A P2P BASED FAILURE DETECTION MODEL FOR DISTRIBUTED SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CELAL KAVUKLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2006

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certified that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ali Hikmet Doğru
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Nihan K.Çiçekli (CENG, METU)

Assoc. Prof. Dr. Ali Doğru (CENG, METU)

Dr. Ali Arifoğlu (II, METU)

Dr. Tolga Can (CENG, METU)

Dr. Aysu Betin Can (II, METU)

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Celal Kavuklu

Signature :

ABSTRACT

A P2P BASED FAILURE DETECTION MODEL FOR DISTRIBUTED SYSTEMS

Kavuklu, Celal

M. S., Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ali Hikmet Doğru

August 2006, 84 pages

A comprehensive failure detection model is proposed to detect service failures in asynchronous distributed systems. The proposed model takes advantage of P2P technology to provide required functionality. When compared to similar studies in failure detection, the presented failure detection model is more autonomous in resolving service dependencies, embodies more flexibility in providing different failure detection functions (like unreliable failure detectors, membership services) and offers more security. A failure detection library is developed using JXTA P2P framework to show realization of such a model.

Keywords: Failure Detection, P2P, Asynchronous Distributed Systems, Unreliable Failure Detector, Membership Service, Service Dependency Resolution.

ÖZ

DAĞITIK SİSTEMLER İÇİN P2P'YE DAYALI HATA BULMA MODELİ

Kavuklu, Celal

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Danışman: Doç. Dr. Ali Hikmet Doğru

Ağustos 2006, 84 sayfa

Asenkron dağıtık sistemlerde oluşan servis hatalarının yakalanabilmesi için bir hata yakalama modeli sunulmuştur. Sunulan model, gerekli fonksiyonaliteyi sağlamak için P2P teknolojisini kullanmaktadır. Sunulan hata yakalama modeli, hata yakalama alanında yapılan benzer çalışmalara oranla, servis bağımlılık çözümlemesinde daha otonomdur, farklı hata yakalama mekanizmalarını (güvenilmeyen hata yakalayıcılar, üyelik sistemleri) desteklemekte daha esnektir ve daha güvenlidir. Sunulan modeli gerçekleyen ve JXTA P2P mimarisini kullanan bir hata yakalama kütüphanesi geliştirilmiştir.

Anahtar Kelimeler: Hata Yakalama, P2P, Asenkron Dağıtık Sistemler, Hata Yakalayıcılar, Üyelik Sistemleri, Servis Bağımlılık Çözümlemesi.

To My Family,

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. Ali Hikmet Doğru, for his guidance throughout the research. I also wish to thank to Mr. Levent Alkışlar, director of RADAR & Electronic Warfare Software Engineering Department, on behalf of ASELSAN Inc. for his guidance and motivation for the completion of this thesis. To my colleagues in Test Engineering Department and RADAR & Electronic Warfare Software Engineering Department; my parents Muharrem and Güler Kavuklu, my sister Aslı Kavuklu, I offer sincere thanks for their emotional support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ACL	Access Control List
API	Application Programming Interface
DNS	Domain Name System
DoS	Denial Of Service
GAUTH	Group Authority
IP	Internet Protocol
LAN	Local Area Network
NAT	Network Address Translation
OS	Operating System
OWL	Web Ontology Language
P2P	Peer to Peer
QoS	Quality Of Service
RDF	Resource Description Framework
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
XML	Extensible Markup Language

CHAPTER 1

INTRODUCTION

The role of distributed systems in our daily life increases constantly. These systems include many types of hardware and software components which cooperate to provide distributed system services like internet services (such as messengers, search engines, etc.), command control applications, computing grids, telecom applications, etc. Components of such distributed systems are generally spread over large scale asynchronous networks like the Internet.

Parallel to the exponential increase in complexity of these systems, management costs also increase. The primary cost of managing a system stems from the need of providing availability of its services. Failure detection and recovery mechanisms have been extensively studied in the area of availability of distributed systems and services. Failure detection is the primary step of providing availability for these systems where after recovery mechanisms can be applied.

The services provided by a distributed system are provided by hierarchical cooperation of various hardware and software components of the system. To achieve availability of these services, we need to monitor the status of these services which in turn requires monitoring the status of distributed components forming service functionality. In a complex distributed environment, determining the status of these components and merging them to derive the status of system services they provide is a hard job. If we think of the scenario where failure of one disk drive can catastrophically result in the failure of transaction services of OS which in turn cause online banking system unavailability or malfunctioning as a chain reaction; we get a better understanding about the complexity of this task. Failures should be carefully analyzed throughout the system to determine the affected parts/services.

Efficient and autonomous failure detection algorithms, analysis and communication services are needed to detect service failures in these systems. Autonomous operation is needed to isolate human errors at failure decisions [10].

1.1. Scope of this work

The purpose of this thesis is to establish a comprehensive failure detection model to derive service failures in asynchronous distributed systems by providing an infrastructure to implement various types of failure detection mechanisms. Realization of this model with a proof-of-concept implementation is presented. The model takes advantage of P2P technology to derive failures of distributed system services that depend on failed/unhealthy processes. It provides dedicated peer group services as failure detection services for each distributed system. It is argued that, in addition to providing required functionality compared to its rivals, such a model is more flexible in employing different failure detection mechanisms (like unreliable failure detectors or group membership services) which makes it more suitable for many types of distributed systems having different failure detection requirements, more autonomous in resolving service failures and more secure.

The reference implementation is a failure detection library developed using JAVA as a programming language and Project JXTA as a P2P platform. The failure detection library checks aliveness of distributed system processes which are realized as peers and analyzes these results in failure detection services which are realized as peer group services in JTXA framework. JTXA framework is a Sun Microsystems, Inc. incubated P2P platform and has a decentralized architecture, without a single point of failure, scalable and supporting frequent node arrivals and departures.

1.2. Organization of the Thesis

Beyond this introductory chapter, the thesis is organized as follows: In Chapter 2, necessary background on distributed systems, distributed system service concept, failure detection and P2P are included. Chapter 3 describes the general requirements of a failure detection model and presents a logical view of such a model. In chapter 4, related work in the failure detection literature is presented in the light of the logical view of a failure detection model. Chapter 5 presents P2P based failure detection model that has been proposed in the thesis. In Chapter 6, realization of the model is presented with realization decisions and a sample implementation. Chapter 7 provides conclusions and further work for this study. Sample implementation provided is modelled with Unified Modelling Language (UML) and implemented using JTXA P2P platform. Documentation for the design is represented in Appendix A. JTXA P2P framework is presented in Appendix B. Documentation for JXTA calls necessary to provide main functionality in the sample implementation is provided in Appendix C.

CHAPTER 2

BACKGROUND

2.1. System Service

In [17], authors define a taxonomy where a *service* is described as the behaviour of the system (service provider) perceived by the users which is another system receiving the service. Service delivery takes place in *service interface* where the perceivable face of the service provider is called as *external state* and the rest as *internal state*.

2.2. Failure Detection Model

Failure terminology defined in [17] is used in this study:

- A *failure* occurs when a service deviates from its correct behaviour, for some definition of correctness.
- An *error* is the deviation of total state of a service from the correct service state
- A *fault* is the determined cause of an error

It is worth to emphasize that an error may not affect the external state of a service so may not cause a failure. If an error caused a service failure, it should have affected at least one external state of the service.

It should also be noted that in a complex distributed system, what is considered as a failure in one layer might be considered as a fault that causes an error and maybe another failure in another layer. For example, a CPU might crash (failure) because of a malfunctioning heat controller (error) which is caused by a bug in its firmware (fault). At a higher level, the failure of the CPU (fault) can prevent access to hard disks (error) of the computer and stop providing any service (failure) to users. This example also explains the complex dependency relations in a distributed system and the challenge of understanding and reasoning about the error.

In this thesis, the failure of a process means that either a process is *fail-stopped* which means it permanently transits to a state that allows other components to detect that it has failed (for instance, by ceasing to send heartbeat messages) or declare failure of itself due to malfunctioning (for instance, because of accumulating unparsed messages in its interfaces or exceptions from memory modules).

2.3. Asynchronous Distributed System

The failure detection model presented in this thesis is based on an *asynchronous network model* where there is no assumption about the message transfer delays or the speed of objects execution. Although it might be possible to construct a synchronous, fixed and reliable network in some scenarios such as for sensors on an airplane wing, such a network model would not be realistic for real world such as the Internet.

> *An asynchronous distributed system is a distributed system composed by a set of m >= 1 hosts {H_1,. . . , H_m} where execute concurrently k >= 1 distributed protocols { A_1 , . . . , A_k } . Each protocol Ai is carried out by nA_i >= 1 objects that cooperate through message exchanging. Each of those objects resides on one of the host in {H_1, . . . , H_m}. Multiple objects can then execute on the same host. Network is reliable i.e., messages are eventually delivered by the network to the intended receiver. There is no assumption about the message transfer delays or the speed of objects execution, so this makes the distributed system asynchronous* [15].

In the scope of this thesis, these objects can be an application or individual tasks/components/threads/processes of an application. These objects cooperatively work to provide the distributed functions of the system {F_1 , . . . , F_h}, where h>=1. For example, a distributed system in a radar provides many functions like broadcasting, antenna rotation, friend or foe interrogation, etc. These functions are provided by the cooperative work between these individual objects.

2.4. Peer-to-Peer (P2P) Architecture

A peer-to-peer (P2P) architecture is defined for distributed applications. A P2P network can be described as peer groups, which are formed with peers with a common interest, that build a self-organizing overlay network on top of an already existing network architecture. Overlay networks provide an abstraction over the underlying network topology and architecture.

Peer Group is a virtual network space consisting of a subset of all devices accessible via an overlay network. These groups provide meeting points for the peers having same concerns and provide *peer group services* that operate in the scope of a peer group and provide services for the group members.

2.5. Failure Detection (FD)

2.5.1. Unreliable Failure Detectors

> *An unreliable failure detector is a distributed oracle that provides objects hints about the behaviour of other objects. An unreliable failure detector can make mistakes by considering correct objects as crashed because of the asynchronous nature of the network (difference between a slow process and a dead process)* [15].

There exist two important characteristics of an unreliable failure detector: *accuracy* and *completeness* [11]. Various unreliable failure detectors are proposed according to these two properties which outline their mistakes [12]. Accuracy property enables failure detectors to remove suspected label from a healthy process after some time. Completeness property enables failure detectors to permanently label crashed processes as suspected after some time.

2.5.2. Membership Service

The membership services are used to reach a consensus on membership of processes in a process group. Membership of a process group changes when its processes (members) join/leave (voluntarily or because of a failure) the group [26]. View of a group membership is the list of processes that are currently members of this group. Membership services transmit these views to their live members in case of updates to their views. This way, live members are informed about failed, recovered and newly joined members. According to view consistency of members, membership services can be classified into two groups: *weakly consistent membership services* and *strongly consistent membership services*.

CHAPTER 3

REQUIREMENTS AND LOGICAL VIEW OF A FAILURE DETECTION MODEL

In this chapter, requirements for a comprehensive failure detection model are presented. Logical blocks of such a model providing necessary requirements are presented at the end of this chapter.

3.1. Requirements

Requirements for a failure detection model are grouped into three categories: failure detection service, management requirements, and communication requirements.

3.1.1. Failure Detection Service

In [16], authors define scalability, adaptability, QoS and flexibility as primary requirements for a failure detection service:

- *Scalability*: A failure detection service for a distributed system should be designed in a scalable way because distributed systems can span hundreds of computing platforms [2]. Scalable failure detection algorithms should provide efficient detection performance even in widely physically distributed environments. It should not cause message explosions by flooding all network and resilient to message losses in the case of network node failures.

- *Adaptability*: The adaptability of failure detection should be perceived as its adaptation to various network situations. For example, a failure detection service should adapt itself by dynamically adjusting the timeout when waiting for a heartbeat message [12].

- *Flexibility*: A failure detection service should be flexible enough to provide different failure detection services for different types of applications.

- *Quality of Service*: A failure detection service should provide QoS guarantees [39]. A service providing QoS guarantees should be dynamically configured by applications to maintain a certain QoS level at runtime. The proposals to increase the QoS of a failure detection service are generally based on monitoring the QoS metrics of the service, and, if required, tuning the service by adjusting some control variables (e.g. the rate of heartbeat messages) so that it maintains QoS to the levels required by the application. In a heartbeat based strategy, for instance, if the detection latency is higher than desired, the frequency of heartbeat sending should be increased as a function of the difference between the desired and the measured detection latencies. Some QoS metrics for failure detection services are as follows: detection time (Speed to detect failures), query accuracy probability (Accuracy for failure detection), mistake rate (Accuracy for failure detection), etc.

3.1.2. Management

- *Security*: In [8], authors emphasize the need for security in large scale failure detection services. The failure detection service should be able to provide various types of security requirements.

- *Dependency Analysis*: A failure detection model should be autonomous enough to take care of system dependencies when deriving health of distributed system services. Analysing dependencies in failures is essential for deriving statuses of distributed system services which in turn increase system reliability. The model should be able to analyse dependency relation of a process on a resource and a service on a process.

3.1.3. Communication

- *Low Communication Overhead*: In order not to increase communication overhead in failure detection and to increase QoS, low-level communication primitives should be used by a failure detection model. These low-level communication primitives can include protocol tests, heartbeats, SNMP codes, etc.

- *Adaptability to different topologies in the network*: A flexible failure detection model should adapt to different topologies in the network by providing efficient broadcasting and unicasting communication protocols that support various types of distributed systems and networks. For instance: a multicast communication in a LAN environment, application level broadcast in wide area networks, a transparent unicast communication through a firewall in wide area networks, etc.

- *Adaptability to dynamism of the network*: A failure detection model operating in wide area network should adapt to dynamism of the network by providing fault-tolerant communication primitives in case of failure of intermediate nodes in a communication path.

3.2. Logical View of a Failure Detection Model

A failure detection model is logically partitioned into two semantic blocks *Failure Detection Service* and *Communication* blocks as shown in Figure 3.1:

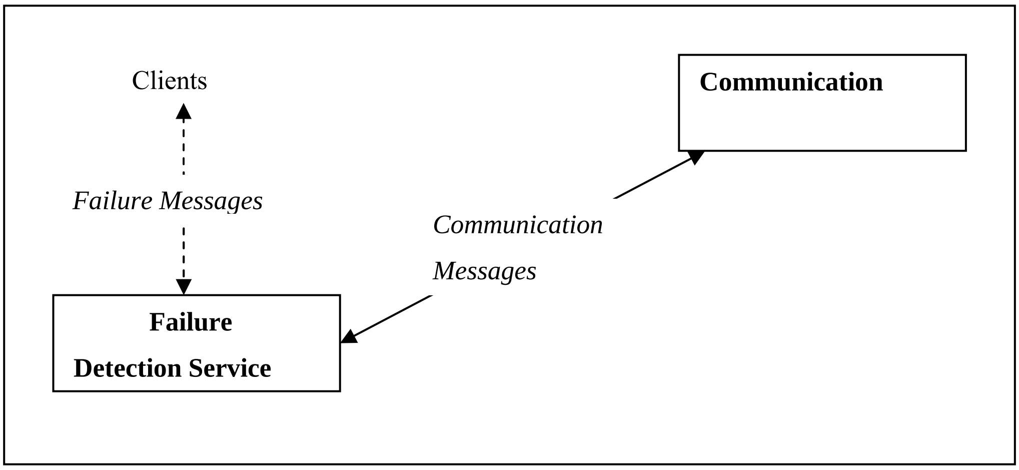


Figure 3.1 Logical View of a failure detection architecture

These blocks should be perceived as logical containers of the functionality necessary to provide mentioned requirements.

Failure detection service block includes failure semantics. In the scope of this block, requirements for a failure detection service and requirements to manage failure detection service should be satisfied.

Communication block includes functionality necessary to provide requirements listed for communication. These communication services include group communication primitives (multicast service, membership services, etc), point-to-point communication primitives (unicast services), and adaptability to underlying network conditions/system scale.

CHAPTER 4

RELATED WORK

In this chapter, first an overview of the previous work that has been produced in failure detection area is presented in the scope of the logical blocks of a failure detection model. Afterwards, existing failure detection models are presented.

4.1. Logical Blocks of a Failure Detection Model

4.1.1. Failure Detection Service Block

Recall that, this block embodies failure detection semantics. It detects failures in the distributed system and manages the failure detection service.

4.1.1.1. Determining Process Failures

In an asynchronous distributed system which is subject to even a single crash failure, it is well known that distributed consensus (agreeing on the same result, a process failure) cannot be solved deterministically. This result is called as *impossibility result*. In such environments, at best, a process can be suspected of having failed by an *agreement*, but no process can ever be known to have crashed because real crashes are indistinguishable from slow processes/communication delays [15].

In literature, there exist two types of atomic broadcast mechanisms, *Unreliable failure detectors* and *group membership services*, to solve agreement problem on process failures in asynchronous distributed systems [49]. Both mechanisms provide estimates about the set of crashed processes in the system. The main difference between a failure detector and group membership service is that a failure detector provides inconsistent information about failures, whereas a group membership service provides consistent information [23][49]. These mechanisms and their implementations have different levels of scalability, adaptability, flexibility and QoS properties [23].

4.1.1.1.1. Unreliable Failure Detectors

In [11], the concept of *unreliable failure detectors* is introduced to reach distributed consensus in asynchronous distributed systems. Unreliable failure detectors can erroneously add correct processes to suspects list at one time or another for a certain period of time, however eventually no correct process is suspected as crashed.

Following the work in [11], various types of unreliable failure detectors have been proposed so far [59][12][60][61][62] each aiming to extend certain characteristics of failure detection like scalability [63][64][65], adaptability [39][62][61], flexibility [61] and QoS [39].

4.1.1.1.2. Group Membership Service

Membership service paradigm is highly studied in scope of failure detection [18][26][27][28]. Membership services are categorized into two according to their strength in consistency:

- *Weakly consistent membership services*: In weakly consistent membership services, each member of a group may have a different membership view; which causes inconsistencies in membership views. Weakly consistent membership services have also been the subject of an extensive body of work [6][14][15]. This work can be broadly classified as differing in speed of failure detection, accuracy, message load, and completeness Epidemic and gossip-style algorithms have been used to build highly scalable implementations of this service [8][29].

- *Strongly consistent membership services*: Strongly consistent membership services use membership list and guarantee that all nodes in the system always see a consistent list through the use of atomic updates. Such membership services are used to build virtual synchrony (virtual synchrony guarantees that membership changes within a process group are observed in the same order by all the group members that remain connected). However, a limitation of virtual synchrony is that it has only been shown to perform well at small scales, such as five node systems [8].

4.1.1.2. Managing Failure Detection Service

4.1.1.2.1. Security

Managing a failure detection service includes applying fine-grained access control policies for the failure detection services it provides. These security mechanisms should remove security vulnerabilities in the system and filter malicious security attacks.

Malicious nodes can attack failure detection service to cause Denial of Service by constantly pumping failure notifications to these groups to cause false alarms / by not sending (or not replying) heartbeats to cause false alarms / by abusing ACID (Atomicity, Consistency, Isolation and Durability) properties of the communication.

Security is an overlooked topic in failure detection and is not extensively studied in failure detection literature. This may be because:

- Failure detection services are generally designed for local area networks
- It causes a performance bottleneck in service. A comprehensive study about this bottleneck is proposed in [30]. A complementary study to this is proposed in [31], where authors explain different designs of security architectures on group communication systems.

In [48], security is listed as a problem for failure detectors through which denial of service (DoS) attacks can be made. A DoS attack to a failure detection service can be done by continuously sending failure messages / heartbeat messages to failure detection service which would eventually cause malfunctioning or unavailability. In [8], the authors provide alternative implementations of their failure notification system to remove its security vulnerabilities. The idea behind these alternatives is to decrease responsibility of each node; in this way, the overall system is not affected too much because of a malicious node. However, to the best of my knowledge, no study exists in the literature providing a comprehensive infrastructure to apply fine-grained access controls in the scope of failure detection in distributed systems.

4.1.1.2.2. Dependency Analysis

The motivating and inspirational idea behind dependency analysis is the work proposed in [6], Sentinel, a P2P based distributed network monitoring system. The authors propose that, distributed system service dependencies in a distributed system can be handled by distributed monitoring peer group services. These monitoring services are aware of each other and their hierarchy represents the dependencies in the services of the system. For example, a web monitoring module should be aware of the health of DNS monitoring module which in turn should be aware of the network monitoring module. Authors argue that their model create more accurate alarms, reduce resource usage, and eliminate the need of a complex and central analysis module. In the proposed solution, however, authors do not mention how they detect failures in the system and resolve dependencies.

4.1.2. Communication Block

4.1.2.1. Multicast Communication

Group communication is the concept of a set of processes communicating through a group abstraction (by multicasting/broadcasting). The idea behind it is multicast groups. A multicast group is identified by its logical name and created / destroyed on the fly by application join/leave requests. A message sent to this group is broadcasted to all live group members. This communication schema depends on transport layer communication.

The problem of group communication is its scalability and dependency on underlying topology. When the group size increases, group communication based algorithms do not scale well. Also groups are generally formed according to underlying topology since they use multicasting for communication. This approach makes group communication dependent on the underlying topology.

Several software toolkits are available to support the construction of reliable distributed systems using group communication mechanisms. Some packages are Horus, Transis, and Totem [38][37][36]. Applications using these toolkits are provided with high-level primitives for creating and managing process groups. These include: a group membership protocol which manages the formation and maintenance of a process group, dealing with node arrivals and departures and high-level group communication services. A process may be removed from a process group because it failed, voluntarily requested to leave, or was forcibly expelled by other members of the group. A group membership protocol must manage these dynamic changes in a coherent manner.

Group communication is a very popular and extensively-studied method of detecting failures in distributed systems. Both unreliable failure detectors and group membership service use group communication and provide processes with estimates about the set of crashed processes in a process group. [26][27][28].

4.1.2.1.1. P2P

P2P systems include protocols that provide multicast communication semantics to route information between nodes. However, in order to provide a group communication system on top of these protocols, a group communication schema is needed. In [3], authors, in addition to presenting an effective self organizing publish subscribe middleware based on P2P infrastructure, provide a valuable discussion about how current P2P technologies can be designed/used to provide efficient group communication schemas.

4.1.2.2. Point-to-Point Communication

As mentioned, in large scale, group membership becomes inefficient as the number of processors or processes in a group increases. Although many systems have been based on group communication services, a gossip-style protocol to share health information of the system turns out to be a more scalable method which is based on point-to-point communication. The gossip protocols have been proved to be an effective protocol for failure detection in distributed systems [18][19][25][13].

An analogy is made with Epidemic, or gossip, based protocols and transmission of a contiguous disease in [16]; dissemination of information in these protocols uses the same randomized way as an infected transmits its illness to others. In the scope of failure detection, all processes in distributed system using epidemic protocols exchange their failure information with randomly picked processes. As opposed to initial idea about this may cause a chaos; after a certain period of time, with high probability, all processes in distributed system obtain any piece of failure information [12]. The primary advantage of gossip-based protocols for failure detection is that they are completely transparent to the underlying topology as opposed to group communication based ones which rely on multicasting [12]. As a result of point-to-point messages, the gossip protocols have little communication overhead. In other words, this class of failure detectors is completely resilient to topology changes, without requiring any additional mechanism [12][21]. However, since gossip based protocols need certain period of time to inform every process about failures, distributed consensus is achieved slower than it would with a group communication based approach.

Hybrid solutions are proposed in the literature to take advantage of both types of communications. Gossip-based protocols are tuned to be applied to group communication paradigm. For instance, a gossip based protocol for group communication could be to randomly select gossip targets from group members and transmit gossips to those targets [13][19].

4.1.2.3. Adaptability to Underlying Network Conditions

There exist several problems in providing communication services for wide area networks. Major problems are network heterogeneity / dynamism / topology differences.

- *Dynamism of the network*: In wide are networks, network is very dynamic in the sense that: at any time many routers are included in the network, the ones which are in the route of communication may fail, packets lost, etc.

- *Different topologies in the network*: There exist several types of topologies in the network which may affect communication protocols. For instance: processes which need to cooperate to provide a service of the distributed system may have different networking conditions: one of them may be in an enterprise network and behind firewalls, whereas another connected to internet using dial-up connections with a dynamic IP, etc.

To provide a certain level of abstraction over existing complex network conditions/topologies, *overlay networks* are used. It is widely accepted that, usage of overlay networks can greatly help to solve network dynamism / heterogeneity / topology difference problems plus improve some QoS constraints by separating unrelated traffic. For instance: usage of overlay networks for routing content over the underlying Internet has proven to be an effective methodology with the examples of MBONE for multicast, the 6BONE for IPv6 traffic [9]. P2P systems provide overlaying networks to constitute dynamic and adaptable networks of peers [24]. Therefore, usage of P2P infrastructure provides us with an overlaying reducing dynamism and heterogeneity of underlying physical network. Various implementations of such P2P overlay networks have been provided in the literature: Chord, Can, Pastry, Tapestry, SkipNet [34][35][20][33][32].

4.2. Existing Failure Detection Models

To the best of the current knowledge, no comprehensive failure detection model satisfying all requirements for failure detection exists in the literature. However, there exist failure detection frameworks providing a subset of functionality that must be provided by a comprehensive model.

4.2.1. FUSE

FUSE is proposed as a lightweight failure notification service for building distributed systems [8]. It uses failure detection semantics similar to unreliable failure detectors however provides stronger distributed agreement. It depends on the membership service paradigm where all live members of a FUSE group will hear a failure notification within a bounded period of time whenever a failure notification is triggered in distributed system. The most appealing ideas behind FUSE are:

- It transmits failure notifications irrespective of node or communication failures which make it adaptable to topology changes and dynamism in the network. FUSE uses a P2P based overlay network (SkipNet [32]) for this adaptation.

- It provides the assumption of detecting failures is a shared responsibility between FUSE and the application which allows applications to implement their own definitions of failure.

- It assures to transmit failure notifications in a bounded period of time regardless of system scale. It provides this facility by the functionality provided by its overlay network where this functionality checks the connectivity of FUSE nodes without using any extra messaging (like heartbeats) but pinging.

FUSE provides group abstraction for failure notifications. Member of a FUSE group that wishes to declare its failure sends a failure notification to the group which is transmitted to all live members of the group. FUSE system can also send failure notifications to a FUSE group whenever a member of the group faces a connectivity failure or node crash.

4.2.1.1. Failure Detection Service Block

- *Failure Detection Service*: FUSE is a lightweight failure detection service and uses unreliable failure detector semantics. It satisfies scalability and adaptability requirements for a failure detection service but lacks flexibility.

- *Managing Security*: FUSE satisfies security (by providing secure implementation alternatives) and performance constraints (failure transmissions in bounded period of time) but does not provide any semantics for QoS constraints.

- *Managing Dependency*: FUSE does not provide dependency semantics directly to detect service level failures. However, dependency semantic can be setup on FUSE system by the clients using the system in the following way: if for each system service and each dependency a FUSE group is created / managed by some management processes, then manually this dependency semantic can be established. However, since this functionality is not inherent in the system, this approach increases overall cost to manage the system.

4.2.1.2. Communication Block

FUSE uses application-level multicasting as a group communication primitive and group membership services to transmit failure notifications. FUSE uses SkipNet overlay network which makes it adaptable to underlying network conditions and scalable. It uses unique FUSE_ID to locate fuse groups, instead of any IP based addressing. However, since it does not support any unicast communication services, FUSE members can not send any direct message to another FUSE member.

FUSE provides alternative topological solutions to the security problems: dropped legitimate failure notifications or unnecessarily generated failure notifications. The authors do not mention authentication/authorization of nodes and any communication security protocol.

4.2.2. Failure Detection Service API

In [16], a failure detection service API is proposed. It uses unreliable failure detection semantics complemented with a hierarchical communication. It consists of three entities, monitorables, monitors and notifiables. The monitor instances monitor monitorable instances and notify notifiable instances in case of failures in monitorables. Monitor instances are also monitorables and form a hierarchical structure between them. The most appealing ideas behind failure detection service API:

- It employs mechanisms to improve scalability, adaptability and QoS of failure detection service.
- It uses pull style of communication to control rate of communication messages. As opposite to push style of communication where monitorables continuously send monitoring information to monitors; in pull style of communication, monitors query monitorables and therefore can control the rate of communication messages dynamically.
- It uses a hierarchical communication infrastructure.

4.2.2.1. Failure Detection Service Block

- *Failure Detection Service*: Failure detection service API uses unreliable failure detector semantics. It generates notification events when probability of failure of a monitorable entity is below a confidence level. It addresses scalability by using pull style of communication and a hierarchical structure which constructs notifications domains. It addresses adaptability requirements by providing a statistical layer at each monitor responsible for calculating average and standard deviation of heartbeat arrival times. This provides an adaptable failure detection service in dynamic asynchronous networks. Flexibility, however, is not supported by the API. Failure Detection service API provides a QoS layer to provide QoS related requirements. QoS layer monitors QoS metrics and if necessary adjust service control parameters to support required level of quality in failure detection.

- *Managing Security*: Failure detection service API does not address any security issues.

- *Managing Dependency*: Failure detection service API does not provide dependency semantics inherently to detect service level failures. To derive dependency failures from the API, complex relations are needed to be set up manually.

4.2.2.2. Communication Block

Failure detection service API uses an event based communication infrastructure to transmit failure messages. The communication infrastructure consists of: event channels, event sources, event listeners, and notification contexts. The event channel is a distributed communication pipe where event sources release events to and event listeners filter and dispatch events from. Notification context is a domain abstraction for efficient transmission of events by using underlying native multicast support.

The communication service of the failure detection service API is not based on an overlay network which makes it vulnerable to the underlying physical network failures. The service does not provide any application level multicast service which limits its operation area to small networks; since in wide are networks native multicast support is not provided thoroughly. The API does not provide any communication level security semantic.

CHAPTER 5

P2P BASED FAILURE DETECTION MODEL

In this chapter, a failure detection model based on P2P concepts is presented. The core of the model is the usage of services localized to peer-groups which are used as distributed failure detection services existing for each distributed system service to detect their failures. The processes of distributed system are mapped to peers in the model for failure detection among the members of the group.

The motivating idea to use P2P based services as failure detection services is the studies in [4][6]. In [4], a distributed network management framework based on a self organizing P2P overlay network is proposed to monitor health of network nodes. In [6], P2P based services are used as distributed network monitors where external state of a service is monitored to decide whether it is healthy or not. However, in the P2P based failure detection model proposed in this thesis, failure of a service is derived from its internal states (like health of its processes, resources, etc) using P2P services.

5.1. P2P Based Failure Detection Model Architecture

The model presented in this thesis references [40] to provide a dependable architecture which hosts highly available failure detection services. Conscientia architecture [40], which is based on service oriented architecture specification [1], aims to provide a dependable architecture ([5] identifies characteristics of a dependable P2P system) for generic P2P applications. The model maps generic concepts of [40] to failure detection domain.

The Conscientia architecture is composed of three types of groups: a *super* group, *category* subgroups and *service* groups. The super group act as a container of other groups in the system. All service groups are categorized in category subgroups.

The category groups are used to scope search context and improve performance for service queries.

The Conscientia architecture identifies three types of peers: rendezvous peers, worker peers and client peers. Rendezvous (RV) peers provide access points for groups in the system and exist in each group in the architecture. The peers who provide a service (worker peers) register themselves with a RV peer by joining a group where service requesting peers (client peers) make a search through. The worker peers provide the main functionality of peer group services. Client peers are simple peers providing an interface to the clients of this framework to discover / query and get results from the services provided.

5.1.1. Peers

- *Worker peer*: Worker peer concept in [40] is mapped to peers who provide failure detection mechanisms in their service groups in the failure detection model. Having many worker peers forming worker groups to provide a failure detection service provides redundancy which removes hotspots from the system. These worker groups exchange their failure information between them by using communication services provided by their service group.

- *Client peer*: This concept is mapped to peers who play a bridge role between system processes cooperating to provide distributed system services and failure detection services. This role includes checking aliveness of the processes, transmitting failure detection messages (failure notifications / heartbeats) between processes and adapting to control messages (to tune parameters of failure detection service) received from failure detection services (like heartbeat intervals, security parameters, etc). In the model, a process can also fire its failure via client peers.

- *Rendezvous peer*: This concept is used as it is defined in [40]. It provides entry point and monitoring services.

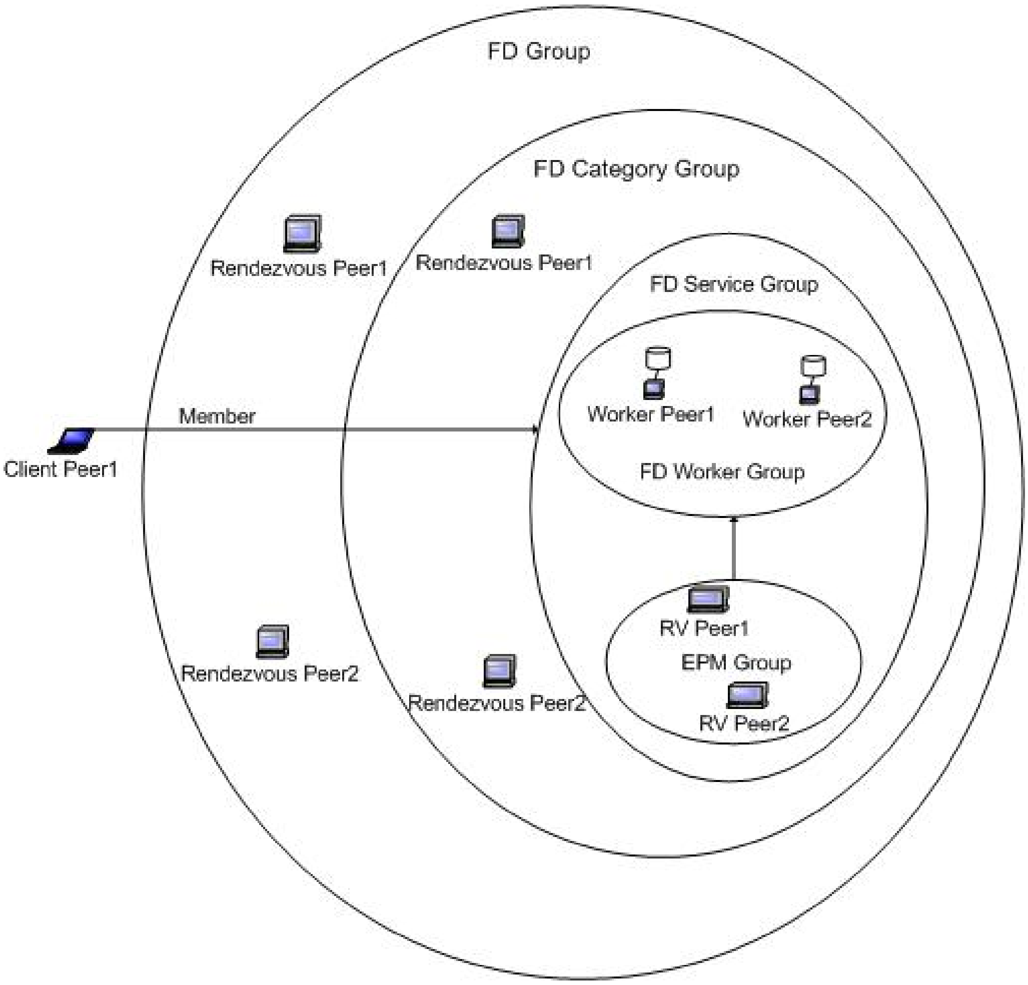


Figure 5.1 Architecture of P2P based failure detection model

(Adapted from [40])

The proposed failure detection services register themselves into related super groups / service categories / service groups where they are discovered by client peers.

5.1.2. Peer Groups

- *Worker group*: This concept is used for grouping worker peers in the scope of a service group. Each worker peer in a worker group provides the same failure detection mechanism which constitutes overall failure detection service provided by the group. Worker groups provide communication services to replicate information among their worker peers.

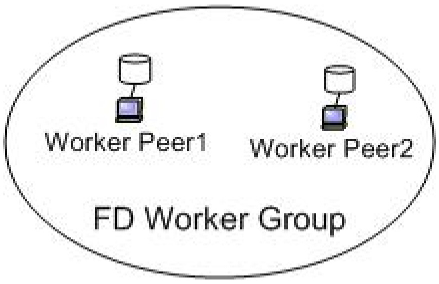


Figure 5.2 Architecture of Worker Group

- *Entry Point & Monitor (EPM) group*: This concept is used for grouping rendezvous peers in the scope of a service group. This group provides load balancing and self-healing primitives in the model. Clients are connected to best available worker group and best available worker peer in the worker group by members of this group.

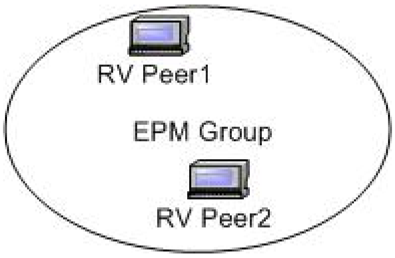


Figure 5.3 Architecture of EPM Group

- *Service group*: This concept is abstracted as a failure detection group existing for each distributed system service in the model. The service group forms a unique failure detection service access scope for a distributed system service. Client peers need to join this group to access the required failure detection service for their processes. For instance, the model in Figure 5.4 represents an activity where the failure of distributed system service A is detected by failure detection service group A (FD Service Group A). To use failure detection service A, client peers 1 and 2 which monitor health of processes 1 and 2 respectively become members of the failure detection service group A. After being a member, client peers use failure detection mechanisms provided by worker peers of failure detection worker group (FD Worker Group) in failure detection service group A.

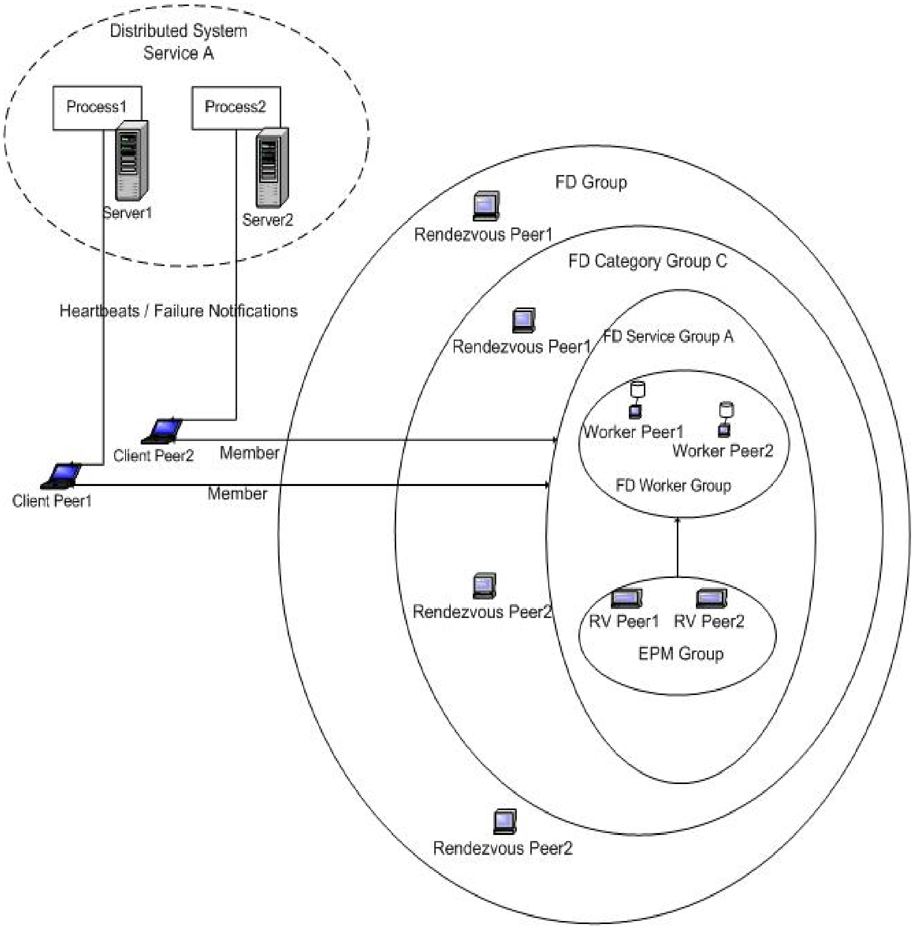


Figure 5.4 Architecture of Failure Detection Service Group

- *Category group*: This concept is used to scope search contexts for failure detection service groups. Similar failure detection services are grouped into the same category group. Figure 5.5 represents the modelling where failure detection groups for the services A and B are categorized under category C. Distributed system services A and B can be thought of as data mining and data transaction services and C as database category.

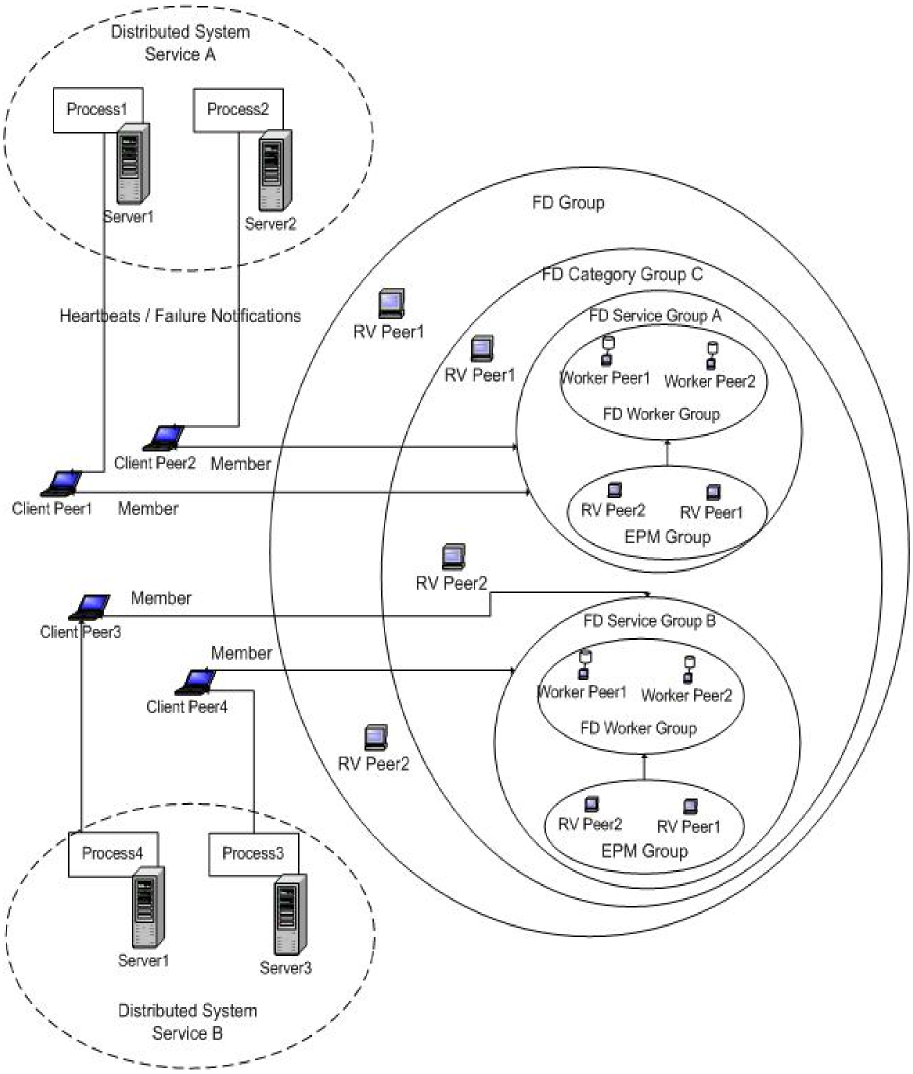


Figure 5.5 Architecture of Failure Detection Category Group

5.1.3. Services

The model includes several services to provide necessary functionality. These include both services of the reference Conscientia architecture and additionally core P2P services [40].

5.1.3.1. Conscientia Core Services

The model includes core services of Conscientia architecture: Entry Point, Monitoring service and Worker service. These services are used to control the self-healing, self-managing and load balancing functionality of the architecture.

- *Entry Point Service*: It runs in all rendezvous peers of a service group and handles all queries. Being in all rendezvous peers provides redundancy and removes hotspots in the model. When a client sends a query to a rendezvous peer, this service distributes the query to worker peers according to optimal distribution strategy. The query caching technique provided by this service ensures responding to every query even in case of selected worker peer failure which is informed by the monitoring service.

- *Monitoring service*: It has a module in each group of the system to keep a health table for each peer in the system and analyze loads; queries scheduled and network delays of the peers. It ensures core services availability with this information.

- *Worker service*: It runs on worker peers in the system and provides main services like responding to queries, sending query results, etc. It dynamically adjusts to the network and service load by automatically providing a load balancing service. The P2P based failure detection model adds a new functionality to this service to make it offer heartbeat statistics to worker peers according to service and network conditions. These statistics include new heartbeat intervals or new heartbeat arrival times.

5.1.3.2. P2P Based Services

The model is named as P2P based failure detection model since it is based on the core P2P services. Although protocols of these core services can differ from implementation to implementation, idea behind these services are generally the same. For instance:

- Communication service, discovery service and location services (resolver service) are used to map a given identifier (a unique key) to a node in P2P systems and route some content towards it.

- Membership services are used to authenticate members of peer groups which scope peers/nodes in P2P systems having separate concerns
- Security services provide secure communication primitives and access control decisions in P2P systems.

Core P2P services run on every peer of the model and management of these services are done via dedicated API calls to underlying P2P system. Figure 5.6 visualizes the relationship between the services and the peers of the model.

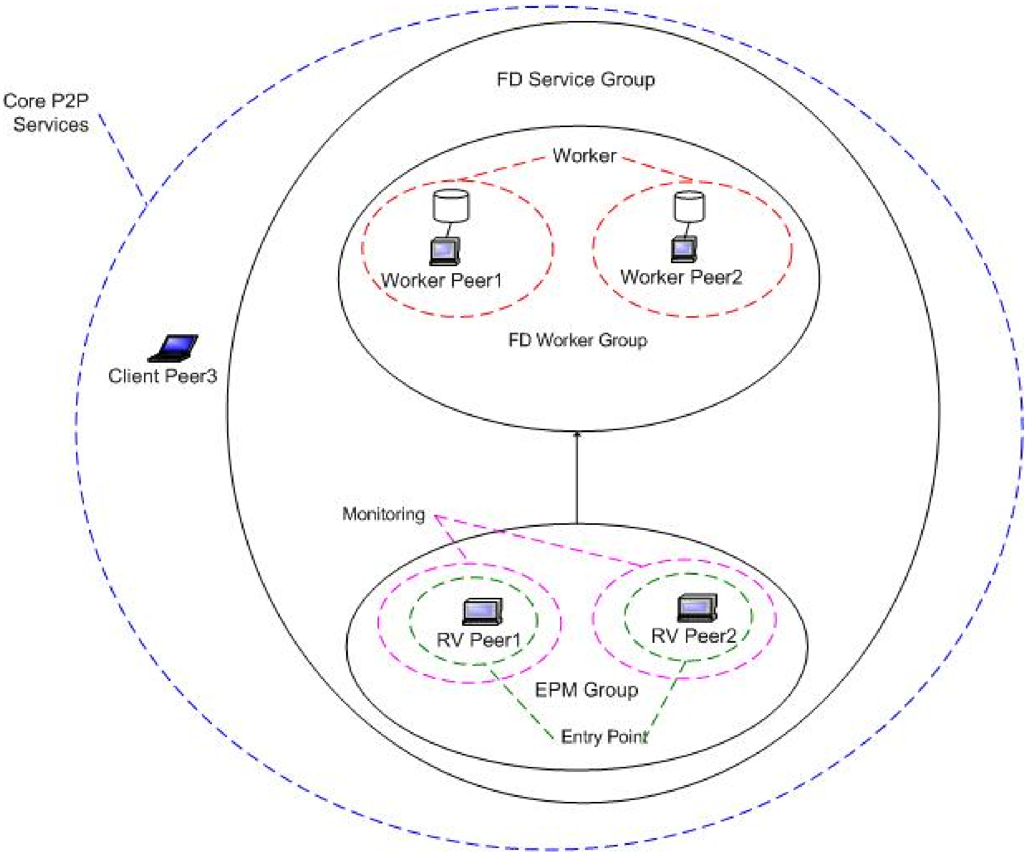


Figure 5.6 Services in P2P Based Failure Detection Model

5.1.4. Messages

The model provides two types of communication messages in the system: *failure messages* and *control messages* as shown in Figure 5.7. The model does not explicitly define structure of these messages whereas XML based semantic mark-up languages like RDF or OWL can be used to represent the semantic in failure and control messages.

- *Failure Messages*: The failure messages in the model include failure semantics like heartbeats, heartbeat queries, failure notifications, etc.
- *Control Messages*: Control messages in the model are used to send system control parameter values to client peers to autonomously tune failure detection system. These parameters include heartbeat sending interval, group keys for security in peer groups, failure detection mechanism types, etc.

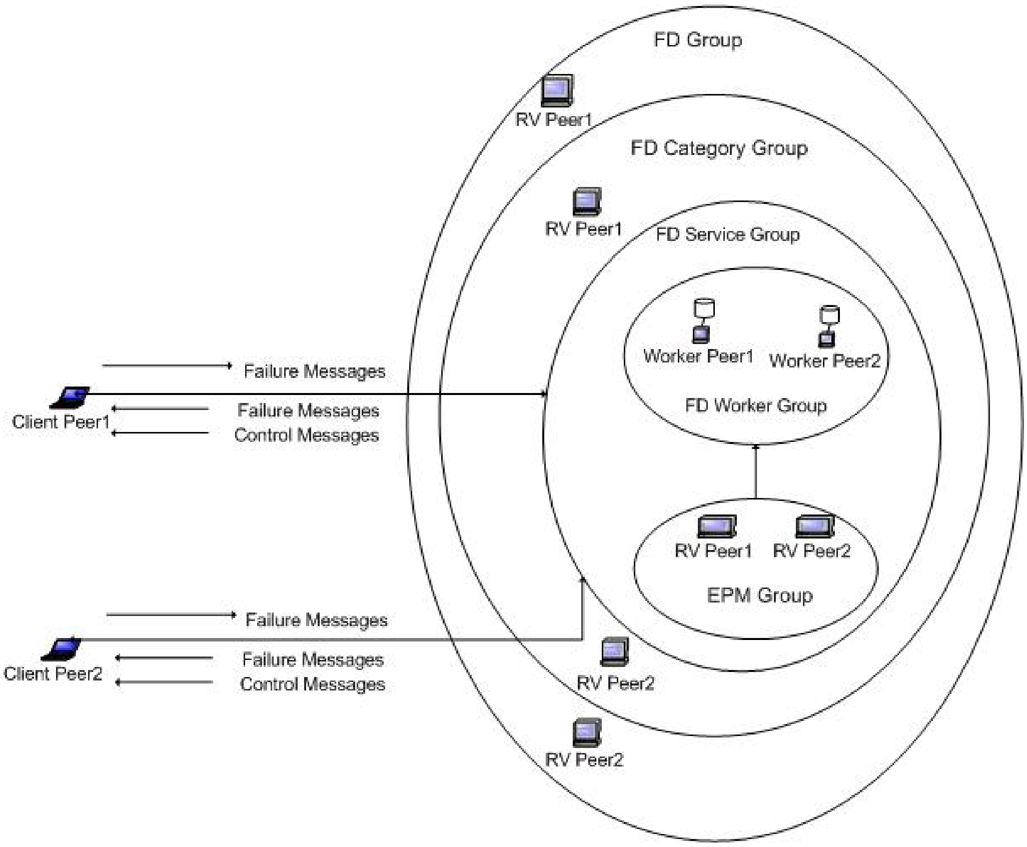


Figure 5.7 Messages in P2P Based Failure Detection Model

5.2. Analysis of the Model

The model is first analyzed according to the requirements listed for a failure detection model. Afterwards, a discussion about the model in the scope of its logical blocks is provided.

- *Scalability*: In the model, failure detection services are abstracted as localized services of a service group. The reference architecture of the model ensures scalability and fault-tolerance of these services by self-healing, self-managing and load balancing functionality of its core services: entry point, monitoring and worker services. The load and health of peers in the model is monitored constantly and an optimal selection strategy is applied which increase scalability of the services provided.

- *Adaptability*: Failure detection services should adapt itself to changing conditions of the network and service where for instance, fluctuating network traffic can cause false alarms in case of late arriving heartbeats. The offerings of the modified worker service in the model are transmitted to clients of the failure detection service by service control parameter messages. This approach provides an autonomous adaptability to network and service conditions

- *Flexibility*: Worker peers in a service group can be configured to provide different failure detection methods like unreliable failure detectors / membership services. This type of flexibility provides distributed systems to use separate failure detection methods for their services. For instance, failure detection service group for service A of the distributed system can include worker peers keeping suspect lists (unreliable failure detection) according to heartbeats received from client peers which query processes cooperating to provide service A; where as failure detection service group for service B of the same system can include worker peers keeping membership views (membership service) for the processes of service B

- *Quality of Service:* The proposed model includes two methods to provide required infrastructure for QoS aware failure detection approaches which try to keep a certain level of QoS in failure detection: tuning failure detection service parameters (heartbeat intervals, heartbeat querying/sending) and changing dissemination method of failure information.

- *Tuning failure detection service parameters*: Worker peers in the model employ different failure detection methods like, weakly or strongly consistent membership services, different type of unreliable failure detectors. These peers use worker services to get statistical information about heartbeats and tune their service parameters according to these statistics. Changes in service parameters are transmitted to client peers by control messages in the model. For instance: Worker peers of a failure detection service can change their heartbeat interval from T0 to T1 to provide certain level of QoS and transmit the new value to client peers using communication services.
- *Dissemination method of failure information*: Dissemination of failure information (suspect lists, membership lists, and failure notifications) can affect QoS of failure detection service provided. A broadcasting strategy can be preferable in small scales whereas in large scales this strategy should be replaced with a gossiping style strategy to keep QoS at a certain level. In gossiping based strategies, failure information is sent to randomly picked client peers by unicast style communication. This results in inconsistencies in failure information throughout the distributed system clients but can have a positive effect on performance. Performance of several gossip-based protocols is studied in [25]. Use of P2P based membership service provides a good means to employ gossip-based protocols where randomly picking in the protocol is done among members of this service.

5.2.1. Failure Detection Block

5.2.1.1. Determining Process Failures

5.2.1.1.1. Unreliable Failure Detectors

Unreliable failure detection can be provided in several ways by the model. Some approaches to provide unreliable failure detection in the model are listed below.

They differ in several QoS parameters like: failure detection time, query accuracy probability, etc.

1. The failure detection service uses communication services to multicast received heartbeats together with suspect lists received from a member to all other members of the group. Each member updates its suspect list itself and derives failures of the service. In this approach, however, for each heartbeat interval, the number of messages transmitted and received can go up to the order of $O(n^2)$ where n is the number of members in the group. In case of frequent heartbeat exchange, these heartbeat messages can flood network and cause poor QoS. Figure 5.8 depicts this approach where the initial numbers in the messages indicate sequence of these messages. At first step, client peer1 sends its heartbeat and suspect list to failure detection service C (provided in FD Service Group C) which are broadcasted to client peers 2, 3 and 4 respectively as a second step.

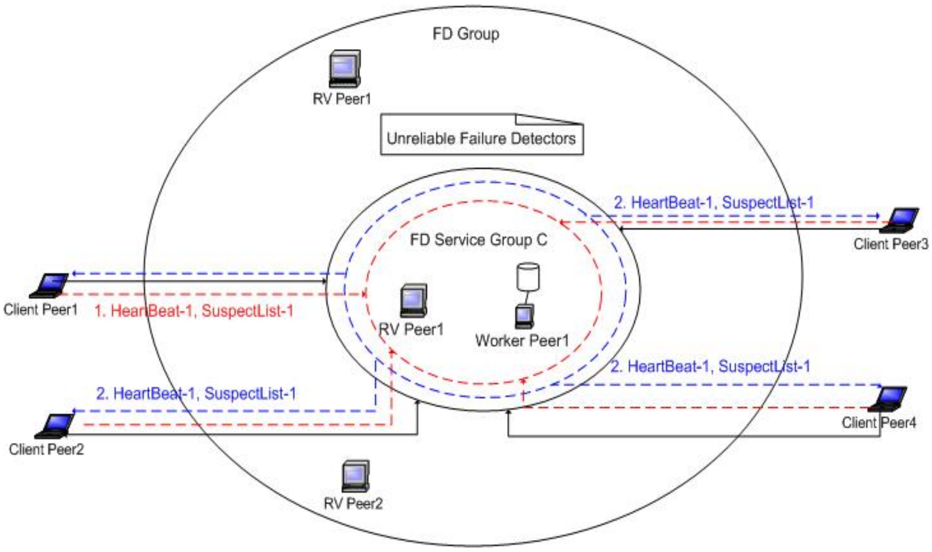

FD Group
RV Peer1
Unreliable Failure Detectors
2. HeartBeat-1, SuspectList-1
Client Peer3
FD Service Group C
Client Peer1
1. HeartBeat-1, SuspectList-1
RV Peer1
Worker Peer1
2. HeartBeat-1, SuspectList-1
2. HeartBeat-1, SuspectList-1
Client Peer4
Client Peer2
RV Peer2

Figure 5.8 Broadcast Based Unreliable Failure Detector

2. The members receive membership list from failure detection service of the group which interact with membership service to get required information. Since the architecture is P2P based, each member interacts with other

members (whose contact information is extracted from membership list) in a peer-to-peer basis and sends heartbeat and suspect lists to other members. The members derive failures of the service from these suspect lists and transmit this failure to group. This failure information is then broadcasted to all group members by worker service.

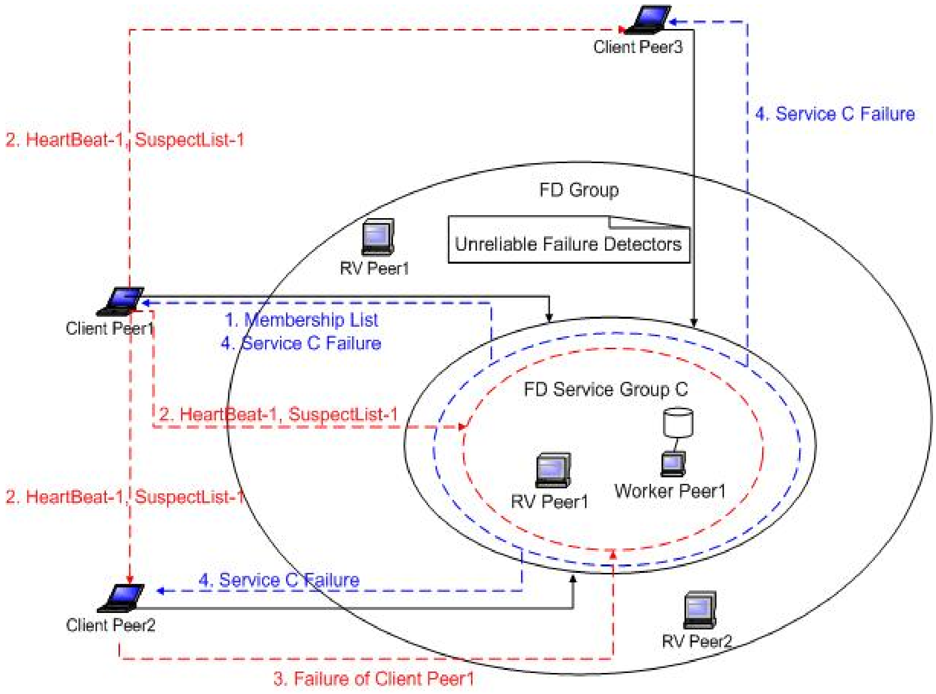


Figure 5.9 P2P Based Unreliable Failure Detector

3. The failure detection service itself updates a suspect list from received heartbeats. In case of suspected members, the service multicasts this list to members with contact information of suspected members as proposed for failure detection in the scope of grid environments in [47]. The members contact with suspected member in a peer-to-peer manner and query their aliveness. The experimental results in [47] show that, this approach provides a good solution in large scale grid environments. Although our environment is not a grid, structured P2P overlays provide a similar infrastructure for this approach.

5.2.1.1.2. Group Membership Service

In membership based failure detection, membership information which can be derived from membership services in the model is used to detect failures of services. The membership based failure detection mechanisms are deployed to worker peers as localized peer group services, interact with membership service of the group and inform members of the group about the failures of the service using existing communication services.

In this type of failure detection, P2P approach is very useful in the sense that most P2P implementations provide an inherent membership messaging in their protocols. Therefore, no additional heartbeat messaging is necessary.

Weakly and strongly consistent membership services can be provided in several ways in the model. Basic approaches to provide group membership services for failure detection in the model are listed below:

1. Membership service of most P2P implementations is a weakly consistent membership service by default. To improve scalability, membership service of a peer group generally does not employ heartbeat based strategies to verify that its members are live which result in erroneous information about members of the group. To provide a weakly consistent membership service, it is enough for failure detection service to use inherent P2P based membership service and communication services of the group to inform members about membership failures.
2. To provide stronger semantics, an additional heartbeat/membership view messaging can be provided by the model. In this approach, failure detection service of the model itself provides a membership service with stronger semantics by analyzing heartbeat messages received from members and if any failures are detected notifies members of the group with its current membership view. Figure 5.10 in the next page depicts this approach.

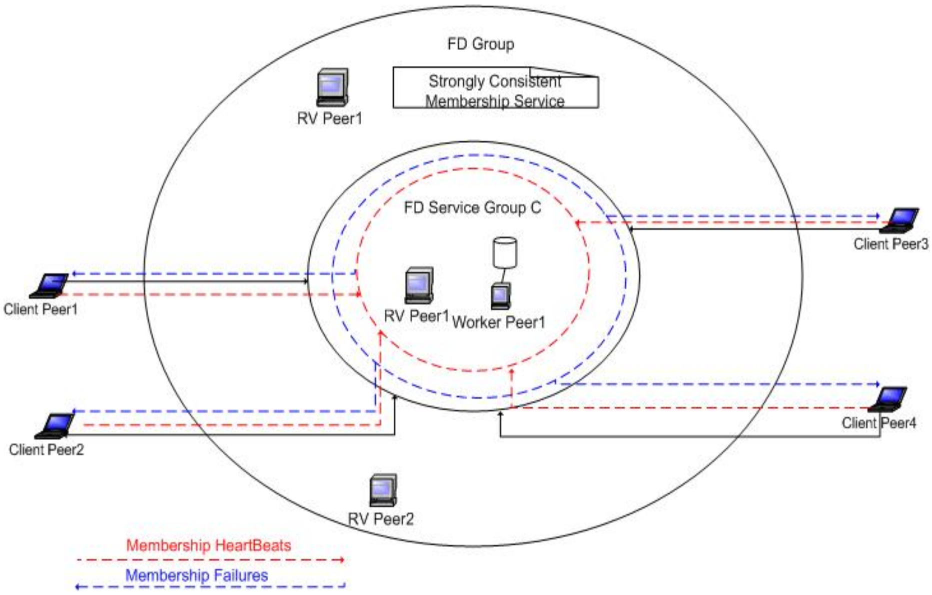


Figure 5.10 Membership Based Failure Detection Service

5.2.1.2. Managing Failure Detection Service

- *Security*: Security is ensured by the core P2P services in the model which provides secure communication and fine-grained access control decisions (like authorization, etc) in the scope of peer groups to prevent malicious attacks (capture and replay / denial of service) or performance degrading caused by malicious peers or applications. The motivation about securing peer groups is well addressed in [22]. Access control policies provide authorization control at the group level and prevent authenticated peers to falsify system. Independent peer group services in the model enable separate admission control techniques [42] to be implemented at each service group for different security requirements of system services. For instance, for a distributed system service which is provided by only 2 processes (and surely 2 client peers in the model), a static key exchange protocol and an Access Control Lists (ACL) based admission control policy is fairly enough. However, for a service which has many processes cooperating to provide the service (and so many numbers of clients in the failure detection model) and dynamically leaving/joining failure detection service groups, a dynamic key exchange protocol [41] with a Group Authority (GAUTH) based admission control policy is necessary. The transmission of security parameters like group charter and certificate are possible via control messages in the model.

- *Dependency Analysis*: The P2P based failure detection model uses a similar approach with [6], to resolve service dependencies in the distributed system. This approach requires failure detection service group of a distributed system service to be an inactive member of the failure detection service group of the service it depends on. Inactive members just receive messages (failure notifications) from the group. Figure 5.11 represents this approach.

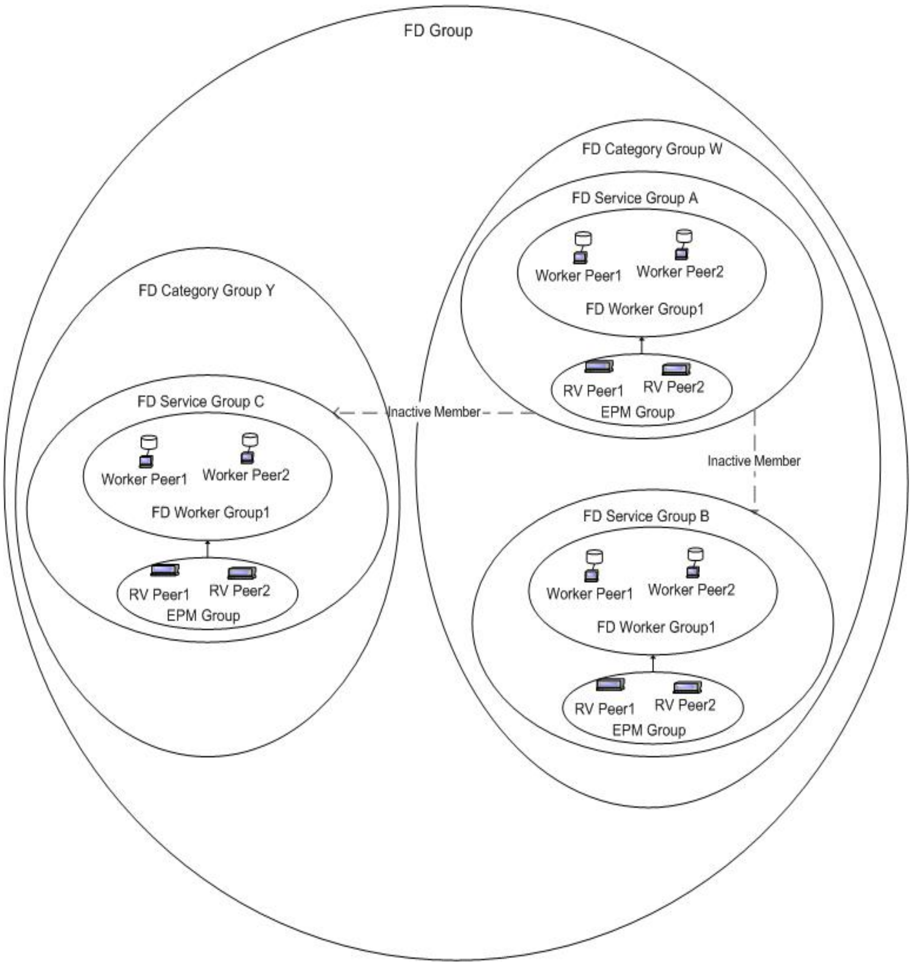


Figure 5.11 Inactive Members to Resolve Service Dependencies

In the figure, there exists a distributed system which has three services A, B and C where A has a dependency on both B and C. Failure of services B and/or C implicitly means that service A also fails. In the model, this dependency is resolved by making failure detection peer group service of A to be an inactive member of peer group of B and C (represented as dotted line). This way, failure detection service for A is notified by failure of services B or C.

5.2.1.3. Evaluation

Table 1 presents the failure detection block characteristics of the related studies and the proposed model.

Table 1 Comparison of Failure Detection Block Characteristics

	FUSE	**Failure Detection Service API**	**Proposed Model**
Scalability	Yes	Yes	Yes
Adaptability	No	Yes	Yes
Flexibility	No	No	Yes
Quality of Service	Failure notifications in bounded period of time.	Yes	Yes
Security	Addresses only secure implementation alternatives.	No	Yes
Dependency Analysis	Manually Setup	Manually Setup	Autonomous

It can be seen from Table 1 that the proposed model provides better flexibility, security and dependency resolution than its rivals. The main feature of the proposed model, which put it one step forward from its rivals, is that it takes advantage of using dedicated failure detection services (in scope of peer groups) for each distributed system service. Separate peer groups enable specialized failure detection services to be deployed for each distributed system service which in turn provides flexibility. The membership services in peer groups provide a suitable environment for both applying security policies which increase security in the system and setting up inactive membership concept for autonomous dependency analysis.

5.2.2. Communication Block

The P2P based failure detection model is based on P2P communication services. P2P communication services provide a good QoS for the underlying communication channel because of their advanced routing and fault tolerant mechanisms. The necessary communication for a failure detection block to work properly is in two forms: Broadcasting and Unicasting.

5.2.2.1. Multicast Communication

Broadcasting is handled by the multicast service provided in P2P based communication services. The P2P based communication services provide multicast communication services for group communication which is visualized in Figure 5.12. This multicasting service should not be confused with TCP/IP multicasting which is blocked while penetrating through corporate networks / internet service providers / firewalls / NAT. P2P technology provides us with several mechanisms (like peer discovery, relay peers, etc) to broadcast a message to multiple listeners in different networks. Since P2P protocols generally run in application level, this type of multicasting is referred to as application level multicasting where broadcasting a message can include both multicast and unicast approaches. In the P2P based failure detection model, application level multicasting service enables peers to be in separate networks and even behind firewalls.

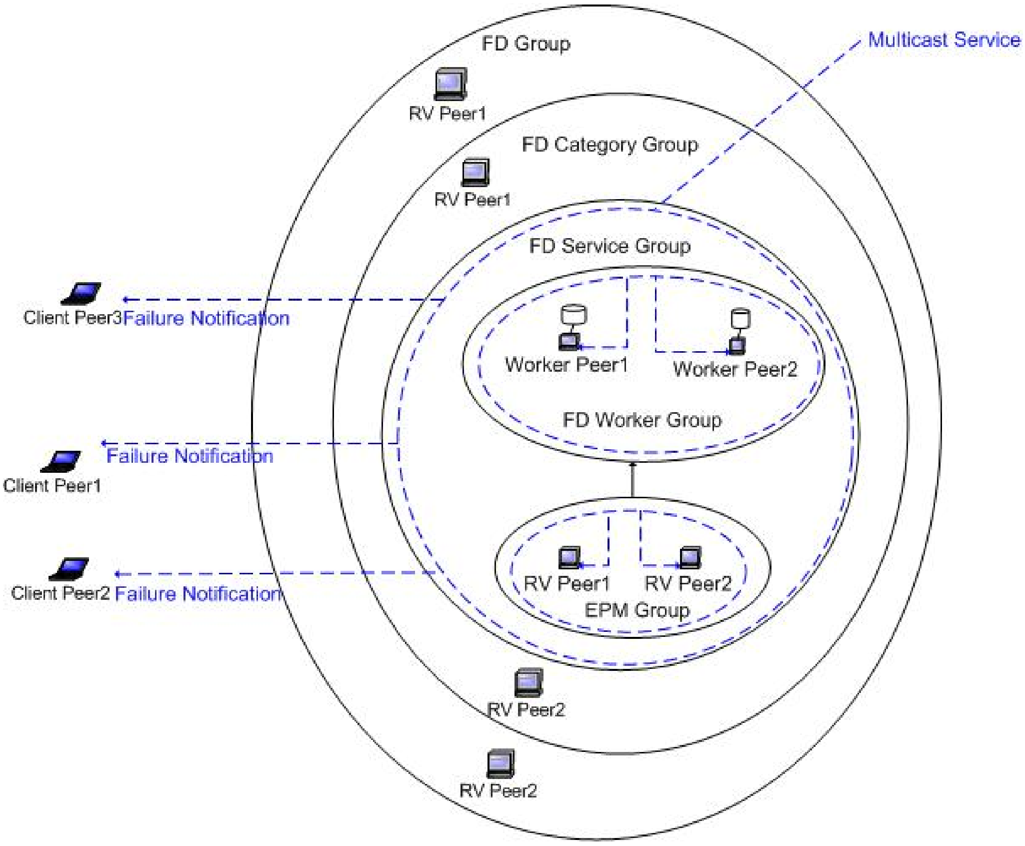


Figure 5.12 Multicast Service in P2P Based Failure Detection Model

5.2.2.2. Point-to-Point Communication

Point-to-point communication is implemented by the unicast services provided in P2P based communication services. The unicast communication in P2P communication services provides peer-to-peer communication functionality. P2P provides several mechanisms (like discovery services / resolver services) to enhance its abstraction from IP / DNS naming based addressing. This enables peers to send messages without knowing IP's of target peers.

5.2.2.3. Adaptability to Underlying Network Conditions

Core P2P communication services provide an overlay network which abstracts them from physical network conditions and failures. The model does not specify topology of underlying P2P network since all P2P networks provide a certain level of fault-tolerance for underlying network failures. Resources at [51][52][53] discuss the level of fault-tolerance in routing/communication in P2P networks having different topologies.

5.2.2.4. Evaluation

Table 2 presents the communication block characteristics of the related studies and the proposed model.

Table 2 Comparison of Communication Block Characteristics

	FUSE	**Failure Detection Service API**	**Proposed Model**
Low Communication Overhead	Yes (P2P based overlay network requires no additional liveness-verifying traffic beyond that already needed to maintain the overlay)	Implementation Dependent	Yes (Based on membership service of the underlying P2P system).
Adaptability to different topologies in the network	Yes (P2P based overlay network)	No (Does not support application level multicasting)	Yes (P2P based overlay network)
Adaptability to dynamism of the network	Yes (P2P based overlay network)	No (Vulnerable to underlying physical network failures)	Yes (P2P based overlay network)

Using P2P based overlay networks provide great advantages in adapting to underlying physical networks as can be seen from Table 2. Communication blocks of both the proposed model and FUSE satisfy required functionality in adapting to underlying network by taking advantage of using P2P based overlay networks.

CHAPTER 6

REALIZATION OF P2P BASED FAILURE DETECTION MODEL

In this chapter, realization of the proposed model is explained together with a sample proof-of-concept implementation which aims to provide a small-scaled failure detection framework using the proposed P2P based failure detection model.

6.1 Realization

6.1.1. Interaction with Distributed System Processes

The implementation of the model (which provides a failure detection framework for distributed systems) should provide necessary interfaces to distributed system processes to notify about failures / to query heartbeats, etc. Type of interfaces (inter-process communication, socket interfaces, programming level interfaces) between failure detection framework and system processes can vary according to packaging of the implementation. Several alternative approaches can be used to package and deploy the implementation (such as a middleware [16][46], a user space library, an OS level service, etc) whereas it is not in the scope of this thesis to discuss advantages/disadvantages of these alternatives.

The sample implementation provided in this thesis is packaged in a user space library (FD library) to remove extra communication overhead between processes of distributed system and failure detection framework. Every process in the distributed system should load an instance of the library and implement required interfaces for communication to use the failure detection framework. Figure 6.1 in the next page represents the transformation of processes through the FD library.

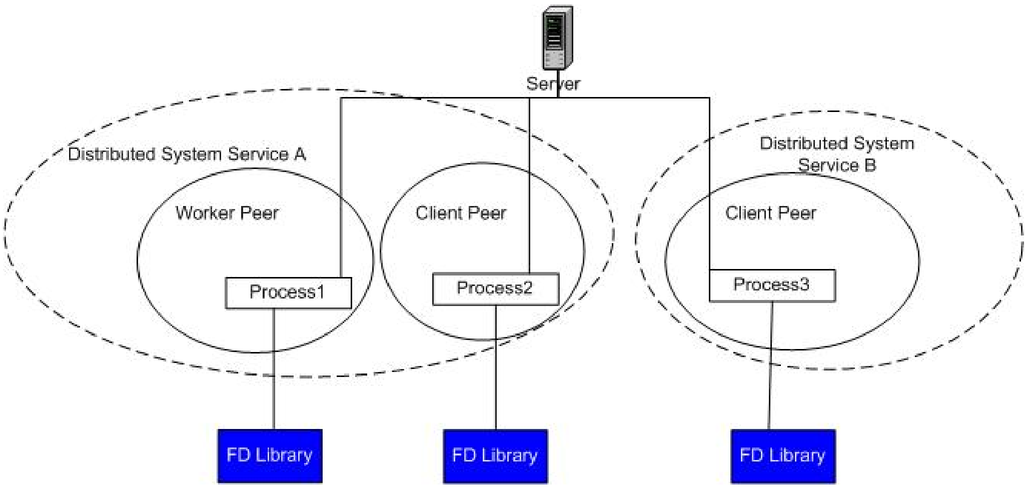


Figure 6.1 Failure Detection Library Mapping Processes to Peers

6.1.2. P2P Platform

The model is based on core P2P Services. Therefore, it should be implemented on a P2P framework supporting these services. Such a framework can be formed by:

- Building efficient infrastructures on top of existing P2P based overlay networks. Such infrastructures enable these P2P overlays with efficient querying [45], application level multicasting [50][57] and security [54]][56] functionality. For instance:
  - In [44], authors build their massively multiplayer game applications on Pastry overlay network donated with multicast infrastructure proposed in [50] for efficient transmission of game states to players.
  - In [7], a distributed information management system, XDM, is proposed which enhances its P2P overlay with data management, topic-based publish/subscribe communication and query processing capabilities.

- Using a comprehensive P2P framework. The open-source Project JXTA [55] is a comprehensive peer-to-peer framework originally conceived by Sun Microsystems Inc. Project JXTA is selected as the P2P backbone of reference implementation of the model because it provides a virtual network overlay and:

- Provides and standardizes core P2P protocols,
- Provides required semantic to manage P2P networks,
- Provides required semantic to enable peers to manage peer groups,
- Provides a platform independent of programming languages (such as C or the Java programming language), system platforms (such as the Microsoft Windows and UNIX operating systems) and network protocols (such as TCP/IP or Bluetooth). The Project JXTA protocols can be implemented on any device having a network heartbeat, including sensors, appliances, network routers, desktop computers, and storage systems which make it suitable [43].

6.2 Sample Implementation

The sample implementation aims to detect service failures of small-scaled distributed systems using the proposed P2P based failure detection model. The implementation is able to detect failures of services of a distributed system by applying different types of failure detection methods and service dependency resolution.

6.2.1. Constraints & Assumptions

The implementation is a proof-of-concept implementation of the model and does not aim to provide high availability of failure detection services in large scales and high load conditions where setting up test and simulation environments for that case is a complex job.

The sample implementation provides necessary failure detection and management APIs for worker peers, rendezvous peers and client peers of the model. Although several approaches (as a layered middleware [16], OS service, a user space library) can be used to package and deploy the API, it is not in the scope of this thesis to discuss pros and cons of these approaches.

Implementation of Conscientia architecture was unavailable at the time of the development phase of this sample; therefore, functionality of its services which provide scalability and adaptability is coded in a degraded manner in the implementation:

- The sample implementation employs a single RV peer per group instead of an EPM group as in the model.

- The sample implementation employs a single worker peer per group instead of worker groups as in the model.

- The sample implementation does not employ category groups as in the model.

- The sample implementation does not employ point-to-point communication.

- The sample implementation does not support monitoring service. However, this service can be easily implemented using peer info service of JXTA for monitoring and metering [58].

6.2.2. Failure Detection Library

The implemented failure detection library provides necessary functionality for the proposed failure detection model. This functionality is wrapped by a shell which provides functions as API interfaces to library users (processes). Necessary JXTA calls for these API interfaces are provided in Appendix – C.

6.2.2.1. Worker Peer API

Failure detection library enables processes to act as worker peers within the failure detection model. With the worker peer API provided, worker peers manage failure detection groups for services of distributed system and send/receive failure/control messages. The management of failure detection groups include creating failure detection groups, defining security levels, identifying failure detection methods, deciding the dissemination of failure information. These processes can either be distributed system processes (processes cooperating to provide distributed system services) or separate processes dedicated for failure detection purposes.

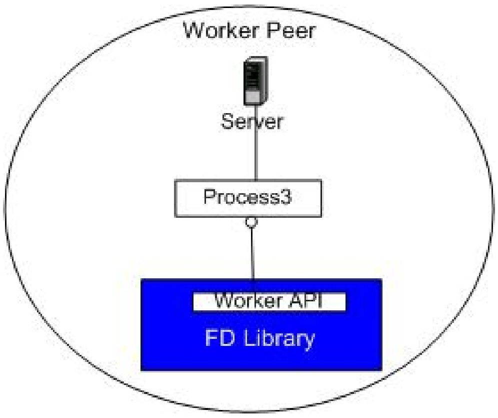


Figure 6.2 Worker Peer API Transforms Processes to Worker Peers

In Figure 6.2, the worker API enables a failure detection process to act as a worker peer in the failure detection model. Creation of failure detection services using this API is represented in Figure 6.3.

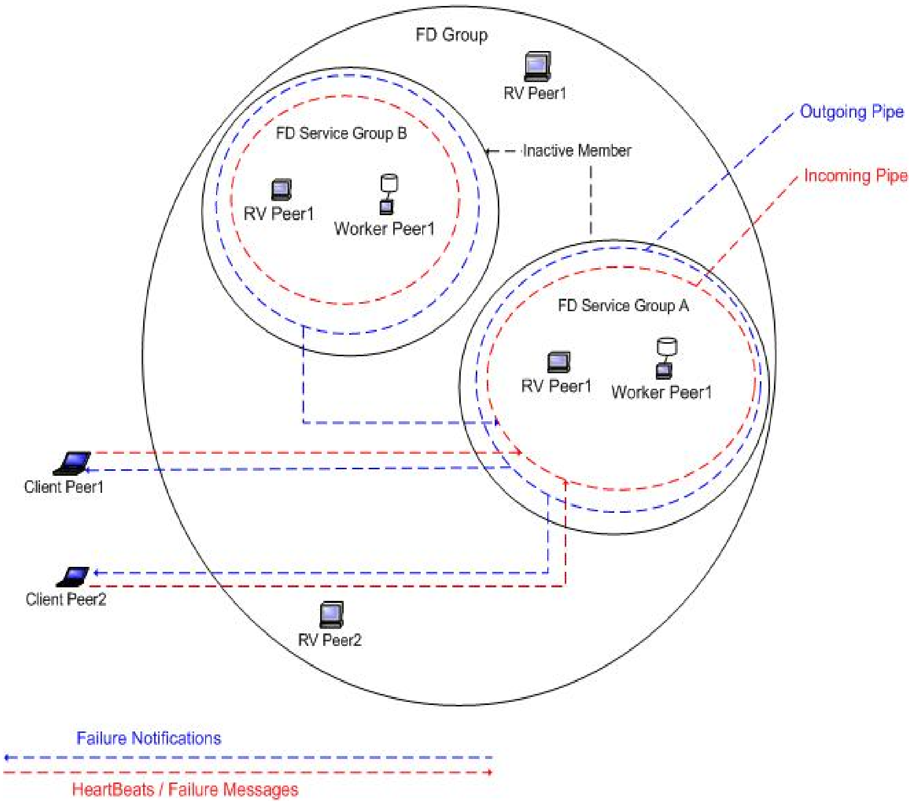


Figure 6.3 Creation of a Failure Detection Service in FD Library

To create a failure detection service in the model, certain steps should be followed. The steps required to create a failure detection service are presented below:

1. **create_FD_group**(groupName): In JXTA, creation of a group is done by preparing the advertisement for the group and publishing it to the discovery service through which client peers can search for.
2. **create_FD_services**(groupName, serviceName, failureDetectionType): Failure Detection services are provided as localized services in a peer group. Creating services for a peer group is done by first creating/publishing of module class/spec advertisements of the service to the discovery service of the target peer group and then creating communication channels (pipes) for the service using the pipe service of the target peer group.

   Although not implemented in this sample, in the scope of a failure detection service, module and specification advertisements for a failure detection service can be used to carry control messages specifying the published failure detection service type and parameters.

   The pipes specify input/output interfaces for a service. For a failure detection service, two pipes should be deployed: an incoming pipe and an outgoing pipe. An incoming pipe provides input for the service whereas an outgoing pipe outputs results of the service. In the scope of the model, failure messages (heartbeats stemming from client peers, failure messages from peer groups of dependent services) are received through incoming pipes and control messages and detected failures are sent through outgoing pipes. Types of incoming and outgoing pipes created for failure detection services in the model are *propagate* types in Project JXTA which provides a broadcast style communication. It connects one output pipe to multiple input pipes.

   After creating the service by specifying its input/output interfaces as pipes, failure detection mechanisms are implemented in this service according to *failureDetectionType* parameter.

3. **while**(FD_group(groupName) **dependsOn** FD_group(serviceGroups))
4. **join_as_Inactive_Member**(serviceGroups): An inactive member joins to a target peer group and uses only outgoing pipe of the failure detection service

declared in that group. This way, the inactive member is only notified with outgoing pipe messages (service control messages and detected failures) of dependent failure detection services.

In these steps, incoming pipe of failure detection service declared at step 2 is connected to outgoing pipes of dependent failure detection services. The dependency relation between distributed system services (and failure detection services) can be either retrieved from static configuration files or from service advertisements.

6.2.2.2. Client Peer API

Failure detection library enables processes of distributed system services to act as client peers as represented in Figure 6.4 in the next page. With the client peer API provided, client peers can join failure detection service groups, discover localized failure detection services for service groups, and join their pipes for communication with these services.

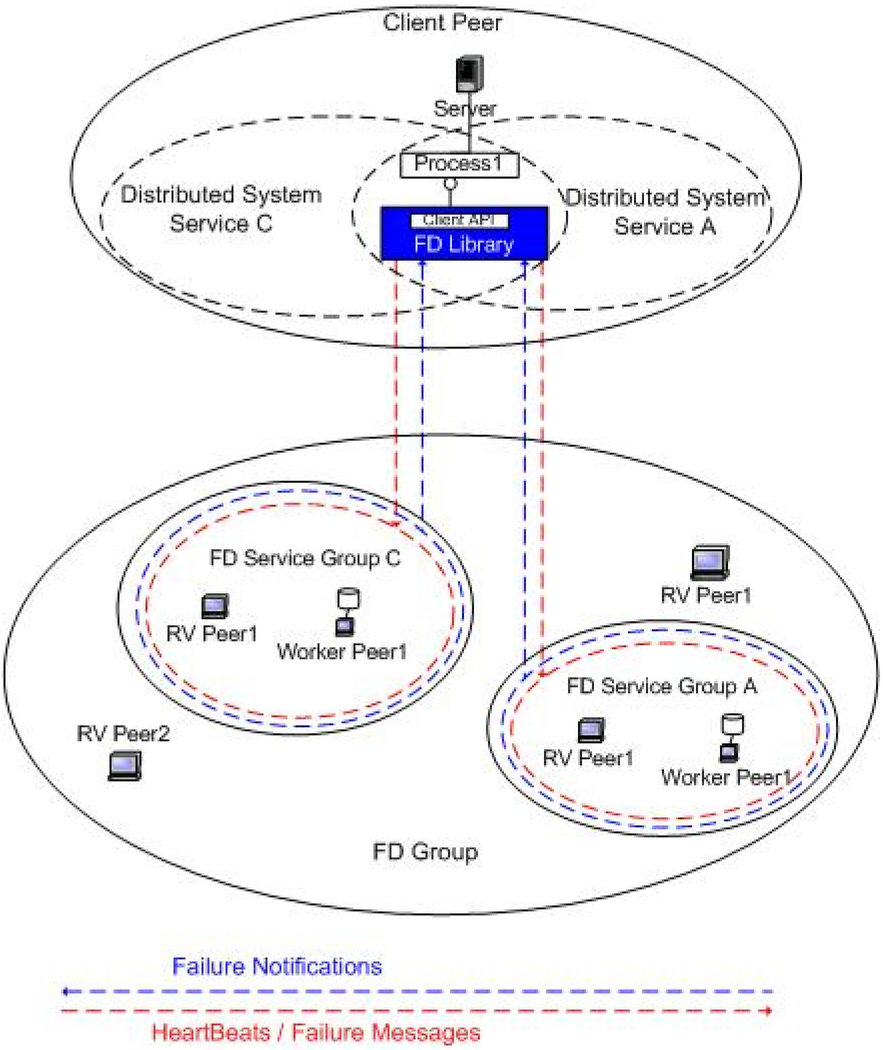


Figure 6.4 Client Peer API Transforms Processes to Client Peers

In the Figure 6.4, process$_1$ of a distributed system provides necessary functionality (probably cooperating with other processes) for distributed system services A and C. The client API enables this process to join to failure detection groups created to detect failures of system services A and C, use failure detection services provided in these groups. Using failure detection services for a group includes providing required behaviour for the service which includes obeying service requirements (like heartbeat intervals, failure detection types) received through control messages. Below is the algorithm for joining a failure detection service in the library:

1. **discover_FD_group**(groupName): In JXTA, discovery of a group is done by first sending group discovery messages using discovery service and afterwards locating required group by its advertisement.
2. **join_ as_Active_Member** (groupName): An active member joins to a target peer group and uses both incoming and outgoing pipes of the failure detection service declared in that group.

6.2.2.3. Rendezvous Peer API

Failure detection library enables processes to act as rendezvous peers as shown in Figure 6.5. There is no specific API function for this interface in the library. Processes that load failure detection library and do not create/join failure detection groups, automatically become a rendezvous peer.

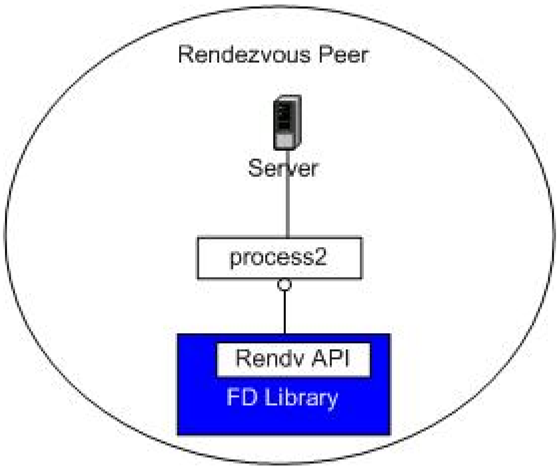


Figure 6.5 Rendezvous Peer API Transforms Processes to Rendezvous Peers

CHAPTER 7

CONCLUSION

In this chapter, conclusions and further work for this study is presented. Conclusions include the conducted work together with the comments on this work.

7.1. Work Conducted

In this thesis, requirements of a comprehensive failure detection model for asynchronous distributed systems have been collected from corresponding studies in literature. To satisfy these requirements, logical blocks are defined which cooperate to provide necessary functionality. Existing failure detection frameworks have been analyzed in the scope of these logical blocks.

A P2P based failure detection model is presented to address service failures in asynchronous distributed systems. In the model, failure detection services are abstracted as lightweight peer group services and dependency between services is resolved by the peer group membership concept in P2P. Analysis of this model is presented in the presence of requirements and logical blocks of a comprehensive failure detection model.

Evaluation of the model is presented. The model satisfies functionality provided by its competitors. Furthermore, the model outperforms its competitors in flexibility, autonomous service dependency resolution and security areas.

A failure detection library is developed to realize the model using JXTA P2P framework. Developed failure detection library is able to support a subset of functionality required for a comprehensive failure detection framework. Realization discussions on implementing the model are included as a guidance material.

7.2. Comments

Although the study provides a model to detect service failures in asynchronous distributed systems, it does not define in what situations failure detection is needed or should be done. The considerations about failure detection need and choice of failure detection method is left to developers.

Provided failure detection model defines several approaches to implement failure detection mechanisms by using P2P technologies. However, these are not the only approaches that can be implemented using P2P technology. Developers or system analysts can implement their own specific approaches based on P2P concepts that are not mentioned in this thesis, to satisfy required functionality for their needs.

Although developed failure detection library has the mentioned limitations, it is a valuable source of information for future studies addressing performance of the model, P2P techniques, efficiency of failure detection mechanisms, effect of choice of deployment type on performance, etc.

7.3. Future Work

The failure detection library developed to realize the presented P2P based failure detection model in this thesis uses an unstructured overlay network (based on JXTA overlay network), provides sample failure detection mechanisms and is designed as a user space library to be used in small scales. As a future study, performance analysis of this library can be performed.

Studies addressing to perform QoS analysis of separate implementations of the model in different scales, with different failure detection mechanism implementations, using different P2P technologies (using structured, unstructured P2P overlay networks) and choice of deployment types (as a middleware, as a user space library, etc) can guide the implementation of efficient failure detection frameworks.

REFERENCES

[1] Olivia Das, C. M. Woodside. “Failure Detection and Recovery Modelling for Multi-layered Service Systems”. In Proc. of PMMCCS-5, Erlangen, September 2001.

[2] Florin Sultan, Aniruddha Bohra, Yufei Pan, Stephen Smaldone, Iulian Neamtiu, Pascal Gallard, and Liviu Iftode. “Non-intrusive Failure Detection and Recovery for Internet Services Using Backdoors”. Technical Report, URL: http://citeseer.ist.psu.edu/633879.html, June 2006.

[3] Markus Oliver Junginger, Yugyung Lee. “A Self-Organizing Publish/Subscribe Middleware for Dynamic Peer-to-Peer Networks”. IEEE Network, pp. 4-6, January/February 2004.

[4] Andreas Binzenh¨ofer, Kurt Tutschku, Bjorn auf dem Graben, Markus Fiedler, and Patrik Carlsson. “A P2P-based Framework for distributed network management”. University of Wurzburg Institute of Computer Science Research Report Series, Report No: 351, January 2005.

[5] James Walkerdine, Lee Melville, Ian Sommerville. “Dependability within Peer-to-Peer Systems”. Workshop on Architecting Dependable Systems, ICSE 2004, May 2004.

[6] Sentinel Project. URL: http://sentinel.jxta.org, June 2006.

[7] DELIS Project. “Requirements for a P2P platform for dynamic management of large scale networks”. Deliverable D2.3.1, February 2005.

[8] John Dunagan, Nicholas J. A. Harvey, Marvin Theimer, Michael B. Jones, Alec Wolman, and Dejan Kostic. “FUSE: Lightweight Guaranteed Distributed Failure Notification”. In Proc. of OSDI 2004, December 2004.

[9] Luis Felipe Cabrera, Michael B. Jones, Marvin Theimer. "Herald: Achieving a Global Event Notification Service". In Proc. of 8th workshop on Hot Topics in Operating Systems, Elmau, German, May 2001.

[10] Armando Fox, Emre Kıcıman, David Patterson, Randy Katz, Michael Jordan, and Ion Stoica. "Statistical Monitoring + Predictable Recovery = Self-*". ACM, ISSN: 1-58113-989-6/04/0010, 2004.

[11] Tushar Deepak Chandra and Sam Toueg. "Unreliable Failure Detectors for Reliable Distributed Systems". Journal of the Association for Computing Machinery, Vol. 43, No. 2, March 1996.

[12] Xavier Défago, Naohiro Hayashibara, and Takuya Katayama. "On the Design of a Failure Detection Service for Large-Scale Distributed Systems". In Proc. of Int'l. Symp. Towards Peta-Bit Ultra-Networks (PBit 2003), pp.88–95, Ishikawa, Japan, Sept. 2003.

[13] Krishnakanth Sistla, Alan D. George, and Robert W. Todd. "Experimental Analysis of a Gossip-based Service for Scalable, Distributed Failure Detection and Consensus". Cluster Computing 6, Volume 6, Issue 3, pp.237-251, July 2003.

[14] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems". In Proc. of International Conference on Distributed Computing System, 1999.

[15] Roberto BALDONI, Fabio ZITO. "Designing a Service of Failure Detection in Asynchronous Distributed Systems". IEEE, ISBN 0-7695-1089-2/01, 2001.

[16] Bruno G. Catăo , Francisco V. Brasileiro ,and Ana Cristina A. Oliveira. "Engineering a Failure Detection Service for Widely Distributed Systems". In Proc. of SBRC'05, Brazil, 2005.

[17] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. "Fundamental Concepts of Dependability". In Third Information Survivability Workshop, October, 2000.

[18] W. Vogels and C. Re. "WS-Membership - Failure Management in a Web-Services World". In Intl. World Wide WebConference (WWW), 2003.

[19] R. van Renesse, Y. Minsky, and M. Hayden. "A Gossip-Style Failure Detection Service". In Proc. Of Middleware'98, pp.55-70, IFIP, The Lake District, UK, 1998.

[20] Antony Rowstron1 and Peter Druschel. "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems". In Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Heidelberg, Germany, November 2001.

[21] Indranil Gupta, Robbert van Renesse, and Kenneth P. Birman. "Scalable Fault-Tolerant Aggregation in Large Process Groups". IEEE, ISBN 0-7695-1101-5/0, 2001.

[22] Zupeng Li, Yuguo Dong, Lei Zhuang, and Jianhua Huang. "Implementation of Secure Peer Group in Peer-to-Peer Network". In Proc. of ICCT, 2003.

[23] André Schiper. "Failure Detection vs Group Membership in Fault-Tolerant Distributed Systems: Hidden Trade-Offs". In Proc. Of PAPM-ProbMiv'02, LNCS 2399, 2002.

[24] B. Pourebrahimi K. Bertels S. Vassiliadis. "A Survey of Peer-to-Peer Networks". Proc. of 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc2005, November 2005.

[25] Mark W. Burns, Alan D. George, and Brad A. Wallace. "Simulative Performance Analysis of Gossip Failure Detection for Scalable Distributed Systems". High-performance Computing and Simulation (HCS) Research Laboratory, Department of Electrical and Computer Engineering, University of Florida, 2000.

[26] Aleta M.Ricciardi and Kenneth P.Birman. "Using Process Groups to Implement Failure Detection in Asynchronous Environments". ACM, ISBN 0-89791-439-2/91/0007/0341, 1991.

[27] Mishra, L. L. Peterson, and R. D. Schlichting. "A Membership Protocol Based on Partial Order". Technical report, University of Arizona, 1990.

[28] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. “Processor Membership in Asynchronous Distributed Systems”. University of California at Santa Barbara, Extended Abstract, 1990.

[29] Richard Andrew Golding. “Weak-consistency group communication and membership”. PhD Thesis, University of California Santa Cruz, December 1992.

[30] Cristina Nita-Rotaru. “The Cost of Adding Security Services to Group Communication Systems”. Technical Report, CNDS-2000-3, Computer Science Department, John Hopkins University, March 2000.

[31] Yair Amir, Cristina Nita-Rotaru, Jonathan Stanton, and Gene Tsudik. “Scaling Secure Group Communication Systems: Beyond Peer-to-Peer”. In Proc. of DISCEX3, Washington DC, April 2003.

[32] Nicholas J.A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. “SkipNet: A Scalable Overlay Network with Practical Locality Properties”. In Proc. of 4th USENIX Symposium on Internet Technologies and Systems, March 2003.

[33] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. “Tapestry: An Infrastructure for Fault-tolerant Wide Area Location and Routing”. Report No. UCB/CSD-01-1141, Computer Science Department, U.C. Berkeley, April 2001

[34] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. “Chord: A Scalable Peertopeer Lookup Service for Internet Applications”. ACM, ISBN 1581134118/01/0008, 2001

[35] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. “A Scalable Content-Addressable Network”. Proc. of SIGCOMM’01 ACM Conference on Applications, Technologies Architectures, and Protocols for Computer Communication, August 2001.

[36] L.E.Moser, P.M.Melliar Smith, D.A.Agarwal, R.K.Budhia, and C.A.Lingley-Papadopoulos. “Totem: A Fault Tolerant Multicast Group Communication System”. Communications of the ACM, Volume:9, Number:4, pp.54-63, 1996.

[37] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. "Transis: A Communication Sub-System for High Availability". In Proc. of 22nd International Symposium on Fault-Tolerant Computing, pp.76-84, IEEE Computer Society Press, July 1992.

[38] Robbert Van Renesse, Takako M.Hickey, and Kenneth P.Birman. "Design and Performance of Horus: A Lightweight Group Communications System". Technical Report, Department of Computer Science, TR94-1442, Cornell University, Ithaca, NY, 1994.

[39] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera. "On the Quality of Service of Failure Detectors". IEEE, ISBN 0018-9340/02, 2002.

[40] Muhammad Asif Jan, Fahd Ali Zahid, Mohammad Moazam Fraz, and Arshad Ali. "Exploiting peer group concept for adaptive and highly available services". In Computing in High Energy and Nuclear Physics, La Jolla California, 24-28 March 2003.

[41] M. Steiner, G. Tsudik, and M. Waidner. "Key agreement in dynamic peer groups". IEEE Transactions on Parallel and Distributed Systems, Volume:11, Number:8, pp.769-780, August 2000.

[42] Yongdae Kim, Daniele Mazzocchi, and Gene Tsudik. "Admission Control in Peer Groups". In IEEE NCA, 2003.

[43] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. "Project JXTA 2.0 Super-Peer Virtual Network". Sun Microsystems Inc., URL: www.jxta.org/docs/, May 2006.

[44] Björn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. "Peer-to-Peer Support for Massively Multiplayer Games". In Proc. of INFOCOM'04, 2004.

[45] Ryan Huebsch, Brent Chun, Joseph M. Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica and Aydan R. Yumerefendi. "The Architecture of PIER: an Internet-Scale Query Processor". In Proc. of the 2005 CIDR Conference, 2005.

[46] Jan Gerke, and Burkhard Stiller. "A Service-Oriented Peer-to-Peer Middleware., In 14. Fachtagung Komminikationen Verteilten Systemen 2005 (KiVS05), LNCS, Kaiserslautern, Germany, 2005.

[47] Amit Jain and R.K. Shyamasundar. "Failure Detection and Membership Management in Grid Environments". In Proc. of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), IEEE, ISBN 1550-5510/04, 2004.

[48] Naohiro Hayashibara, Adel Cherif, and Takuya Katayama. "Failure Detectors for Large-Scale Distributed Systems". IEEE, ISBN 1060-9857/02, 2002.

[49] Péter Urbán, Ilya Shnayderman, and Andre Schiper. "Comparison of Failure Detectors and Group Membership: Performance Study of Two Atomic Broadcast Algorithms (Extended Version)". Technical Report IC/2003/15, Faculté d'Informatique et Communications (I&C) École Polytechnique Fédérale de Lausanne (EPFL), 2003.

[50] Antony Rowstron1, Anne-Marie Kermarrec1, Miguel Castro1, and Peter Druschel. "SCRIBE: The design of a large-scale event notification infrastructure". In Proc. of 3rd International Workshop on Networked Group Communication (NGC2001), UCL, London, UK, November 2001.

[51] Leonardo Mariani. "Fault-tolerant routing for p2p systems with unstructured topology". In Proc. of the 2005 International Symposium on Applications and the Internet (SAINT 2005), IEEE Computer Society, February 2005.

[52] K.P. Gummadi, R. Gummadi, S.D. Gribble, S. Ratnasamy, S. Shenker, and I.Stoica. "The Impact of DHT Routing Geometry on Resilience and Proximity". ACM SIGCOMM, Karlsruhe, Germany, August 25–29, 2003.

[53] Zhiyu Liu, Guihai Chen, Chunfeng Yuan, Sanglu Lu, and Chengzhong Xu. "Fault Resilience of Structured P2P Systems". In Proc. of 5th International Conference on Web Information Systems Engineering, Brisbane, Australia, November 22-24, 2004.

[54] The Peer-to-Peer Trusted Library Project. URL: http://sourceforge.net/projects/ptptl/, June 2006.

[55] The Project JXTA: P2P Framework. URL: http://www.jxta.org/, May 2006.

[56] P. McDaniel, A. Prakash, and P. Honeyman. “Antigone: A flexible framework for secure group communication”. In Proc. of 8th USENIX Security Symposium, pages 99–114, Aug. 1999.

[57] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, James W. O'Toole. “Overcast: Reliable Multicasting with an Overlay Network”. Proc. of 4th Symposium on Operating System Design and Implementation, San Diego, 2000.

[58] JXTA Metering and Monitoring Project. URL: http://meter.jxta.org, May 2006.

[59] Felber, P., Guerraoui, R., D´efago, X., and Oser, P.. “Failure detector as first class objects”. In International Symposium on Distributed Objects and Applications (DOA), pages 132–141, Scotland, 1999.

[60] Stelling, P., DeMatteis, C., Foster, I. T., Kesselman, C., Lee, C. A., and von Laszewski. “A fault detection service for wide area distributed computations”. Cluster Computing, 2(2):117–128, 1999.

[61] Hayashibara, N., D´efago, X., and Katayama, T.. “The φ accrual failure detector”. In Symposiumon Reliable Distributed Systems (SRDS’2004), pages 66–78, Florian´opolis, Brazil, 2004.

[62] Bertier, M., Marin, O., and Sens, P.. “Implementation and performance evaluation of an adaptable failure detector”. In DSN ’02: Proceedings of the 2002 International Conference on Dependable Systems and Networks, pages 354–363. IEEE Computer Society, 2002.

[63] Gemmell, J.. “Scalable reliable multicast using erasure-correcting re-sends”. Technical report, msr-tr-97-20, Microsoft Research Center, 1997.

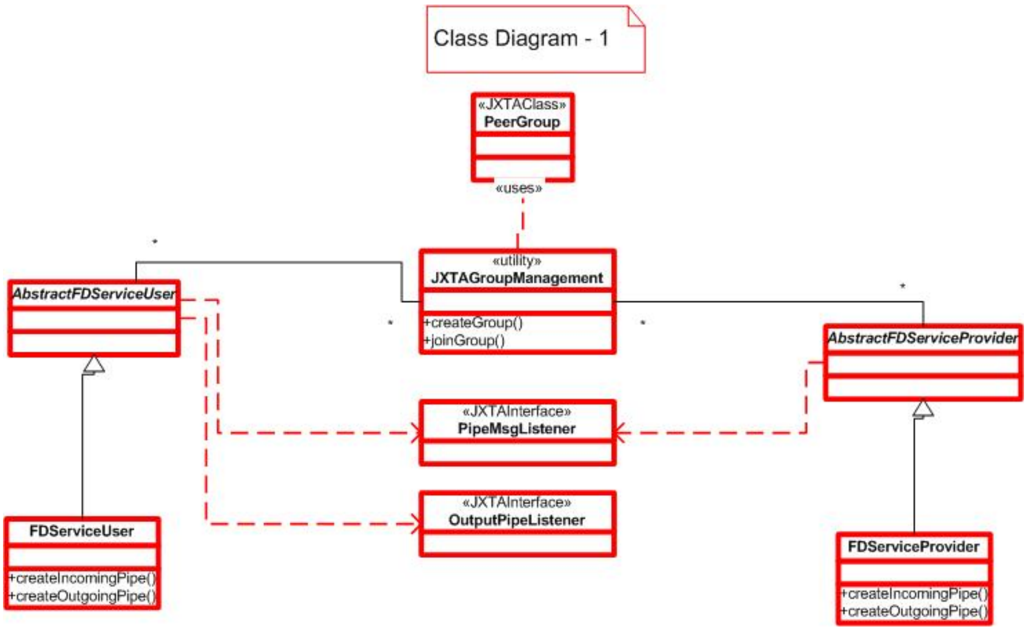
[64] Chu, Y.-H., Rao, S. G., and Zhang, H.. “A case for end system multicast”. In Measurement and Modeling of Computer Systems, pages 1–12, 2000.

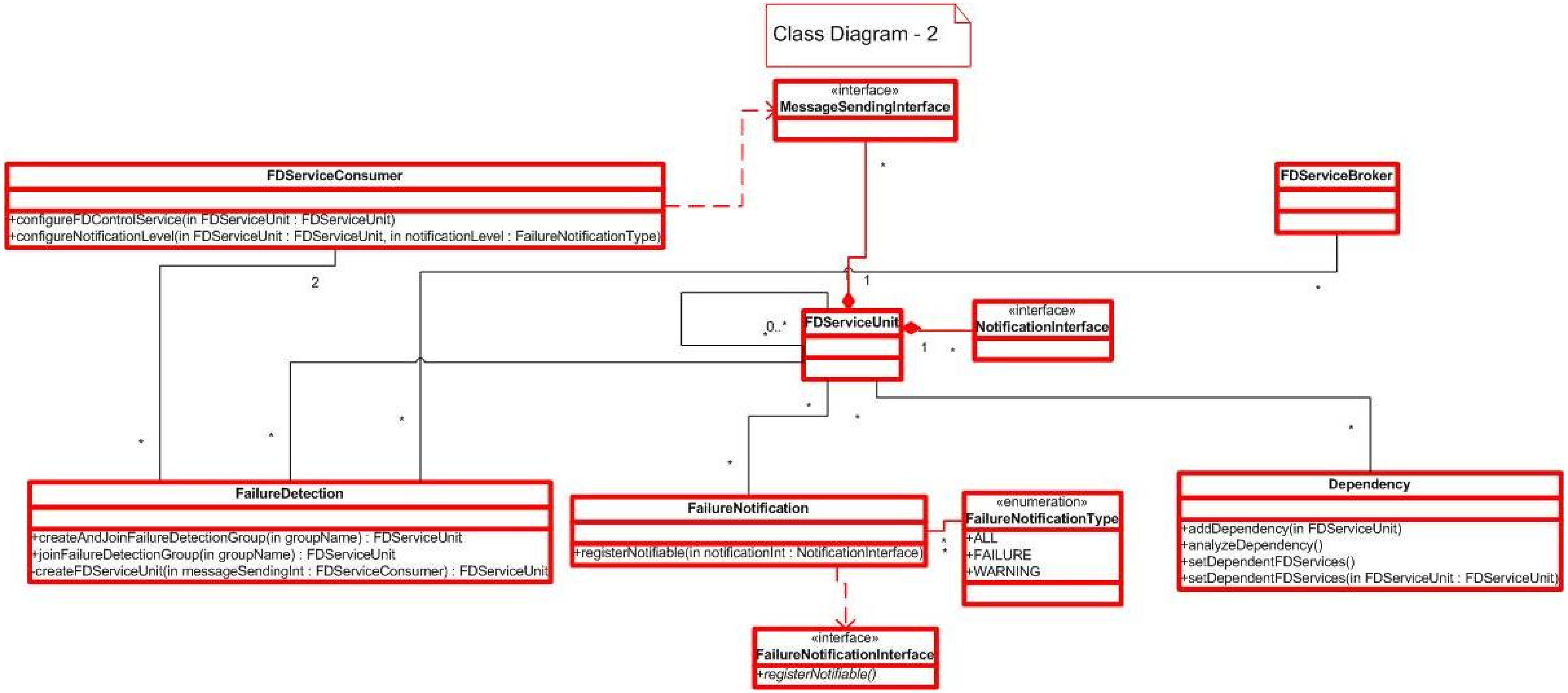
[65] Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y.. “Bimodal multicast”. ACM Transactions on Computer Systems, 17(2):41–88, 1999.

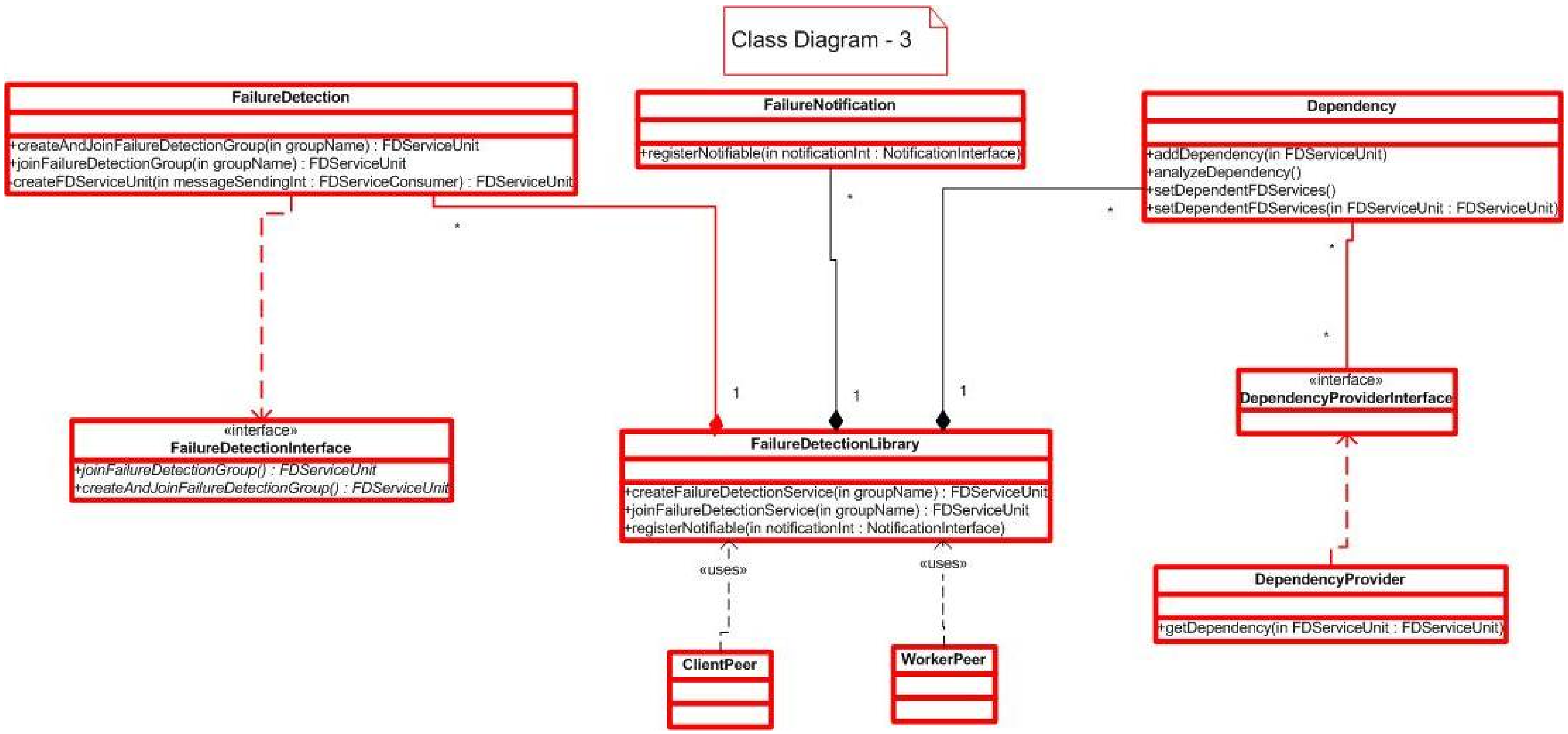
APPENDIX A

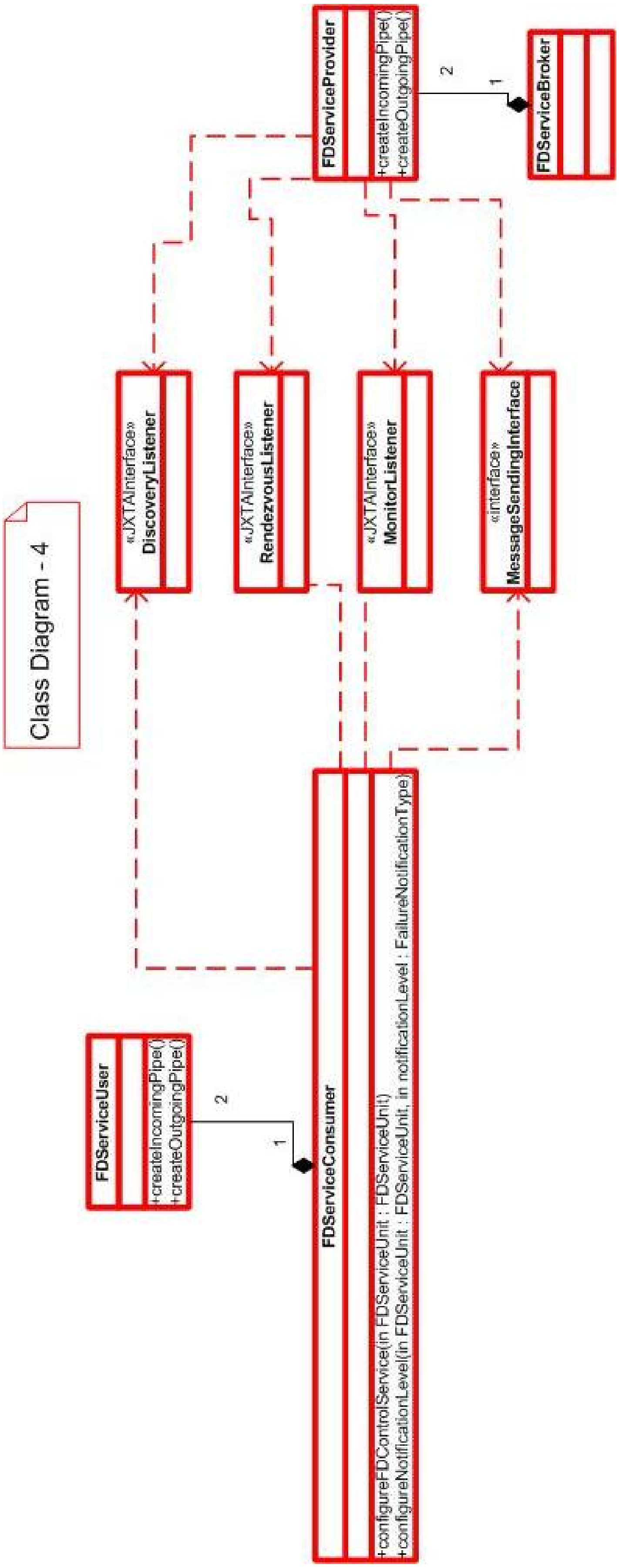
FAILURE DETECTION LIBRARY DESIGN

A.1. Class Diagram



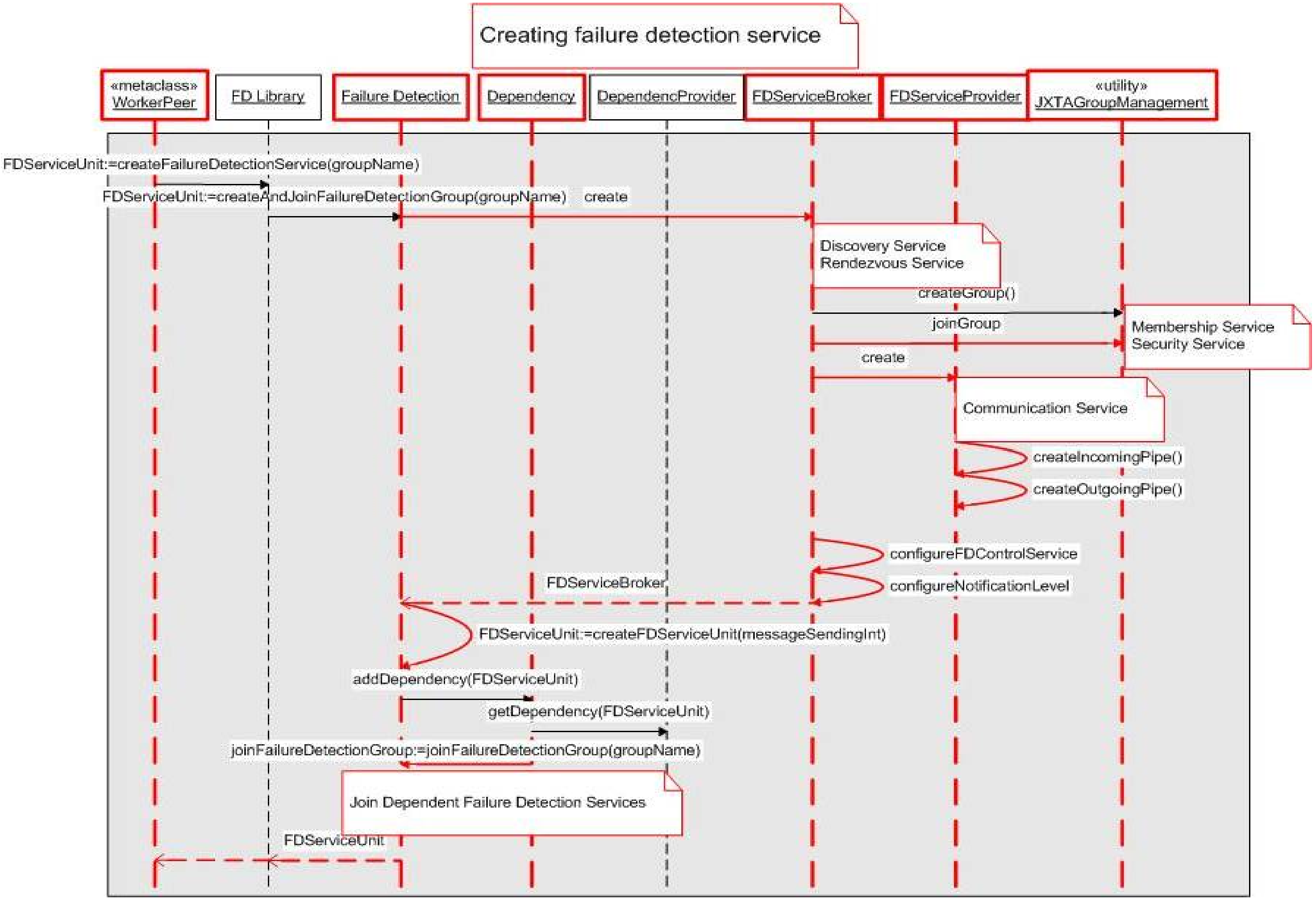

Class Diagram - 2
«interface»
MessageSendingInterface
FDServiceConsumer
+configureFDControlService(in FDServiceUnit : FDServiceUnit)
+configureNotificationLevel(in FDServiceUnit : FDServiceUnit, in notificationLevel : FailureNotificationType)
FDServiceBroker
FDServiceUnit
0..*
1
*
2
«interface»
NotificationInterface
FailureDetection
+createAndJoinFailureDetectionGroup(in groupName) : FDServiceUnit
+joinFailureDetectionGroup(in groupName) : FDServiceUnit
-createFDServiceUnit(in messageSendingInt : FDServiceConsumer) : FDServiceUnit
FailureNotification
+registerNotifiable(in notificationInt : NotificationInterface)
«enumeration»
FailureNotificationType
+ALL
+FAILURE
+WARNING
Dependency
+addDependency(in FDServiceUnit)
+analyzeDependency()
+setDependentFDServices()
+setDependentFDServices(in FDServiceUnit : FDServiceUnit)
«interface»
FailureNotificationInterface
+registerNotifiable()

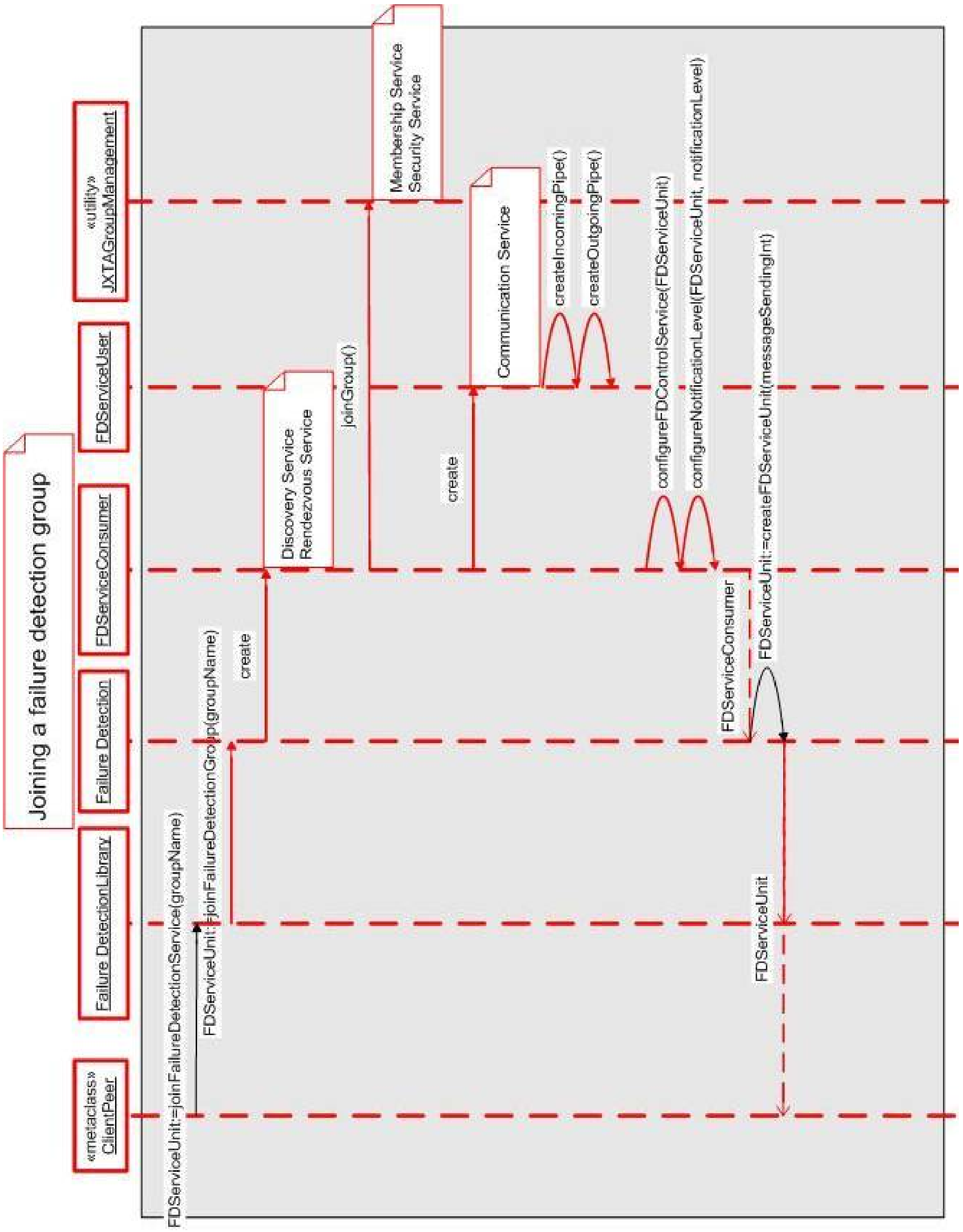

Class Diagram - 3
FailureDetection
+createAndJoinFailureDetectionGroup(in groupName) : FDServiceUnit
+joinFailureDetectionGroup(in groupName) : FDServiceUnit
-createFDServiceUnit(in messageSendingInt : FDServiceConsumer) : FDServiceUnit
FailureNotification
+registerNotifiable(in notificationInt : NotificationInterface)
Dependency
+addDependency(in FDServiceUnit)
+analyzeDependency()
+setDependentFDServices()
+setDependentFDServices(in FDServiceUnit : FDServiceUnit)
*
*
*
*
*
1
1
1
«interface»
FailureDetectionInterface
+joinFailureDetectionGroup() : FDServiceUnit
+createAndJoinFailureDetectionGroup() : FDServiceUnit
FailureDetectionLibrary
+createFailureDetectionService(in groupName) : FDServiceUnit
+joinFailureDetectionService(in groupName) : FDServiceUnit
+registerNotifiable(in notificationInt : NotificationInterface)
«interface»
DependencyProviderInterface
«uses»
«uses»
ClientPeer
WorkerPeer
DependencyProvider
+getDependency(in FDServiceUnit : FDServiceUnit)

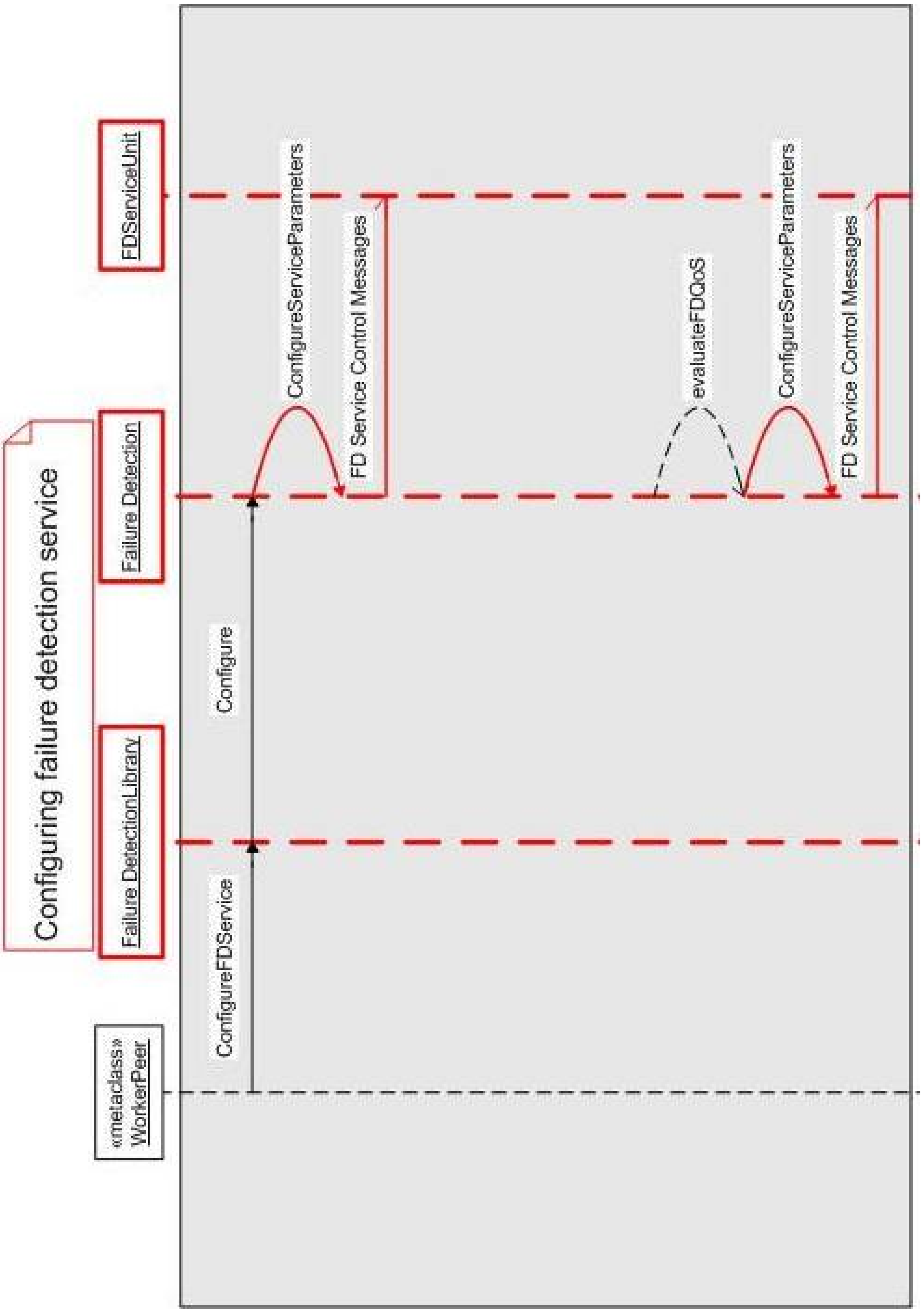

Class Diagram - 4
FDServiceUser
+createIncomingPipe()
+createOutgoingPipe()
2
1
FDServiceConsumer
+configureFDControlService(in FDServiceUnit : FDServiceUnit)
+configureNotificationLevel(in FDServiceUnit : FDServiceUnit, in notificationLevel : FailureNotificationType)
«JXTAInterface»
DiscoveryListener
«JXTAInterface»
RendezvousListener
«JXTAInterface»
MonitorListener
«interface»
MessageSendingInterface
FDServiceProvider
+createIncomingPipe()
+createOutgoingPipe()
2
1
FDServiceBroker

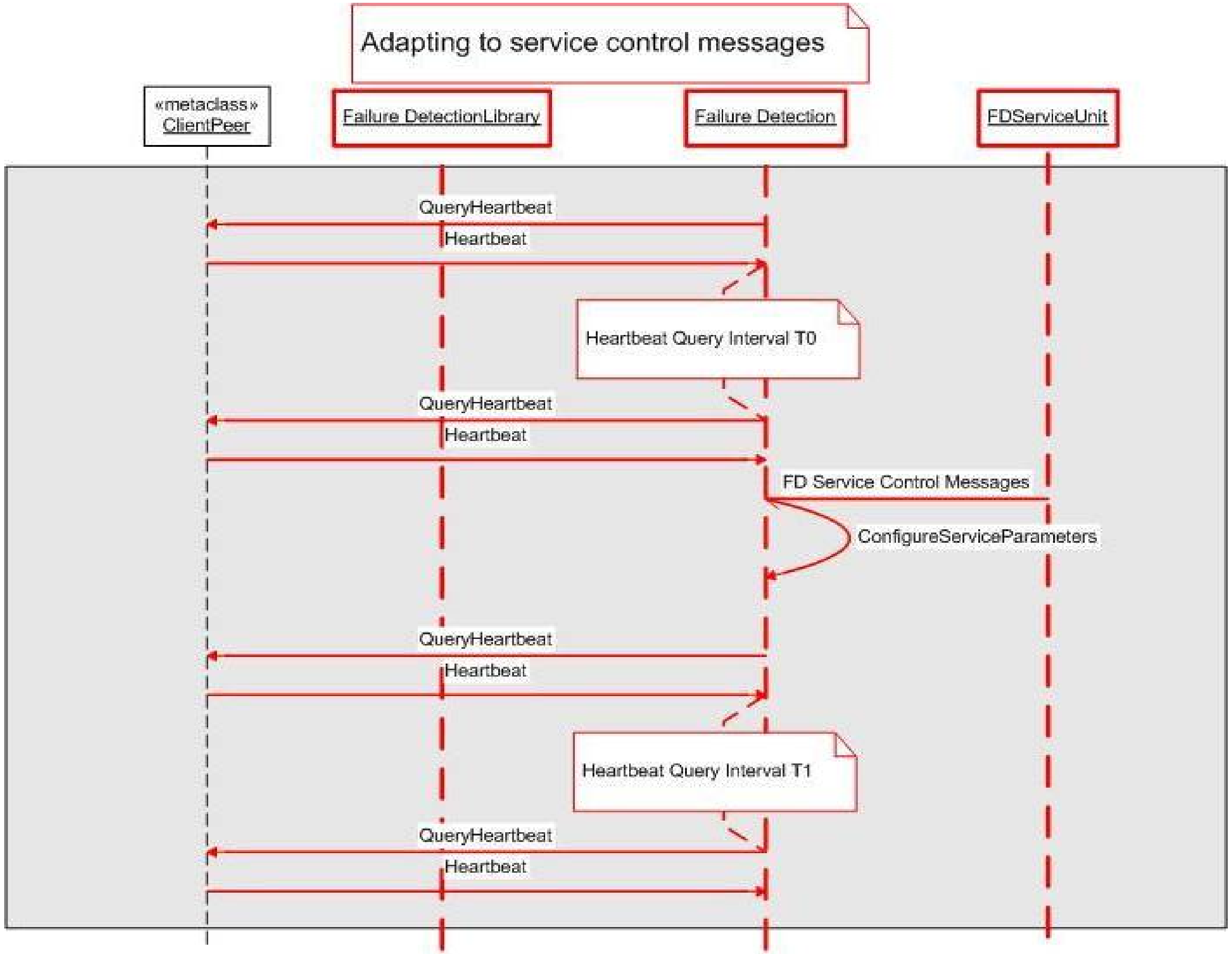
A.2. Message Sequence Diagrams

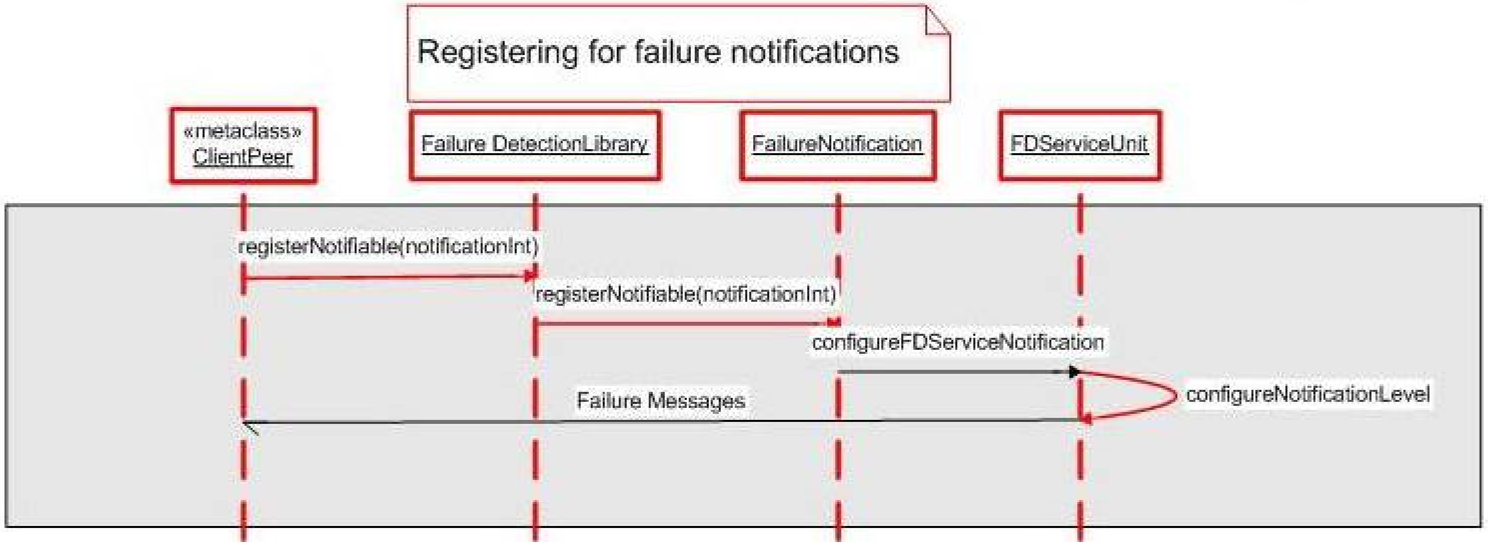
Message sequence diagrams in the design of the library are presented below. These diagrams realize the main scenarios in the library: creating/joining/configuring failure detection services, adapting to service control parameters, and registering for failure notifications.


Creating failure detection service
«metaclass» WorkerPeer
FD Library
Failure Detection
Dependency
DependencProvider
FDServiceBroker
FDServiceProvider
«utility» JXTAGroupManagement
FDServiceUnit:=createFailureDetectionService(groupName)
FDServiceUnit:=createAndJoinFailureDetectionGroup(groupName)
create
Discovery Service
Rendezvous Service
createGroup()
joinGroup
Membership Service
Security Service
create
Communication Service
createIncomingPipe()
createOutgoingPipe()
configureFDControlService
configureNotificationLevel
FDServiceBroker
FDServiceUnit:=createFDServiceUnit(messageSendingInt)
addDependency(FDServiceUnit)
getDependency(FDServiceUnit)
joinFailureDetectionGroup:=joinFailureDetectionGroup(groupName)
Join Dependent Failure Detection Services
FDServiceUnit


Joining a failure detection group
«metaclass»
ClientPeer
Failure DetectionLibrary
Failure Detection
FDServiceConsumer
FDServiceUser
«utility»
JXTAGroupManagement
FDServiceUnit:=joinFailureDetectionService(groupName)
FDServiceUnit:=joinFailureDetectionGroup(groupName)
create
Discovery Service
Rendezvous Service
joinGroup()
Membership Service
Security Service
create
Communication Service
createIncomingPipe()
createOutgoingPipe()
configureFDControlService(FDServiceUnit)
configureNotificationLevel(FDServiceUnit, notificationLevel)
FDServiceUnit:=createFDServiceUnit(messageSendingInt)
FDServiceConsumer
FDServiceUnit
FDServiceUnit


Configuring failure detection service
«metaclass»
WorkerPeer
Failure DetectionLibrary
Failure Detection
FDServiceUnit
ConfigureFDService
Configure
ConfigureServiceParameters
FD Service Control Messages
evaluateFDQoS
ConfigureServiceParameters
FD Service Control Messages


Adapting to service control messages
«metaclass» ClientPeer
Failure DetectionLibrary
Failure Detection
FDServiceUnit
QueryHeartbeat
Heartbeat
Heartbeat Query Interval T0
QueryHeartbeat
Heartbeat
FD Service Control Messages
ConfigureServiceParameters
QueryHeartbeat
Heartbeat
Heartbeat Query Interval T1
QueryHeartbeat
Heartbeat


Registering for failure notifications
«metaclass»
ClientPeer
Failure DetectionLibrary
FailureNotification
FDServiceUnit
registerNotifiable(notificationInt)
registerNotifiable(notificationInt)
configureFDServiceNotification
configureNotificationLevel
Failure Messages

APPENDIX B

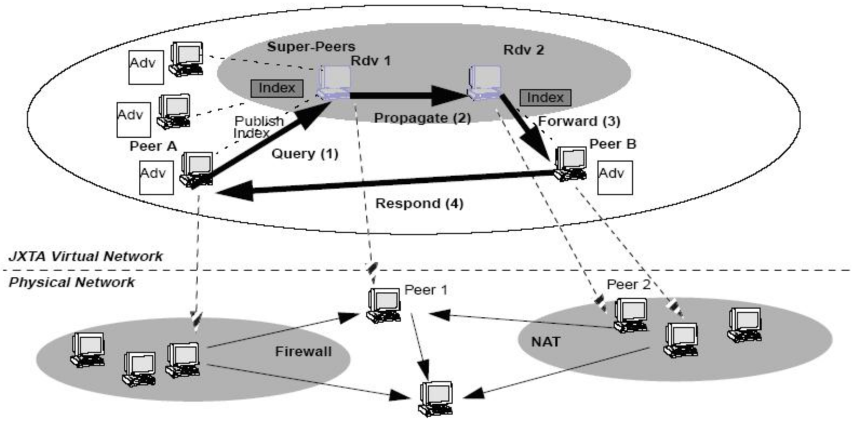
Project JXTA, P2P FRAMEWORK

Information (both textual and visual) in this section is mainly compiled from [43].

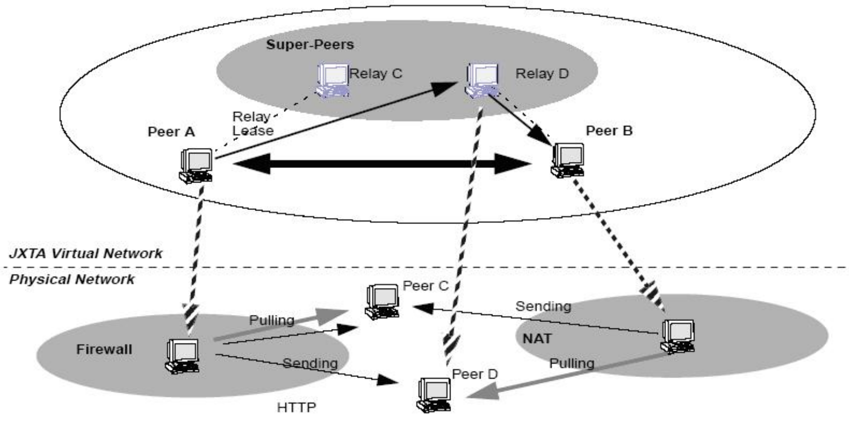
B.1. Peer Architecture

In addition to simple peers in P2P paradigm, rendezvous and relay peer concepts are introduced in Project JXTA.

- *Rendezvous Super-Peers*: Project JXTA proposes rendezvous peers which use a resolver infrastructure to find advertisements of resources. Rendezvous conceptually corresponds to well-known advertisement locations to search for a resource and resolution infrastructure provides discovery of these resource advertisements. Rendezvous super-peers can also organize themselves into a rendezvous network to provide efficient bootstrapping and high availability of advertisements.



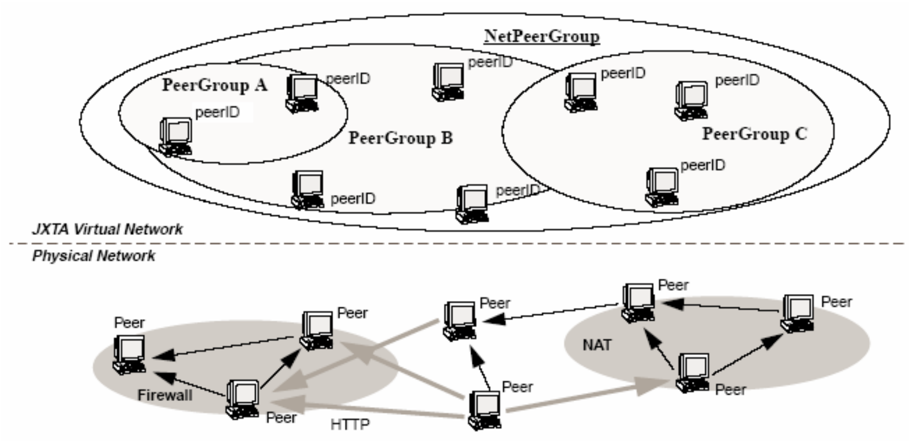
- *Relay Super-Peers*: The Project JXTA proposes relay super-peers to play a bridge role between peers that do not have direct physical connectivity (NAT, firewalls). Playing the bridge role includes spooling messages for unreachable edge peers. Usage of relay peers in message routes is handled by JXTA protocols and transparent to applications.



In the figure above, Peer A wants to send a message to Peer B. Since Peer B is behind a NAT (Peer B address is not reachable from Peer A), Peer A cannot send a message directly to Peer B. Peer B uses the relay Peer D to make itself reachable.

B.2. Peer Group Architecture

Peers in the Project JXTA network self-organize into *peer groups* where peer groups represent dynamic sets of peers that have a common set of interests, and have agreed upon a common set of policies (membership, content exchange, etc.). The creation, publishing and discovering of these peer groups are specified in Project JXTA.



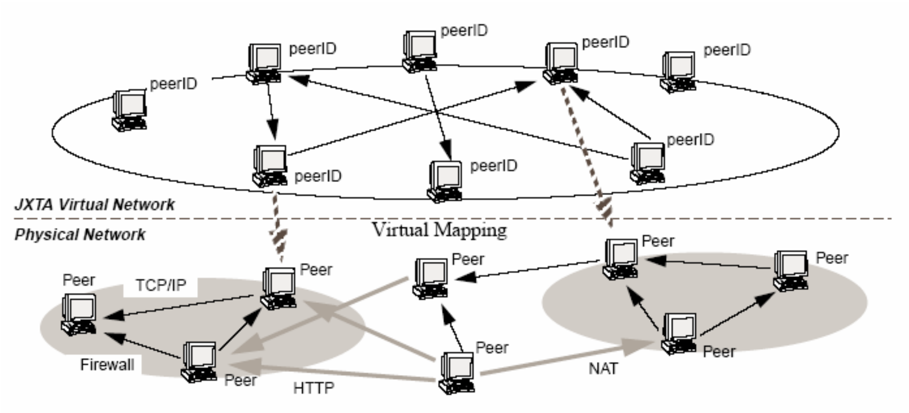
In the above figure, PeerGroup A shows a peer group that is a subset of a physical firewall domain.

PeerGroup B shows a peer group that is spanning multiple physical domains. Peergroup C shows a peer group that is exactly mapping the boundary of a NAT domain. A peer can belong to multiple peer groups at the same time.

A peer becomes a member of NetPeerGroup at boot time which acts as the root peer group in Project JXTA. Peer groups typically publish a set of services called peergroup services with associated protocols (discovery, resolver, pipe, peer info, and rendezvous). Advertisements of peer groups include the list of all peer group services. Creators of these peer groups customize their peer group services to according to their needs. Peer group services are composed of a collection of instances of this service which run on each member of the peer group. These instances can work autonomously as replicas or by cooperate with each other to provide the service. This type of abstraction provides highly available services in the scope of a peer group. If one peer fails in the peer group, the overall service is not affected as far as it is available in another peer member.

B.3. JXTA Virtual Overlay Network

The Project JXTA protocols establish a *virtual network* overlay that allow peers to directly interact and self-organize independently regardless of their network connectivity and topology (behind firewalls, NATs, non-IP networks). Project JXTA provides a configurable overlay policy. As a default, it provides a loosely-coupled unstructured policy but peer group creators have the ability to overwrite the default policy and define their own policies. Loosely coupled unstructured form is designed for highly fluctuating and unpredictable environments. A detailed description of overlaying policy of Project JXTA can be found in [43].

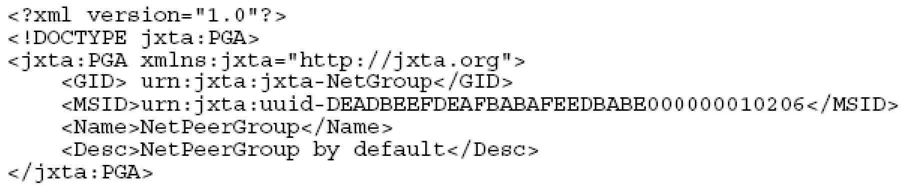


B.4. Identification of Resources

To identify resources and services in Project JXTA, JXTA IDs and advertisement mechanisms are used.

- *JXTA IDs*: The Project JXTA addresses its network resources (peer, pipe, data, peer group, etc.) in a uniform and location independent way. Every network resource in a Project JXTA network is assigned a unique JXTA ID. Any resource in the Project JXTA network is accessed by using its JXTA ID.

- *Resource Advertisements*: All network resources in the Project JXTA network, such as peers, peer groups, pipes, and services are represented by advertisements. Project JXTA uses advertisements which are represented as language-neutral metadata structures in XML documents to describe resources. Project JXTA standardizes advertisements for the following core JXTA resources: peer, peer group, pipe, service, metering, route, content, rendezvous, peer endpoint, transport. Below is a sample Project JXTA peer group advertisement.



```
<?xml version="1.0"?>
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
    <GID> urn:jxta:jxta-NetGroup</GID>
    <MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206</MSID>
    <Name>NetPeerGroup</Name>
    <Desc>NetPeerGroup by default</Desc>
</jxta:PGA>
```

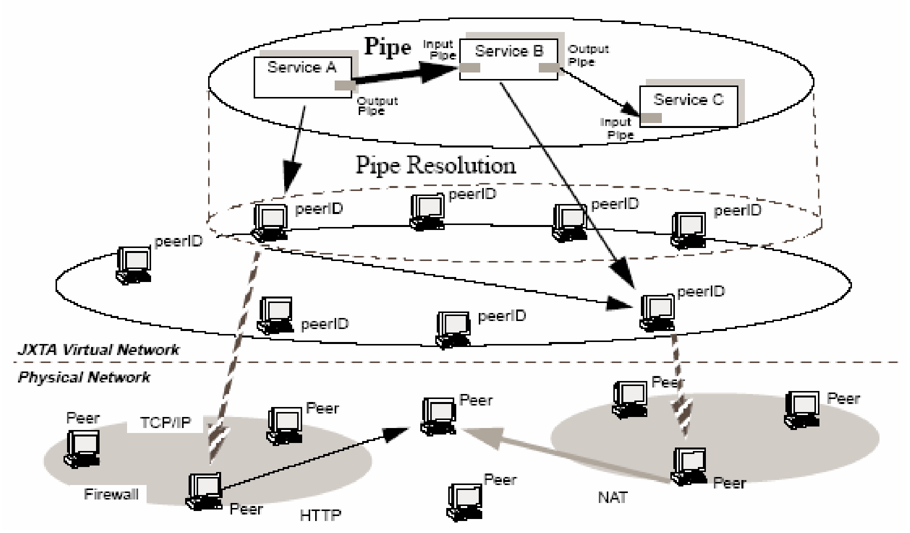
B.5. Communication

Project JXTA uses a binary wire format for communication between peers. This format supports both XML and binary payloads to be sent. The messages are sent through virtual communication channels called *Pipes* in Project JXTA. It is a virtual communication abstraction between peers where peers can send and receive messages through. They can connect one or more peers. The pipe ends are referred as input pipe (receiving messages through) and output pipe (sending messages through). Pipes are categorized into two modes of communication in Project JXTA:

- *Point-to-point pipe*: It provides a unicast style communication. It connects exactly two pipe ends with a unidirectional and asynchronous channel. In these types of pipes, no reply or acknowledgment operation is supported.

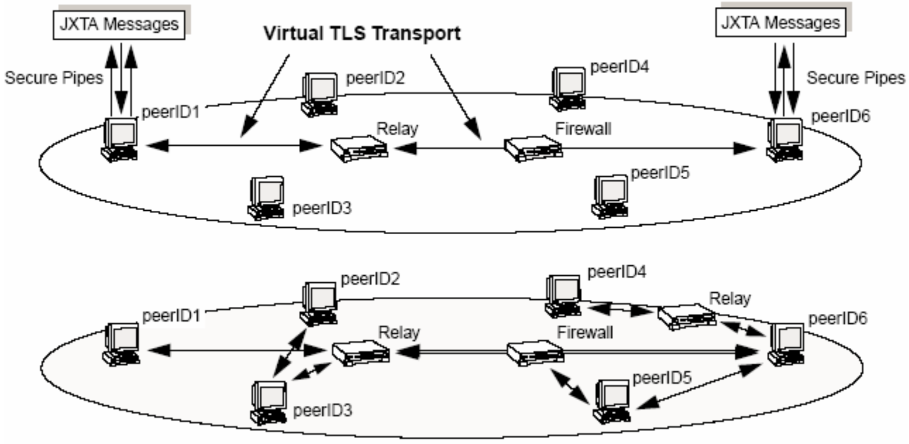
- *Propagate pipe*: It provides a broadcast style communication. It connects one output pipe to multiple input pipes. The propagate message is sent to all input pipe ends in the current peer group context. In implementations of this type of communication.

    - IP multicast can be used if propagation scope maps to an underlying physical IP based subnet.
    - Point-to-Point communication cans be used on transport where multicast is not provided (e.g. HTTP).



B.6. Security

Project JXTA allows peers to own their certificate authorities and provides strong cipher algorithms where default cipher suite is RSA1024 with 3DES and SHA-1. It provides secure communication (also called as secure pipes) between peers by using Transport Layer Security (TLS). The model instantiates a virtual TLS transport after resolving endpoints of secure pipes where the instantiated transport bi-directionally secures communication with TLS support, and is independent of underlying network topology. (It may go through JXTA relays and pass between different network topologies). To support bi-directional security, virtual transport requires each communication peer to possess X509.V3 root certificates.



In the scope of peer groups, Project JXTA security model can be used to implement an authentication mechanism based on X509.V3 certificates. When a peer wants to join a peer group, it receives the root certificate of peer group creator through a secure TLS connection. To be a member, it then acquires a group membership X509.V3 certificate which is signed with the private key of group creator's root certificate. These certificates are locally stored on peer and guarded with a password phrase.

APPENDIX C

JXTA CALLS FOR FAILURE DETECTION API FUNCTIONS

C.1. Worker Peer API

Create_FD_Group:

Description: Creation of a group is done by constructing the group and publishing it to the discovery service through which client peers can search for:

Synopsis:

```
//Construction of a group

ModuleImplAdvertisement implAdv = parent.getAllPurposePeerGroupImplAdvertisement();

parent.newGroup(null, implAdv, groupName, "FD group adv descr");

//Publishing the group

parent.getDiscoveryService().remotePublish(adv);
```

Create_FD_Services:

Description: In JXTA, creating services for a peer group is done by first creating/publishing of module class/spec advertisements of the service to the discovery service of the target peer group and then creating communication channels (pipes) for the service using the pipe service of the target peer group.

Synopsis:

Creating/Publishing module class advertisements of the service:

```
// Publishing Module Class  advertisements
```

```
        ModuleClassAdvertisement mcadv = (ModuleClassAdvertisement)
                                    AdvertisementFactory.newAdvertisement(
                                      ModuleClassAdvertisement.getAdvertisementType());
        mcadv.setName("JXTAMOD:" + advName);
        mcadv.setDescription("FD Service X");
        mcID = IDFactory.newModuleClassID();
        mcadv.setModuleClassID(mcID);
        discovery.publish(mcadv);
        discovery.remotePublish(mcadv);
        /*
        …
        */
        // Publishing Module Spec advertisements
        ModuleSpecAdvertisement mdadv =
(ModuleSpecAdvertisement)AdvertisementFactory.newAdvertisement(
ModuleSpecAdvertisement.getAdvertisementType());
        mdadv.setName("JXTASPEC:" + advName);
        mdadv.setVersion("Version 1.0");
        mdadv.setCreator("sun.com");
        mdadv.setModuleSpecID(IDFactory.newModuleSpecID(mcID));
        mdadv.setSpecURI("http://www.jxta.org/Ex1");
        pipeAdvertisement = createPipeAdvertisement(advName,pipeType);
        mdadv.setPipeAdvertisement(pipeAdvertisement);
        discovery.publish(mdadv);
        discovery.remotePublish(mdadv);
```

Creating communication channels (pipes) by publishing pipe advertisements to discovery service of the target peer group.

```
        //Publishing pipe advertisements

        ModuleClassID mcID = publishModuleClassAdvertisement(discovery,serviceName);

        try {
    pipes.createInputPipe(publishModuleSpecAdvertisement(discovery,serviceName,mcID,PipeService.P
ropagateType),this);

        } catch (IOException e) {

        e.printStackTrace();

        }
```

Implementing unreliable failure detector semantics in failure detection services:

```
        //Receive failure messages from incoming pipe as pipe message notifications

                Message.ElementIterator en =
((PipeMsgEvent)arg0).getMessage().getMessageElements();

                MessageElement msgElement =
((PipeMsgEvent)arg0).getMessage().getMessageElement(null, "FailureMessageTag");

                notificationInt.notify(msgElement.toString());

        /*Keep a suspect list according to received failure messages from members. If any consensus is
reached on failure of a process, broadcast its failure to all members. If no consensus is reached, broadcast
suspect list to all active members.*/

        //Broadcast failure messages through outgoing pipe

        Message msg = new Message();

        StringMessageElement sme = new StringMessageElement("FailureMessageTag ", data , null);

        msg.addMessageElement(null, sme);

        ((OutputPipe)outPipe).send (msg);
```

Implementing membership semantics in failure detection services:

```
    //Receive membership messages from incoming pipe as pipe message notifications

        Message.ElementIterator en =
((PipeMsgEvent)arg0).getMessage().getMessageElements();

        MessageElement msgElement =
((PipeMsgEvent)arg0).getMessage().getMessageElement(null, "FailureMessageTag");

        notificationInt.notify(msgElement.toString());

    /*

    Analyze membership view according to received membership messages. If any change occurs in
membership view, broadcast membership views to members.

    */

    //Broadcast failure messages through outgoing pipe

    Message msg = new Message();

    StringMessageElement sme = new StringMessageElement("FailureMessageTag ", data , null);

    msg.addMessageElement(null, sme);

    ((OutputPipe)outPipe).send (msg);
```

C.2. Client Peer API

Discover_FD_Group:

Description: In JXTA, discovery of a group is done by first sending group discovery messages using discovery service and afterwards locating required group by its advertisement.

Synopsis:

Sending group discovery message using discovery service:

```
    // group discovery message

    discovery.getRemoteAdvertisements(null, DiscoveryService.GROUP,null, null, 5);
```

Locating required group by its advertisement:

```
//Locating the group

Enumeration en = res.getAdvertisements();

while (en.hasMoreElements()) {

   adv = (PeerGroupAdvertisement) en.nextElement();

    if(adv.getName().equals(requiredServiceGroupName)){

       PeerGroup newPeerGroup = parent.newGroup(adv);

 }

}
```

A sample failure detection group advertisement:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jxta:PGA><jxta:PGA xmlns:jxta="http://jxta.org">

<GID> urn:jxta:uuid-19CED815D57D4337891B9AD1D182816D02</GID>

<MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010306</MSID>

<Name>Service A</Name>

<Desc>Failure Detection Group for System Service A</Desc>

</jxta:PGA>
```

Join_FD_Group:

Description: In JXTA, joining to a peer group is done by authenticating to membership services of the group. It is done by following calls to JXTA constructs.

Synopsis:

Joining a failure detection group

```
        //Joining a FD group

        MembershipService membership = group.getMembershipService();

        AuthenticationCredential authenticationCred = new AuthenticationCredential( group, null,
credentials );

        Authenticator authenticator = membership.apply(authenticationCred);

          if (authenticator.isReadyForJoin())

            Credential myCred = membership.join(authenticator);

        else

            ; //Failure
```

Use_FD_Service:

Description: In JXTA, using a service is done by first sending service discovery messages using discovery service, locating service by its advertisement and connecting to incoming and/or outgoing pipes of the service.

Synopsis:

Sending service discovery messages:

```
        //Sending service discovery messages

        discovery.getRemoteAdvertisements(null, DiscoveryService.ADV, "Name",
serviceSpecificationAdvertisement,1, null);
```

Locating and connecting to Service:

```
        //Extracting pipe advertisements from service advertisements

        ModuleSpecAdvertisement mdsadv = (ModuleSpecAdvertisement) en.nextElement();

        PipeAdvertisement pipeAdvertisement = mdsadv.getPipeAdvertisement();
```

```
//Using pipe advertisements to use service pipes

pg.getPipeService().createInputPipe(pipeAdvertisement, this);

pg.getPipeService().createOutputPipe (pipeAdvertisement, this);
```

A sample outgoing pipe advertisement for a failure detection service:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jxta:PipeAdvertisement>

<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">

<Id>urn:jxta:uuid-1C5D9D10A3534F23A480FAB85D61EEF07AC5B4603161405887F1D0DE7078F14604</Id>

<Type>JxtaPropagate</Type>

<Name>OUTGOING_PIPE</Name>

<Desc>Outgoing Pipe Advertisement for Failure Detection Service A</Desc>

</jxta:PipeAdvertisement>
```