_client_id()  
21             raises(NoContext);  
22         long get_retention_id()  
23             raises(NoContext);  
24         TimeBase::TimeT get_expiration_time()  
25             raises(NoContext);  
26     };  
27 };
```

28 **9.25.1 NoContext exception**

```
29  
30 exception NoContext { };  
31
```

32 Exception that indicates a Current operation was invoked outside of an FT invocation.

1 **9.25.2 get_client_id operation**

2
3 **string get_client_id() raises(NoContext);**
4

5 Returns the client ID extracted from **FT_REQUEST** service context. **NoContext** exception will be raised if the operation is
6 called outside of an FT operation.

7 **9.25.3 get_retention_id operation**

8
9 **long get_retention_id() raises(NoContext);**
10

11 Returns the retention ID extracted from **FT_REQUEST** service context. **NoContext** exception will be raised if the operation
12 is called outside of an FT operation.

13 **9.25.4 get_expiration_time operation**

14
15 **TimeBase::TimeT get_expiration_time() raises(NoContext);**
16

17 Returns the expiration time extracted from **FT_REQUEST** service context. **NoContext** exception will be raised if the
18 operation is called outside of an FT operation.

19

20 **9.26 FaultNotifier**

21 This submission uses the CORBA Fault Tolerance definition of FaultNotifier (see [FTCORBA]).

22 Failure reports are conveyed to the Fault Notifier by the middleware independent Fault Detectors and by the Fault Notifier to
23 the entities that have been registered for such notifications. The Fault Detectors and Fault Notifier use a well-defined event
24 type to convey a given failure event.

25 This specification defines an event type that is understood by the FT-enabled fault tolerance middleware. Vendors or the
26 OMG may extend this to convey other failure or fault event types; but only the following failure event type will guarantee
27 portability over FT-enabled infrastructures.

28 Middleware-independent mechanisms are responsible for enforcing the fail-stop semantics for the failed location.

29 **9.27 ServerCrashFault**

30 The ServerCrashFault event type reports that a given process at a given location has crashed. The definition for the event type

1 is as follows:

```
2
3 CosNotification::StructuredEvent fault_event;
4 fault_event.header.fixed_header.event_type.domain_name = "FT_CORBA";
5 fault_event.header.fixed_header.event_type.type_name = "ServerCrashFault";
6 fault_event.filterable_data_length(2);
7 fault_event.filterable_data[0].name = "FTDomainId";
8 fault_event.filterable_data[0].value = /* Value of FTDomainId bundled into any */;
9 fault_event.filterable_data[1].name = "Location";
10 fault_event.filterable_data[1].value = /* Value of Location bundled into any */;
```

11 The **filterable_data** part of the event body contains the identity of the crashed process as a pair of name-values: the fault
12 tolerance domain identifier and the process's location identifier.

13 **9.28 RecoveryObserver**

14 Observer interface for being notified whenever recovery is in progress in the case of primary/backup passive replication, i.e.,
15 a backup is being promoted to become a primary.

```
16
17 module LWFT {
18     local interface RecoveryObserver
19     {
20         /**
21         * Instructs the backup replica that recovery is starting.
22         */
23         void on_recovery();
24     };
25 };
26
27
```

28 **9.29 Fault Detection**

29 **9.29.1 PullMonitorable interface**

30 CORBA Fault Tolerance [FTCORBA] defines a *PullMonitorable* interface for use by pull-based Fault Detectors. FT-enabled
31 middleware Fault Detectors may ping periodically process callback objects supporting the **PullMonitorable** interface by
32 invoking the **is_alive()** operation.

33

```

1  module FT {
2      interface PullMonitorable
3          boolean is_alive();
4      };
5  };

```

6 9.29.1.1 is_alive operation

7

```
8  boolean is_alive();
```

9

10 *“This operation informs the pull-based Fault Detector whether the object is able to accept requests and produce replies. The*
11 *monitored object may return true directly to indicate its liveness, or it may perform an application-specific “health” check*
12 *(for example, assertion check) within the operation and return false if the test shows that the object is in an inconsistent state.*

13 *Return Value*

14 *Returns true if the object is alive and ready to take further requests, and false otherwise.”*

15 9.29.2 PullMonitorableServer interface

16 A ServerCallback specialisation for monitoring the health of the FT process by the FT-enabled middleware .

17

```

18  module LWFT
19  {
20      interface PullMonitorableServer : FT::PullMonitorable,
21          ServerCallback
22      {
23      };
24  };

```

25

26 9.30 ObjectGroupManager

27 The Portable Object Adapter may use the **PortableGroup::ObjectGroupManager** for group membership operations as
28 specified in the Fault Tolerant CORBA when created with **AUTO_OBJ_GRP_MNGR** or
29 **USER_CTRL_OBJ_GRP_MNGR** values for ObjectGroupMembershipPolicy.

30 When **AUTO_OBJ_GRP_MNGR** is used for POA creation, the POA should first resolve the object group name using the
31 **Locator** interface or from the **NameService** when no Locator is available.

```

1  module PortableGroup {
2      // Specification of ObjectGroupManager Interface
3      interface ObjectGroupManager {
4          ObjectGroup create_member
5              (in ObjectGroup object_group,
6               in Location the_location,
7               in _TypeId type_id,
8               in Criteria the_criteria)
9          raises (ObjectGroupNotFound,
10                 MemberAlreadyPresent,
11                 NoFactory,
12                 ObjectNotCreated,
13                 InvalidCriteria,
14                 CannotMeetCriteria);
15         ObjectGroup add_member
16             (in ObjectGroup object_group,
17              in Location the_location,
18              in Object member)
19         raises (ObjectGroupNotFound,
20                 MemberAlreadyPresent,
21                 ObjectNotAdded);
22         ObjectGroup remove_member
23             (in ObjectGroup object_group,
24              in Location the_location)
25         raises (ObjectGroupNotFound, MemberNotFound);
26         Locations locations_of_members
27             (in ObjectGroup object_group) raises(ObjectGroupNotFound);
28         ObjectGroupId get_object_group_id
29             (in ObjectGroup object_group) raises(ObjectGroupNotFound);
30         ObjectGroup get_object_group_ref
31             (in ObjectGroup object_group) raises(ObjectGroupNotFound);
32         Object get_member_ref
33             (in ObjectGroup object_group,
34              in Location loc)
35         raises(ObjectGroupNotFound, MemberNotFound);
36     }; // end ObjectGroupManager
37 }; // end of PortableGroup
38

```

39 9.31 Group Object Adapter

40 The Group Object Adapter (GOA) specified in the MIOP specification may also be used for encapsulating group membership
41 for CORBA objects/components. Use of **AUTO_OBJ_GRP_MNGR** POA policy at POA creation shall create a Group Object
42 Adapter. The GOA may perform automatic calls to `add_member()` and `remove_member()` on the `ObjectGroupManager`.

43 The `ObjectGroupManager` may be obtained using `ORB.resolve_initial_references("FTObjectGroupManager")`;

```

1
2  module PortableServer {
3
4      exception NotAGroupObject {};
5      typedef sequence <PortableServer::ObjectId> IDs;
6
7      interface GOA : ::PortableServer::POA {
8          PortableServer::ObjectId
9          create_id_for_reference(in Object the_ref)
10             raises (NotAGroupObject);
11         IDs
12         reference_to_ids (in Object the_ref)
13             raises (NotAGroupObject);
14         void
15         associate_reference_with_id
16             (in Object ref, in PortableServer::ObjectId oid)
17             raises(NotAGroupObject);
18         void
19         disassociate_reference_with_id
20             (in Object ref, in PortableServer::ObjectId oid)
21             raises(NotAGroupObject);
22     }; // end interface GOA
23 };

```

24 9.31.1 create_id_for_reference operation

```

25
26  PortableServer::ObjectId
27  create_id_for_reference(in Object the_ref)
28      raises (NotAGroupObject);

```

29 The operation `create_id_for_reference()` takes as an argument a widened Group IOR and generates a unique `ObjectId` for that
30 reference. This returned `ObjectId` is later associated with a servant via the standard API in the POA; that is,
31 `activate_object_with_id()`.

32 9.31.2 reference_to_ids operation

```

33
34  typedef sequence <PortableServer::ObjectId> IDs;
35  IDs reference_to_ids (in Object the_ref)
36      raises (NotAGroupObject);
37

```

38 The operation `reference_to_ids()` takes as an argument a widened Group IOR and returns a sequence of `ObjectIds` that are
39 currently associated with the Group IOR. To be resilient to hardware failures only one `ObjectId` is allowed.

40 9.31.3 associate_reference_with_id operation

41

```
1  void
2  associate_reference_with_id
3      (in Object ref, in PortableServer::ObjectId oid)
4      raises(NotAGroupObject);
```

5
6 The operation takes a previously generated ObjectId and associates it with a group reference. Servants activated using this
7 ObjectId will be candidates for receiving FT group requests via the group information provided in the IOR. To be resilient to
8 hardware failures only one ObjectId is allowed. The operation silently ignores repeat/duplicate associations of a
9 POA/ObjectId pair with the provided object reference.

10 This operation performs an automatic call to add_member() operation on the ObjectGroupManager. Compliant ORBs / FT
11 implementations should make the ObjectGroupManager available to the Object Adapter by **ORB::resolve_initial_references**
12 (**"FTObjectGroupManager"**).

13 **9.31.4 disassociate_reference_with_id operation**

```
14
15  void
16  disassociate_reference_with_id
17      (in Object ref, in PortableServer::ObjectId oid)
18      raises(NotAGroupObject);
```

19 The operation takes a previously generated ObjectId and removes the association it had with a group reference. Servants
20 activated using this ObjectId will no longer receive FT group requests via the group information provided in the IOR. The
21 operation silently ignores disassociations that no longer or never existed.

22 **9.32 FT CCM Component**

23 Use of object group for components provider ports (facets, supported interface and push consumer ports). The Container
24 shall handle group membership operations.

25 **9.32.1 Navigation/Introspection**

26 Used to retrieve and connect ports. To be able to connect a group of components, those navigation and events interfaces have
27 to return group references (IOGR) when required, but also individual port references (IOR).

28 If a group reference to a navigation or event interface is used, a group reference to the port should be returned.

29 If an individual/non-group reference to a navigation or event interface is used, an individual reference to the port should be
30 returned.

31 Selecting whether a group reference or an individual reference is used, CORBA Component Containers may rely on the
32 **FTCurrent** object.

1 **9.32.2 Component Activation**

2 Components that are required to be notified upon promotion of their hosting server process from backup to primary shall
3 support a **Recoverable** interface.

4 **9.33 At most once semantics**

5 Cf. 8.5.1

6 FT-aware CCM containers shall provide access to the **FTCurrent** object through standard **CCM2Context::**
7 **resolve_service_reference("FTCurrent")** operation.

8 **9.34 TransportProperties**

9 To allow for a predictable recovery time, applications may require some control on transport protocol parameters. Real Time
10 CORBA [RTCORBA] specifies a **TCPProtocolProperties** interface to allow the configuration of TCP protocol specific
11 configurable parameters.

```
12  
13 local interface TCPProtocolProperties : ProtocolProperties {  
14     attribute long send_buffer_size;  
15     attribute long recv_buffer_size;  
16     attribute boolean keep_alive;  
17     attribute boolean dont_route;  
18     attribute boolean no_delay;  
19 };
```

20 Non-RT CORBA ORBs however shall provide a means to enable use of keepalive property on TCP transport. The –
21 **ORBKeepAlive** option to **ORB_Init()** shall activate the keepalive on open TCP connections. ORB implementers are free to
22 enable keepalive on TCP connections by default.

23

24 **9.35 ServiceLocator interface**

25 The **ServiceLocator** interface enables an **ObjectGroup** to be obtained from the **ObjectGroupName** that identifies the
26 **ObjectGroup**.

27

```

1  module LWFT
2  {
3      interface ServiceLocator
4      {
5          PortableGroup::ObjectGroup locate (in ObjectGroupName group)
6              raises (PortableGroup::ObjectGroupNotFound);
7
8          attribute ServiceLocator fallback;
9      };
10 };

```

11

12 The ServiceLocator interface may also be used by the POA (in the case when local interface for use by the POA (in the case
13 of AUTO_OBJ_GRP_MNGR ObjectGroupMembershipPolicy) to locate the ObjectGroup reference needed for group
14 membership via the PortableGroup::ObjectGroupManager interface.

15 Compliant ORBs / FT implementations should make this available to the Object Adapter by
16 **ORB::resolve_initial_references("FTServiceLocator")**.

17 **9.35.1.1 locate operation**

18

```

19 PortableGroup::ObjectGroup locate (in ObjectGroupName group)
20     raises (PortableGroup::ObjectGroupNotFound);

```

21

22 This operation will return the object group reference of the ObjectGroup named by the ObjectGroupName or throw a
23 ObjectGroupNotFound exception if the name is not recognised.

24 **9.35.1.2 fallback attribute**

25

```

26     attribute Locator fallback;

```

27

28 If this attribute is not nil then the Locator will try the locate() on this Locator instead of raising ObjectGroupNotFound.

29

30 **9.36 ForwardingServiceLocator**

31 The ForwardingServiceLocator is an interface that extends the behaviour of Service Locator with two pseudo operations as
32 follows:

33

```

1  module LWFT
2  {
3      interface ForwardingServiceLocator : Locator
4      {
5          // Pseudo operations
6          // FT::ObjectGroup FT_Locate ();
7          // Requests made to FT_Locate should invoke any
8          // registered RequestDecoders and will then call
9          // ::locate returning the located object to
10         // the caller.
11
12         // <any other request>
13         // Will invoke any registered RequestDecoders and will
14         // then call ::locate returning a LOCATION_FORWARD
15         // response to the located object to the caller.
16     };
17 };

```

19 The proposed mechanism makes use of location forwarding techniques to locate and return the latest ObjectGroup reference
20 for a Client.

21 Object group references created by the FT-enabled middleware should at minimum contain a profile with a
22 ForwardingServiceLocator endpoint information.

23 Fully compatible ORBs may contact the ForwardingServiceLocator component by and send a request message using the
24 reserved operation name 'FT_Locate'. FT_Locate is a new reserved operation name to use in request messages, the expected
25 response to which is a reply with a LOCATION_FORWARD status value and an encoded IOR in the message body or an
26 Object reference return value with a normal success return status.

27 A transport connection opened with the endpoint information of a ForwardingServiceLocator can be used to locate
28 ObjectGroups by despatching CORBA requests to it as though the objects were located there. The behaviour of the
29 ForwardingServiceLocator endpoint when it receives a request for an object_key that would normally not exist there is the
30 following:

- 31 1. Utilise whatever *RequestDecoders* it has registered to extract an ObjectGroupName.
- 32 2. If not successful then the ForwardingServiceLocator will return OBJECT_NOT_EXIST system exception with
33 COMPLETED_NO completion status.
- 34 3. If an ObjectGroupName is decoded then the value will be passed to Locator::locate
- 35 4. If the locate call returns an ObjectGroup the behaviour depends upon the request_id of the request. If the request
36 id was FT_Locate then the ObjectGroup should be returned to the client. Any other request to the ObjectGroup
37 should be returned as a LOCATION_FORWARD response.

38

1 **9.37 RequestDecoder**

2 A request decoder provides a portable way to provide a mechanism to extract the ObjectGroupName from a CORBA Request.
3 These can be plugged in at any PI request interception point to provide ObjectGroup location capabilities. It is envisaged that
4 ORB / service implementers will provide a mechanism for application developers to register an ordered list of these to
5 provide location capabilities (see ServiceLocator interface).

```
6  
7 module LWFT  
8 {  
9 local interface RequestDecoder  
10 {  
11 // From the given request extract the ObjectGroupName  
12 boolean get_name_from_request  
13 (in PortableInterceptor::RequestInfo request,  
14 out ObjectGroupName group_name);  
15 };  
16 };
```

18 **9.38 Necessary ORB modifications**

19 Fully compliant ORBs must be capable of accepting and correctly responding to GIOP request messages using a new
20 reserved operation name FT_Locate. In this section we will explain why there is a need for this. The details of how ORBs that
21 do not implement this message type can still interact with the service will be covered in §9.39.

22 The key principle behind the correct functioning of the LWFT mechanism proposed here is the notion of forwarding
23 references between a Client and a suitable Server replica. Some aspects of the RT CORBA specification require that
24 information regarding individual objects be provided in the IOR. To support this, while it is not required that all objects be
25 registered at the point that they are created, it is necessary for the Forwarder to be capable of querying a particular object for
26 its full direct (i.e. not redirected via the Forwarder) IOR. The LWFT mechanism therefore requires that the response given to
27 any request message is a LOCATION_FORWARD one. This differs to the standard GIOP LocateRequest message type.

28 As defined in Chapter 15 of the CORBA 3.0.3 Specification ('General Inter-ORB Protocol'), there are two possible request
29 message types which can be used by standard ORBs: Request and LocateRequest. The standard GIOP response types for
30 these request types are Reply and LocateReply respectively. Provided that no exception occurs there are two valid status
31 values that can be carried in the header of a Reply message, and three in a LocateReply message:

32

| Request Type | Response Type | Possible non-exceptional header status values | Meaning |
|---------------|---------------|---|--|
| Request | Reply | NO_EXCEPTION | The message body contains the result of an operation invocation. |
| | | LOCATION_FORWARD | The message body contains an IOR. |
| LocateRequest | LocateReply | OBJECT_HERE | The server returning this message can accept requests for the object specified in the LocateRequest message. |
| | | OBJECT_FORWARD | The message body contains an IOR. |
| | | UNKNOWN_OBJECT | The object specified in the LocateRequest message is not known at this server. |

1 **Table 2: The GIOP response types and their possible status header values**

2

3 Both Reply and LocateReply have a status header value which enables them to forward Clients IORs in their message body.
4 The problem with LocateReply is that the CORBA specification does not dictate that all ORBs must implement location
5 forwarding behaviour at the Server's side. ORBs which do not implement location forwarding will therefore only ever
6 respond to LocateRequest messages using OBJECT_HERE or UNKNOWN_OBJECT status values. There may also be ORB
7 implementations which do not always choose to return forwarding type responses to requests.

8 This is clearly a problem for object group references created by a POA with ObjectGroupMembershipPolicy value of
9 AUTO_ORB_CTRL as it relies entirely on the status of a response message being a reference forwarding type every time.
10 The solution to this problem is to implement a new reserved request operation, FT_Locate, to which the only possible (non-
11 exceptional) response is either a LOCATION_FORWARD or a normal return of an object reference.

12 **9.39 Providing LWFT to unmodified ORBs**

13 An important requirement of this LWFT design is that it can still provide a limited service to ORBs that do not implement the
14 new FT_Locate reserved operation name described above in §9.36. If a Server ORB with POAs using
15 ObjectGroupMembershipPolicy value of AUTO_ORB_CTRL does not support requests using FT_Locate, it is not possible
16 for the Forwarder to obtain an IOR for an individual object from the Server. As a result of this, any IOR which is forwarded
17 to the Client (the CORBA standards state that any ORB must be able to process a LOCATION_FORWARD on the Client's
18 side) must come from the Forwarder alone. In this situation it is possible for the Forwarder to generate an IOR for the Client
19 using the endpoint information that it usually uses to contact the Server. Forwarders should interpret a BAD_OPERATION
20 exception response to a request using FT_Locate as indicating that a Server replica object does exist, but that the Server ORB
21 does not support this reserved operation name. It will be possible to control which Server replicas will be contacted using the
22 FT_Locate operation name by using a suitable policy.

23 While it may initially sound like there are no negative effects produced by applications using the LWFT Middleware in this
24 manner, there unfortunately is one key problem: any application not using location forwarding on the Server's side will not be
25 fully compatible with RT CORBA or any other services which require the use of IOR Tagged Components on a per object
26 (rather than per ORB or per POA) basis. In the normal approach, the IOR forwarded back from the Server can contain
27 additional pieces of information, in the form of IOR Tagged Components, which are required to provide protocol support for
28 applications implementing the Real-time CORBA specification. For Servers not implementing location forwarding the
29 Forwarder must build an IOR using the information that the Server registered itself with. As these registered details might
30 contain out of date tagged information, if any at all, the IORs generated from this data will not be suitable for use by Real-

1 time applications.

2 Although Server ORBs not implementing FT_Locate will not be fully compatible with RT CORBA, or any other service
3 requiring the presence of up to date additional information in IORs, they will still be able to use the basic LWFT service. This
4 ensures that unmodified ORBs will be able to gain some advantage from the proposed LWFT service, and that as high a level
5 of interoperability as possible is provided.

6