Fault Tolerance in Mobile Agents: State-of-the-Art and Challenges

Bassey Isong and Eyaye Bekele
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Bassey Isong
E-mail: isongb77@gmail.com

Eyaye Bekele
E-mail: eyayeb@gmail.com

University advisor(s):
Kennet Henningsson
Software Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25
ABSTRACT

The flexibility offered by mobile agents is quite noticeable in distributed computing environments. But the flexibility comes with a set of new levels of complexity due to their autonomous nature. The mobile agent paradigm introduce additional threats since agents systems are prone to failures originating from bad communication, security attacks, agent server crashes, system resources unavailability, network congestion or even deadlock situations. In such events, mobile agents either get lost or damaged (partially or totally) during execution. In order to gain solid foundation at the heart of today’s e-society, the mobile agent technology paradigm must address the issue of reliability. Making mobile agents fault tolerant is a measure taken to increase the dependability and reliability of agent-based application. Mobile agent’s fault tolerance is gaining momentum and many approaches have been proposed. Despite these efforts, the field still suffers from set backs in the form of persistent challenges. This study analyzes the existing fault tolerance approaches against a proposed generic fault tolerance framework that consists of a monitoring, planning and recovery process execution phases. Following the analysis, this study brings about the state-of-the-art in mobile agent’s fault tolerance approaches and the lingering challenges that affect mobile agent’s fault tolerance implementations from being efficiently and fully realized. The study will serve as a guide to future researches and possible solution for a more reliable and transparent fault tolerance in mobile agents.

Keywords: Mobile Agents, Fault tolerance, Reliability, Failure, Challenges, Agent platforms, State-of-the-art.
ACKNOWLEDGMENT

We would like to extend our sincere gratitude to our advisor Kennet Henningsson for his guide and constructive comments throughout our thesis writing. We would also like to thank Sofia Swartz, Librarian in Info-centre Library, who assisted us in identifying search resources and developing our search terms.

Our special thanks to our families and friends who have been supporting and encouraging us throughout our journey. We thank God for making everything happen.
# Table of Contents

ABSTRACT ................................................................................................................................. 1

ACKNOWLEDGMENT .................................................................................................................. II

CHAPTER 1 ..................................................................................................................................... 1

INTRODUCTION .............................................................................................................................. 1

1.1 INTRODUCTION ...................................................................................................................... 1
1.2 BACKGROUND ....................................................................................................................... 1
1.3 RELATED WORK ..................................................................................................................... 2
1.4 MOBILE AGENTS .................................................................................................................. 3
  1.4.1 Characteristics of Mobile Agents .................................................................................. 4
  1.4.2 Benefits of Mobile Agents .......................................................................................... 5
  1.4.3 Application Areas of Mobile Agents ......................................................................... 6
1.5 PROBLEM STATEMENTS ........................................................................................................ 7
  1.5.1 Aims and Objectives .................................................................................................... 7
  1.5.2 Research Questions ..................................................................................................... 7
  1.5.3 Research Methodology ............................................................................................. 9
  1.5.4 Scope of the Work ...................................................................................................... 10
  1.5.5 Terminology ............................................................................................................... 10

CHAPTER 2 ..................................................................................................................................... 12

SYSTEMATIC LITERATURE REVIEW .......................................................................................... 12

2.1 OVERVIEW ............................................................................................................................ 12
2.2 PLANNING THE REVIEW ....................................................................................................... 12
  2.2.1 Review Objectives ...................................................................................................... 12
  2.2.2 Review protocol ........................................................................................................... 13
2.3 CONDUCTING THE REVIEW ............................................................................................... 23
  2.3.1 Identification of Research .......................................................................................... 23
  2.3.2 Selection of Primary Studies ..................................................................................... 24
  2.3.3 Selection Execution ..................................................................................................... 24
  2.3.4 Study Quality Assessment ......................................................................................... 26
  2.3.5 Data Extraction and Execution ................................................................................ 27
  2.3.6 Data Synthesis ........................................................................................................... 28
2.4 REPORTING THE REVIEW .................................................................................................... 28

CHAPTER 3 ..................................................................................................................................... 30

SYSTEMATIC REVIEW RESULTS ............................................................................................... 30

3.1 BASIC CONCEPTS .................................................................................................................. 30
  3.1.1 Agent-centric Approach ........................................................................................... 31
  3.1.2 System-centric Approach ......................................................................................... 31
3.2 FAULT MODELS .................................................................................................................... 32
  3.2.1 Communication Failure ............................................................................................ 33
  3.2.2 Crash Failure .............................................................................................................. 33
  3.2.3 Agent/Agent Software Failure .................................................................................. 34
3.3 FAULT TOLERANCE SCHEMES .......................................................................................... 34
  3.3.1 Check-pointing Scheme ............................................................................................ 35
  3.3.2 Replication Scheme ................................................................................................. 36
3.4 FAULT TOLERANCE PROTOCOLS ...................................................................................... 38
# FAULT TOLERANCE FRAMEWORK

## 4.1 Performance Factors
- 4.1.1 Agent Size
- 4.1.2 Number of Replica
- 4.1.3 Message Size
- 4.1.4 Number of Messages
- 4.1.5 Number of Hops/Hosts
- 4.1.6 Frequency of Checkpointing

## 4.2 Platform Support
- 4.2.1 Fault Tolerance Components
- 4.2.2 Available Agent Platforms

## 4.3 Challenges
- 4.3.1 Reliable Fault Detection
- 4.3.2 Network Partitioning
- 4.3.3 Lack of Platform Support
- 4.3.4 Full Transaction Support
- 4.3.5 Scalability
- 4.3.6 Lack of Transparency

## 4.4 Addressing the Research Questions
- 4.4.1 Systematic Review Question 4
- 4.4.2 Systematic Review Question 5
- 4.4.3 Systematic Review Question 6

## 4.5 Summary

## Chapter 5

## 5.1 Fault Tolerance Execution Properties

## 5.2 Fault Tolerance Framework Execution Cycle
- 5.2.1 Monitor/Detection Phase
- 5.2.2 Planning Phase
- 5.2.3 Execution Phase

## 5.3 Proposed Fault Tolerance Framework
- 5.3.1 Monitor/Detection Component
- 5.3.2 Planning Component
- 5.3.3 Execution Component
- 5.3.4 Backup Manager Component

## 5.4 Framework Validation

## 5.5 Validity Threats
- 5.5.1 Publication Bias
- 5.5.2 Identification of Primary Studies
- 5.5.3 Data Extraction Consistency
- 5.5.4 Framework Validation
- 5.5.5 Generalization of the Framework

## 5.6 Summary

## Conclusion and Recommendation
1.1 Introduction

Mobile agents are of paramount interest in recent distributed computing trends in both academia and industrial fields. They have distinct characteristics that make them flexible in deployment thereby making the design, implementation, and maintenance of distributed systems a very easy task. This technological paradigm aims at shifting computation towards the data rather than data to the computation [1]. Their overall flexibility makes them more desirable for building distributed applications than other technological paradigms such as client-server, peer-to-peer, etc [2].

Research activities on mobile agent technologies and its application in recent years have been gaining momentum but despite the proliferation, issues of reliability still remain at low level. Their reliable and consistent execution is not isolated from failure especially in the environments they operates. The exponential growth of distributed heterogeneous environments such as the Internet inherently exposes execution of Mobile agents to this adverse condition [3], [4]. Therefore, in order that mobile agent’s technology can gain solid grounds at the heart of our today’s industrial applications, they have to be made reliable enough through fault tolerance to withstand adverse environmental situations. Fault tolerance aims to provide reliable execution of agents even in the face of system failure, although there are unaddressed challenges in achieving absolute tolerance.

Currently, many mobile agents’ fault tolerance approaches in variety of mobile agent platforms exist. These approaches employed different mechanisms to increase the reliability of mobile agents’ execution especially in the failure detection and recovery aspect. Most of the recent approaches are optimizations based on existing mechanisms, some are hybrid-based, while some are exception handling based. For instance, recent fault tolerance approaches mostly rely basically on replication but each approach introduces an optimization to the replication process so as to either gain performance or lower cost of replication. Bringing together state-of-the-art in fault tolerance approaches in mobile agent will serve as a starting point for researchers who want to conduct their studies around similar topics. Furthermore, the study will outline remaining challenges in fault tolerance approaches in mobile agent system so as to give a clear direction to future researchers who will work in this area of study.

1.2 Background

In recent years, the field of Mobile agents has engrossed considerable attention stimulated by the exponential growth of the Internet. Mobile agents are encapsulated pieces of software containing code and data that are able to migrate from one host to another and perform certain task autonomously [3]. They operate in a distributed computing environment consisting of heterogeneous devices and platforms. Their ability to migrate in heterogeneous environment is achieved by making them execute in virtual machine platform and scripted
language environment [3]. The technology has being greatly demonstrated in many applications domains such as in Network Management, Telecommunications, e-Commerce, Information Retrieval, mobile/pervasive computing, artificial intelligence, Workflow Management and Internet Computing, etc [5].

The proliferation of system dependability came with a cost of exceptional challenges in terms of faults in both hardware and software. Naturally, faults in software are logical in nature which is hard to visualize, classify, detect, and correct [2]. Mobile agents like any other software systems are not secluded from operating in abnormal situations. They have a certain level of exposure to fault since they are composed of autonomous entities in distributed dynamic environments [4]. Mobile agents may encounter traditional errors that specifically emerge during migration request failure, communication exceptions or security violation [6]. To operate despite these failures, reliability issues are of paramount importance in the effort to incorporate agent-driven systems into today’s e-society.

Fault tolerance is a mechanism which guarantees system’s reliability by making the system able to withstand faults and resume service in the face of failure. The goal is to allow systems to tolerate faults that remain in the system after its development [2]. In order that fault tolerance in mobile agents accomplished its developmental goals, a reliable execution of mobile agents must adhere to two execution properties; non-blocking and exactly-once execution [7]. This means a single failure must not block the execution of agents and a single execution process must be done once and only once to avoid multiple execution of the same process. However, the different strategies in place aimed commonly to provide mobile agents with fault tolerance and preserve the execution properties.

Several fault tolerance schemes for mobile agents, through algorithm and architecture targeting diverse mobile application exist. Some of the schemes tends to be curative in nature (e.g. exception handling), while others seems to be more preventive [8], [9], [10]. The preventive schemes are generally categorize into either check-pointing or replication-based schemes but in some cases a hybrid of both schemes exist [8]. Mobile agent’s fault tolerance implementation is gaining momentum, and in spite of the numerous schemes for providing reliability, fault tolerance in mobile agent still faced with huge unaddressed challenges that have impeded its full realization.

1.3 Related Work

Based on the zeal to build reliable fault tolerance techniques capable of withstanding unfavourable and unpredictable behaviour of systems, several failure detection and recovery schemes have been designed [9], [6]. Moreover, today most of the schemes exist as an optimization of the main schemes with each having its own strengths and weaknesses in terms of cost-effectiveness and the overheads they present. For instance, literature has shown that the performance overhead of check-pointing is lower than the cost of replication. However, replication offers more reliability and shorter recovery time [8], [9]. As a result, the bottom line in following a particular strategy falls under making trade-off between recovery speed and the cost of the fault tolerance feature.

Bringing the state-of-the-art in fault tolerance in mobile agent to our knowledge, a study was conducted by Qu et al. [9] surveyed the state-of-the-art of agent-based fault tolerance techniques, giving account of the technology proposed so far, their advantages and drawbacks in addition to discussing single point of failures in the approaches. The paper also went further to perform comparison of some of the protocols and describes their limitations and highlighting areas of improvements. However, the description of the fault models, vital fault tolerance protocols in use as well as their assumptions, merits and pitfalls and practicality were not discussed by Qu et al. [9]. Also, in similar study, S. Pleisch
distinguished between two kinds of failures: site and communication failures [11]. The study attempted to categorize the existing protocols and concluded that a simple classification of today's protocols is not always clear-cut for the fact that they are often used as hybrid of the two fault-tolerance techniques [11]. J. Briot et al. [10] based their study on fault tolerance in multi-agent system with emphasis on replication mechanisms as well as its extension to exception handling techniques. Other similar studies were carried, which together will aid us to present the state-of-the-art in fault tolerance in mobile agents but most studies did not address the challenges in achieving state-of-the-art.

Considering the challenges faced by mobile agent’s fault tolerance mechanism, Serugendo et al. [6] reiterated that the challenges are exacerbated by certain agents features combined with specific errors they are faced with, which has impeded fault tolerance schemes from being fully efficient or directly applicable for the realization of mobile applications. Most of the challenges are seen as shortcomings of the existing fault tolerance protocols according to Qu et al. [9] while others are due to lack of involvement of application programmers of mobile application in fault tolerance design [6]. As a result of these factors, these challenges ought to be taken into consideration in order to build a flexible and reliable fault tolerance schemes.

Although mobile agents’ technologies demonstrate flexibility in some aspects, there is a cost of increased complexity in developing fault tolerant applications in mobile agent context [12]. Knowing the state-of-the-art and the challenges is of the essence and result-oriented. The intent of this study is to explore the existing fault tolerance approaches in the world of mobile agents and analyze their employed mechanisms in order to establish the state-of-the-art. Going by the state-of-the-art in fault tolerance mobile agents will guide us in analysing the current unaddressed challenges surrounding basic fault tolerance implementations for mobile agents which in turn will provide guide for future researches on the looming challenges and possible solution for a more reliable fault tolerance in mobile agent system.

1.4 Mobile Agents

Mobile agents are executing program consisting of code, data and execution state that is embedded with certain intelligence and the ability to autonomously migrate across the network representing users in various tasks and satisfying its owner’s request [2], [9]. In other words, mobile agent acts like personal assistant to user(s). See figure 1.1. While migrating and executing on several heterogeneous machines or hosts connected to the network, a mobile agent must be able to communicate with other agents and host systems as well. As a result, the platforms usually provide a reliable protocol for communication. Mobile agents being created in one execution environment, its state and code can be transported along with it to another execution environment in the network where execution resumes. They also have the ability to sense their environment and carry out a set of activities to attain their mission and achieve its goal.
The concept of mobile agents began to gather momentum in the research and industrial fields starting with the Telescript, an agent platform where migrating active objects as appropriate model for sustaining applications on public networks were launched [13]. Subsequent to Telescript implementation, several diverse projects both at research institutes and in the industry levels began to surface. Some of the popular platforms are Aglets, Agent Tcl, Knowbots, Telescript, Voyager, Mole, Tacoma, Grasshopper, James, Swarm etc [2], [1], [13].

1.4.1 Characteristics of Mobile Agents

Mobile agents exhibit certain characteristics that allows for effective deployment of services and applications in a more dynamic, flexible, and customizable manner in a distributed systems. These features makes them more desirable for building mobile applications the traditional client-server paradigm, etc [2]. These features include:

- Mobile agents naturally operate in heterogeneous environments [9]. They usually operate in wide-area and diverse networks where either the reliability or the security assumptions of the computers and the network connections respectively are not a factor.

- Mobile agents act autonomously on their own on behalf of the user [8]. They have the ability to precisely initiate their migration by itself and execute its owner’s request unlike other traditional software objects that are centrally governed by the core operating system or middleware. They move independently from one host and can effectively make a decision of where to move to in order to resume execution. [14].

- Mobile agents exhibit a multi-hop ability [14]. That is, they are able to travel with their code, data, and execution state more than once by resuming their execution in
another server in the network after completing their tasks in the first server visited.
In contrast, mobile code is transferred only once in other mobile paradigm.

- Mobile agents heavily rely on the underlying protocol for communications by way of interactions and message exchanges in order to successfully carry out and execute certain task in the in the agent system [9]. During migration, mobile agents can communicate synchronously or asynchronously with other agents or the host systems. Others are through network-based mechanism and local inter-process communication mechanism.

1.4.2 Benefits of Mobile Agents

In this section, some of the benefits provided by mobile agents that make them preferable for the creation of distributed systems and particularly mobile agent applications than other paradigms (See figure 1.2). These include:

- They conserve bandwidth especially in applications where there is large amount of data to be processed on different hosts backed by the movement of code to the data rather than the data to the code [15].

- They can exhibits asynchronous and autonomous interaction since they can be delegated to carry out a certain task and with the intelligence embedded on them, can decide on its own whom to interact with at execution time [8].

- They offer extended flexibility in disconnected data operations especially in environments with intermittent connection such as the Internet [16]. Unlike client-server application, it can execute the requested operation on a specified host and wait until connection is available before they migrate to the next host.

- They can improve network latency with better response time than client-server applications since the intermediate results are not sent back and forth between client and server, which may have resulted in delays caused by network congestion [16]. In this case, only the final result of the computation will be transmitted to the server.

- They are robust and fault tolerant because of their ability to dynamically respond to unfavourable conditions and events in a distributed system [17].

- They provide support for heterogeneous environments through their mobility frameworks, which separated the executing host and its operating system. That is, Mobile agents are generally independent of the computer-layer and transport-layer and dependent only on their execution environment [18].

- Mobile agents have better scalability especially with respect to their flexible in dynamic deployment [16]. In client-server development, modification of the client may require redeployment of client installation. In mobile agent systems, on the other hand, the modified agent can be injected once and installs itself wherever it is required to install.
1.4.3 Application Areas of Mobile Agents

Mobile agents have been considered greatly in many application areas such as e-commerce, network monitoring and management, distributed information retrieval, telecommunications, remote device control and configuration, Internet etc [2], [5]. In this section, few examples of each application domain are presented.

1. **E-commerce and M-commerce**: Electronic commerce (e-commerce) involves the buying and selling of goods electronically. Mobile agents play a vital role in e-commerce systems by ways of engaging in transactions, perform negotiations, and products discovery by roaming a market place on behalf of the user [8], [19]. M-commerce (mobile electronic commerce) is similar to e-commerce but involves a mobile device in the execution of the requested task.

2. **Network monitoring and management**: Network management systems are characterized by distributed data processing and decision making, which would create a bottleneck if brought to be managed centrally. However, mobile agents can be delegated the task of network management to perform administrative tasks such as applying network policy, monitor quality of service, etc. In addition to the flexibility, mobile agents offer considerable advantages by conserving network bandwidth, decentralizing computation load, quickly responding and scaling easily [20].

3. **Distributed information retrieval**: The distributed information retrieval domain is characterized by the distributed and heterogeneous computing environment such as the Internet. Mobile agents in such heterogeneous environments can used to access and filtering data, perform local searches and effectively conserve bandwidth by avoiding network transfer of intermediate data [19].

4. **Telecommunication**: The environment of telecommunications has benefited tremendously from the services offered by mobile agents in terms of flexibility [8], [19]. Using mobile agents in telecommunication networks helps to overcome the problem in scalability and traffic congestion, which are typical problems in existing centralized client-server solutions [21]. The technology plays a vital role in both service personalization and wireless network communication system. In service personalization, the application is in the areas of value-
added service, intelligent home, and traffic telematics. On the other hand, in wireless
network communication systems, the application is geared toward reducing load caused by
message exchange between network elements.

5. Remote device control and configuration: Mobile agents in this application domain are
essential in carrying out monitoring activities in critical real-time systems such as intrusion
detection, information filtering and control functions [22]. One beneficial factor here is that
agent’s policies and routes can be handled dynamically.

However, the flexibility offered by mobile agents is not consequentially isolated from
challenges especially when the goals of introducing agents is conquered by malicious or
errant hosts, erratic Internet behaviours or resource scarcity [6]. A failure in a mobile agent
system may result in a partial or total loss of the agent. Therefore, for mobile agent
technology to be fully integrated into the heart of our today’s industrial applications,
reliability and security mechanisms have to be addressed [9], [12]. The reliability issue is
being addressed by fault tolerance mechanisms, which is the focus of this study. Although
agent’s security is an important research area, it is outside of the scope of this study.

1.5 Problem Statements

Although there have been a lot of effort in the area of mobile agent fault tolerance, there is
no comprehensive detail bringing together and summarizing the efforts. The main gap in this
research area is the fact that the existing fault tolerance approaches focus on a specific aspect
of the problem, usually in one or two specific fault modes. In order to improve reliability of a
system, faults that may violate the execution property of an agent system need to be
addressed in the fault tolerance measure. However, there is no fault tolerance framework in
the existing recognized studies that serves as a guideline for realizing the state-of-the-art in
mobile agent fault tolerance.

1.5.1 Aims and Objectives

The aim of this study is to study the state-of-the-art and develop a generic framework for
achieving fault tolerance in mobile agents. During the course of this study, we shall carry out
the following operations to achieve our objectives:

- Conduct a systematic literature review of fault tolerance approaches in mobile agents
  system to identify the current state of research.
- Identify the techniques and approaches used for achieving fault tolerance in mobile
  agents by reviewing the recognised fault tolerance implementations in research.
- Identify the factors that influence execution of fault tolerant mobile agents reported
  in the recognised fault tolerance implementations in research.
- Conduct a literature review to identify platform supports provided and reported in
  current recognized fault tolerant agent platforms.
- Identify the challenges reported in the recognized fault tolerance implementations in
  research.
- Propose a generic framework for mobile agent’s fault tolerance based on the results
  of the systematic review and literature review.

1.5.2 Research Questions

The following are the research questions we intend to answer in this study. They include:

RQ1: What is the state-of-the-art in research in the recognised fault tolerant mobile agents?
Here we will carry out a research on mobile agent’s fault tolerance using comprehensive systematic review, focusing on mobile agent’s fault tolerance implementations over the years in research. This will help to understand the research directions of fault tolerance in mobile agents.

**RQ2:** What are the recognised techniques/approaches in mobile agent fault tolerance in research?

A comprehensive literature review and paper survey will be conducted in the publications identified to be relevant to this question. Based on the current state of research, the recognised techniques/approaches in current state of research will be studied. The protocols used will be analyzed to understand how they work, what factors affect them and the challenges faced in implementing the protocols. Furthermore, the kind of support offered from the agent platform will be analyzed.

**RQ3:** What could be the generic fault tolerance framework that would enable the benefit of utilizing the available protocols while considering both factors and challenges facing the existing approaches?

Given the results obtained from both RQ1 and RQ2, a generic fault tolerance framework will be proposed so as to assist the implementation of fault tolerant mobile agent features.

### 1.5.2.1 Deliverables

The following form our deliverables against each research question:

**D1.RQ1:**
- Discussion of basic concepts of fault tolerance in mobile agents. This includes the basic schemes/techniques used, properties of mobile agent execution and recovery techniques.

**D2.RQ2:**
- Develop a selection criteria to select the approaches
- List of implemented fault tolerance approaches in mobile agent technology
- List of protocol design components
- List of influencing factors influencing execution of the available fault tolerance protocols
- List of platforms used in implementing the available fault tolerance protocols in current state of research
- List of challenges in implementing the available fault tolerance protocols

**D3.RQ3:**
- Develop a generic framework of fault tolerant mobile agent based on some of the results of the systematic review.
- Support the validation of the framework using a cross section of the available representative fault tolerance approaches

**D4.RQ1, 2, 3:** Final draft of thesis report.

### 1.5.2.2 Expected Outcomes

The expected outcome of this study will be a report providing the recognised current state-of-the-art in mobile agent’s fault tolerance. The key outcome to validate the study will be a systematic review that will list and describe the mechanisms to achieve fault tolerance in mobile agents, factors influencing fault tolerant execution, features provided by available
recognised platforms and challenges hindering implementations. Based on the knowledge and result obtained from systematic review/comprehensive literature review, we will propose a generic framework that will implement fault tolerance in mobile agents.

1.5.3 Research Methodology

To satisfy the research questions highlighted and in order to understand the current state-of-the-art in mobile agent’s fault tolerance, a qualitative approach was used. This approach adopted the comprehensive systematic literature review. Systematic literature review is a process of identifying, evaluating and interpreting relevant studies that are in a particular research area of interest [23]. Searching a fact from different perspectives and arriving at the same or similar conclusions through analysis of different data types can also provide a strong credibility of our research findings [24]. The main reasons for conducting systematic literature review are to summarize existing findings, to identify gaps in existing research and to provide framework for new research activities [23].

A survey of research papers relevant to our studies was carried out in different information sources in order to identify acceptable researches in the field. A paper sample based on a number of criteria overwhelmed by sources of publication and year of publication was used. The sample comprised published papers from various recognised sources which span between the periods 1998 to 2008. The rationales behind the survey conduction are as follows:

I. To enhance the credibility and accuracy of the findings written in this work.
II. To obtain information only from peer-reviewed publications and recognised sources.

The systematic review was carried out by utilizing some of the guidelines laid down in the Kitchenham technical report [23]. Moreover, the systematic review guideline aided us more in establishing facts in the directions of fault tolerance in mobile agents, recognised techniques/approaches, influencing factors, platform supports and challenges.

The platforms support was further investigated by reviewing literatures and product manuals/documentations. We believe the additional literature review activity, for investigating more on the platform support, does not introduce bias since the list of the platforms is going to be generated as a result of the systematic review. We feel the additional resources will strengthen the information about available support offered by the existing platforms. Based on the result obtained from the systematic review and the analysis of existing studies, this study proposed a generic framework that will serve as a guide for commencing fault tolerance implementations for mobile agent.
The research domain of this study is mobile agent technology, which is a branch of distributed computing. The study will be giving special attention to fault tolerance in mobile agents. Faults may originate from a variety of sources during mobile agent execution. This study will limit the fault tolerance aspect to the reliable execution of mobile agent. Fault tolerance measures to enhance mobile agent’s reliability by overcoming faults originating from security will not be considered in this study.

This study will explore the existing mobile agent’s fault tolerance approaches/implementations with respect to the mechanism they achieve fault tolerance. The study does not analyze the implementations (codes/application) rather make a literature review of the approaches.

Based on the characteristics of fault tolerance elements, the study will consider design features provided by the existing mobile agent’s platforms that assist fault tolerance implementations. The study does not focus on any specific mobile agent platform but considers both academic and commercial platforms with certain fault tolerance capability.

In this study, the context of mobile agents includes agent systems with co-operating multiple agents that communicate with other agents. The failure of one of the cooperating agents can be viewed as failure of an agent. Moreover, the phrases ‘agents’ and ‘mobile agents’ are used interchangeably and refer to the same thing in this study. However, the phrase ‘mobile agent system’ and ‘agent systems’ refers to a system composed of mobile agents, their platform and the operating system altogether.

### 1.5.5 Terminology

<table>
<thead>
<tr>
<th>Terms</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td>Fault tolerance is the ability of a system to withstand failure and continue to provide service in the event of an internal or external error. Fault tolerant systems are designed to ensure that in the event of a failure, crash, or a major user error, data is not lost and the system can continue to provide its specified</td>
</tr>
<tr>
<td><strong>services.</strong></td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>Mobile agent</td>
<td>Mobile agents are encapsulated pieces of software containing code and data that are able to migrate from one host to another and perform certain task autonomously [3]. Mobile agents are technological paradigms that aim at shifting computation towards the data rather than data to the computation [1].</td>
</tr>
<tr>
<td>Message size</td>
<td>Mobile agents communicate with each other and with the controlling process via messaging. The message size is the size of the information exchanged between the agents or the controlling process.</td>
</tr>
<tr>
<td>Network Partitioning</td>
<td>A network partition is the partitioning of the network into separate boundaries due to communication failure.</td>
</tr>
<tr>
<td>Primary Agent</td>
<td>Primary agent is the main agent that is responsible for executing a certain task.</td>
</tr>
<tr>
<td>Clone Agent</td>
<td>Clone Agent is an exact copy of the primary agent that serves as a standby and can take the place of the main agent in case of primary agent failure.</td>
</tr>
<tr>
<td>Witness Agent</td>
<td>Witness Agent is an agent that works in collaboration with the primary agent but it is not an exact copy of the primary agent. Witness agents have designated task of monitoring primary agents but can not take over the task execution of primary agents.</td>
</tr>
<tr>
<td>Scalability</td>
<td>Scalability is the ability of a fault tolerant design to scale with growing number of agents.</td>
</tr>
<tr>
<td>Transparency</td>
<td>Fault tolerance features in some cases are wrapped as services and the user of the services may not need to explicitly manage the fault tolerance feature such as checkpointing during migration. The user of the transparent checkpoint service in this case only specifies migration while the service takes care of the checkpoint management.</td>
</tr>
</tbody>
</table>
CHAPTER 2

SYSTEMATIC LITERATURE REVIEW

2.1 Overview

Research literature reviews provides the usual means of gathering and interpreting a collection research findings conducted in a given research field. Review results usually offers the best guide for further research as well as providing a state-of-the-art description of the strengths, weaknesses, and findings in a research field, which in turn can be used for decision making. However, until last decade literature review has been considered as falling short of basic scientific standard. This is due to the fact that research methods weren't strictly applied thereby leading to biased results [23].

The systematic literature review methodology was introduced in order to get rid of or minimizes the inconsistencies associated with less scientifically rigorous review methodologies through strict qualitative research methods resulting to objective/unbiased results. Systematic literature review is a methodologically rigorous review of research results that provides a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest [23]. This methodology is usually prepared by a team of at least two reviewers embedded with a thorough understanding of both the knowledge area and the review methodology with the overall intent to minimizing human error and bias [23].

A systematic review however, is a form of secondary study stemming out of individual studies that contributes to primary studies [23]. The process involve in conducting a systematic literature review is not an easy job especially when the whole process must be precisely defined and documented, and must contain intermediate results. The approach is surrounded with several distinct activities, which are grouped into three main phases of planning, conducting, and reporting of the review [23]. However, a pilot review is suggested for the reviews as well as various research questions or a huge set of primary studies. The stages involve are viewed as being sequentially treated but a number of iterations are indispensable in achieving the desire results.

The sections below show a detailed elucidation of how each of these activities was carried out. Following the Kitchenham et al. [23] guidelines, features that differentiated systematic literature reviews from standards literature reviews are identified.

2.2 Planning the Review

2.2.1 Review Objectives

This systematic review aimed to summarize the existing mobile agent’s fault tolerance approaches in recent years. The systematic review would provide a list of reported and recognized techniques and approaches, influencing factors, platform supports and challenges in mobile agent’s fault tolerance. The results of the systematic review will be helpful in identifying the current state-of-the-art and developing the generic framework for mobile agent’s fault tolerance.
2.2.2 Review protocol

The review protocol constitutes a programmed procedure that describes the methods to be used in carrying out the systematic review. The protocol aimed at specifying the methods for selecting the primary studies as well as to make it unbiased [23]. In essence, it is designed to highlight details on the way judgements on the inclusion of individual papers will be made as well as the inclusion of search strategy for discovering studies relevant to stated review questions.

2.2.2.1 Background

Systematic reviews are mostly used in the field of medicines and are regarded as the strongest form of medical evidence [23]. Nevertheless, the success of evidence-based medicine has prompted many other disciplines like software engineering, etc to adopt a similar approach in their research. In 2004, Kitchenham et al. [23] recommended that software engineering researchers should adopt “Evidence-Based Software Engineering” (EBSE). This means that an evidence-based approach research should be applied to software engineering research. EBSE was introduced based on a number of reasons out of which one is to provide “a common goal for individual researchers and research groups to ensure that their research is directed to the requirements of industry and other stakeholder groups” [23]. Evidence in this perspective is seen as a synthesis of best quality scientific studies on a specific topic or research question [23]. It is this method of synthesis that is referred to as systematic literature review.

Systematic literature review has been employed by a great number of researchers in diverse discipline to summarize existing information concerning some phenomenon in a detailed and unbiased way. However, in the context of this study, the systematic review aimed to summarize the existing mobile agent’s fault tolerance approaches. The systematic review would provide a list of reported and recognized techniques/approaches, influencing factors, platform supports and challenges in mobile agent’s fault tolerance. The results of the systematic review will be helpful in identifying the current state-of-the-art and developing the generic framework for mobile agent’s fault tolerance.

2.2.2.2 Review Questions

In this research work, the main research questions are broken down into the following systematic review research question intended to be answered in order to guide the systematic review. Moreover, the systematic review research questions make the process of addressing the core research question manageable and easier to understand.

SRQ1. What is the state-of-the-art in research in the recognised mobile agent’s fault tolerance?

SRQ2. What available fault models are considered in designing fault tolerance protocols?

SRQ3. What are the available approaches and their design elements in the available recognised mobile agent’s fault tolerance in current state of research?

SRQ4. What factors influences mobile agent’s fault tolerance execution?
SRQ5. How much supports are offered by the mobile agent’s platforms used in implementing the fault tolerance features?

SRQ6. What challenges exist and how do they affect the implementation of the fault tolerance in mobile agents?

Table 2.1: Systematic Review Research Questions Mapping

<table>
<thead>
<tr>
<th>Research Questions</th>
<th>Systematic Review Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1</td>
<td>SRQ1</td>
</tr>
<tr>
<td>RQ2</td>
<td>SRQ2, SRQ3</td>
</tr>
<tr>
<td>RQ3</td>
<td>SRQ4, SRQ5, SRQ6</td>
</tr>
</tbody>
</table>

Although the systematic review questions seem independent questions, the output from each question contributes to sufficiently address the core research questions, which are the focus of this study. For instance, RQ3 is addressed by combining the outputs from SRQ1 to SRQ6. Furthermore, after carrying out the systematic review we expect the followings:

- Relevant information concerning the present state-of-the-art with regard to fault tolerant mobile agent implementation.
- Identify gaps in current research, solutions, trends and future research within this scope.
- Give recommendations regarding the features that should be incorporated into the fault tolerance approach in order to provide suitable and extensible support for reliable mobile agent execution. These will be extracted from the conclusions of the SR.

2.2.2.3 Strategy

This study considers the review of 10-years’ efforts in fault tolerance mobile agents, between the period from January 1998 to December 2008. This period was selected in order to amass sufficient information that can enrich this study and give a reflection of the current trend in mobile agent fault tolerance. Although the concept of mobile agents originated in the 1970s, the number and quality of studies concerning mobile agent fault tolerance in the years prior to 1998 were limited since the mobile agents fault tolerance research field was at its infancy stage. The year 2009 was excluded since the year has not yet finished; hence it is difficult to conclude any statistical data for 2009. Furthermore, the periods were favourable for reflecting the trends in mobile agent fault tolerance and sufficiently explore the scope that surrounded this study. Subsequent sections highlighted the strategies involve in achieving the set objectives.

2.2.2.3.1 Search Terms

Table 2.2 contain the available search terms/strings to be use in this work to obtain the essential information vital to our study. There are used manually in the search process. Each search terms listed will be applied on each of the search resources selected for this systematic review process. Generic terms such as “fault tolerance” were not used since they are too general to produce a relevant result. For example, the trial search for “fault tolerance” produced results in almost all computing areas like object oriented programming, hardware, embedded systems, security, etc.

Table 2.2: Search Terms
### 2.2.2.3.2 Resources Used

The sources selected for use in the search for the primary studies were part of those recommended by Kitchenham et al. [23] and those provided by the BTH Library Service. We considered them appropriate for this review due to the fact that they contain peer-reviewed works published in journals, digital libraries, conferences, proceedings and workshops which are of recognized quality within the research community. A trial search was conducted in all of the search resources to assess the quantity of the results. The relevance of the results was assessed based on the trail search result. The following databases were selected for searching the research articles for this study:

1. Compendex
2. Inspec
3. IEEE
4. Google Scholar
5. ACM
6. Springer
7. Scirus (only journals)

Sources such as Yahoo, Google and CiteseerX were not included since they contain huge amount of irrelevant information that do not meet the requirements of this review. The search results from Compendex, Inspec and IEEE were more relevant to the search terms and the number of the results was not too much. However, the search results from Google Scholar, ACM, Springer and Scirus contained a lot of irrelevant publications after a few pages and they produced enormous number of results. As a result, results from Google Scholar, ACM, Springer and Scirus were partially reviewed (see Table 2.3). The number of search results in Table 2.3 is not reproducible since the search index in the database resources change continuously. Scirus gives search result from web sources and journal sources. Only journal results were considered from Scirus so as to narrow down the number of results. Furthermore, for Google Scholar and ACM, the first 100 results were reviewed, whereas the first 50 results were considered for Springer and Scirus since the latter databases produced more irrelevant result after the first few pages. The review process did not specifically focus on certain journal or conference proceedings. The reason for not focusing on specific journal or conference is because the database resources mentioned above either have indexes for the conferences and journals or do the search activity in conferences and journals databases.

#### Table 2.3: Search Results Consideration

<table>
<thead>
<tr>
<th>No.</th>
<th>Search term</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mobile agent AND fault tolerance</td>
</tr>
<tr>
<td>2</td>
<td>mobile agent AND fault protocol</td>
</tr>
<tr>
<td>3</td>
<td>fault tolerance design in mobile agent</td>
</tr>
<tr>
<td>4</td>
<td>state of the art in fault tolerant mobile agent</td>
</tr>
<tr>
<td>5</td>
<td>mobile agent fault tolerance survey</td>
</tr>
<tr>
<td>6</td>
<td>fault tolerant mobile agent platform</td>
</tr>
<tr>
<td>7</td>
<td>issues mobile agent fault tolerance</td>
</tr>
<tr>
<td>8</td>
<td>challenges mobile agent fault tolerance</td>
</tr>
<tr>
<td>9</td>
<td>challenges mobile agent fault tolerance platform</td>
</tr>
<tr>
<td>10</td>
<td>checkpoint mobile agent fault tolerance</td>
</tr>
<tr>
<td>11</td>
<td>replication mobile agent fault tolerance</td>
</tr>
<tr>
<td>12</td>
<td>performance mobile agent fault tolerance</td>
</tr>
<tr>
<td>13</td>
<td>fault model in mobile agent</td>
</tr>
<tr>
<td>14</td>
<td>mobile agent fault tolerance mode</td>
</tr>
<tr>
<td>15</td>
<td>mobile agent fault tolerance framework</td>
</tr>
</tbody>
</table>

Sources such as Yahoo, Google and CiteseerX were not included since they contain huge amount of irrelevant information that do not meet the requirements of this review. The search results from Compendex, Inspec and IEEE were more relevant to the search terms and the number of the results was not too much. However, the search results from Google Scholar, ACM, Springer and Scirus contained a lot of irrelevant publications after a few pages and they produced enormous number of results. As a result, results from Google Scholar, ACM, Springer and Scirus were partially reviewed (see Table 2.3). The number of search results in Table 2.3 is not reproducible since the search index in the database resources change continuously. Scirus gives search result from web sources and journal sources. Only journal results were considered from Scirus so as to narrow down the number of results. Furthermore, for Google Scholar and ACM, the first 100 results were reviewed, whereas the first 50 results were considered for Springer and Scirus since the latter databases produced more irrelevant result after the first few pages. The review process did not specifically focus on certain journal or conference proceedings. The reason for not focusing on specific journal or conference is because the database resources mentioned above either have indexes for the conferences and journals or do the search activity in conferences and journals databases.
<table>
<thead>
<tr>
<th>No.</th>
<th>Search term</th>
<th>Source</th>
<th>Result</th>
<th>Considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mobile agent AND fault tolerance</td>
<td>Compendex</td>
<td>334</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inspec</td>
<td>393</td>
<td>393</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google Scholar</td>
<td>15,500</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scirus (journals)</td>
<td>423</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IEEE</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACM</td>
<td>493</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Springer</td>
<td>1150</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>mobile agent AND fault protocol</td>
<td>Compendex</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inspec</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google Scholar</td>
<td>14,600</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scirus (journals)</td>
<td>265</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IEEE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACM</td>
<td>684</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Springer</td>
<td>1583</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>fault tolerance design in mobile agent</td>
<td>Compendex</td>
<td>88</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inspec</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google Scholar</td>
<td>14,300</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scirus (journals)</td>
<td>367</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IEEE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACM</td>
<td>458</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Springer</td>
<td>968</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>state of the art in mobile agent fault tolerance</td>
<td>Compendex</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inspec</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google Scholar</td>
<td>8,480</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scirus (journals)</td>
<td>117</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IEEE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACM</td>
<td>128</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Springer</td>
<td>274</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>mobile agent fault tolerance survey</td>
<td>Compendex</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inspec</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google Scholar</td>
<td>6,750</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scirus (journals)</td>
<td>182</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IEEE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACM</td>
<td>201</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Springer</td>
<td>430</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>fault tolerant mobile agent platform</td>
<td>Compendex</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inspec</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google Scholar</td>
<td>7950</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scirus (journals)</td>
<td>134</td>
<td>50</td>
</tr>
<tr>
<td>Issue</td>
<td>Subsection</td>
<td>Compendex</td>
<td>Inspec</td>
<td>Google Scholar</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-----------</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>7</td>
<td>issues mobile agent fault tolerance</td>
<td>55</td>
<td>78</td>
<td>300</td>
</tr>
<tr>
<td>8</td>
<td>challenges mobile agent fault tolerance</td>
<td>24</td>
<td>24</td>
<td>4990</td>
</tr>
<tr>
<td>9</td>
<td>challenges mobile agent fault tolerance platform</td>
<td>4</td>
<td>4</td>
<td>95</td>
</tr>
<tr>
<td>10</td>
<td>checkpoint mobile agent fault tolerance</td>
<td>28</td>
<td>28</td>
<td>3,980</td>
</tr>
<tr>
<td>11</td>
<td>replication mobile agent fault tolerance</td>
<td>34</td>
<td>34</td>
<td>86</td>
</tr>
<tr>
<td>12</td>
<td>performance mobile agent fault tolerance</td>
<td>119</td>
<td>119</td>
<td>0</td>
</tr>
<tr>
<td>Study Selection Criteria and Procedure</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2.2.2.4 Studies selection process</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Based on the source selection already defined, the review process aims to discover those studies that provided direct evidence with regard to our stated research question. This is done by defining a basic inclusion and exclusion criteria in accordance with the research question, along with a procedure through which this selection was applied. This selection process is explained in subsequent sections.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2.2.2.4.2 Definition of inclusion and exclusion criteria</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a.) Inclusion criteria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>We considered articles on mobile agent’s fault tolerance available and published between January 1998 and December 2008 inclusive. The criteria vital to aid the suitability judgement of all inclusive articles in the systematic review are as follows:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Studies conducted in the past 10 years i.e. between 1998 and 2008
2. Articles written in English language
3. Articles discussing reliability of mobile agent
4. Articles discussing fault tolerance model in mobile agent systems
5. Articles on mobile agent platforms and reliability
6. Articles discussing issues and challenges in mobile agent systems

b.) Exclusion criteria
We considered article exclusive based on the following criteria:

1. Studies conducted more than 10 years ago
2. Articles discussing failures due to security
3. Articles discussing application of mobile agents for fault tolerance in another study area
4. Articles that focus on exception handling
5. Articles that are not subject to peer-review
6. Articles that use formal methods to describe the fault tolerance model, which is difficult to compare and contrast with other studies.

The exclusion criteria were created after conducting a trial search using some of the search terms on the database resources. Although the concept of mobile agents was around since the 70s, the fault tolerance aspect and their implementations were closely studied in the past 10 years. Studies discussing security and exception handling are out of the scope of this study. Faults originating from security are not in the scope of this study. Exception handling is a curative form of fault tolerance but the focus of this study is on the preventive aspect. Studies that discuss formal models are difficult to understand and transform their result into the descriptive model that is comparable to other prototype implementations.

2.2.2.5 Quality Assessment
The selected research articles will be evaluated against a number of checklist questions. The definition of quality in this context is a measure of the presence of detailed and supporting information in the studies that aids in answering the research questions. For example, studies that support their findings with experimental data are considered to be more concrete than model simulations. The assessment is a form of internal validation so as to minimize bias. Both authors will conduct the assessment first separately then check together to reach consensus on the differences. A description of the checklist questions and evaluation method for the assessment are given below. Each question has three options to choose from with different weight for each option. The weights for the options are: Yes=1, Partial=0.5 and No=0. The maximum score a certain publication can get is 8.

Table 2.4: Quality Assessment Checklists

<table>
<thead>
<tr>
<th>Quality Question</th>
<th>Choice (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Article discusses a fault tolerance model</td>
<td>Y: yes (discusses the model for detecting and recovering)</td>
</tr>
<tr>
<td></td>
<td>P: partly (discusses the model for either detecting or recovering)</td>
</tr>
<tr>
<td></td>
<td>N: no</td>
</tr>
</tbody>
</table>

The exclusion criteria were created after conducting a trial search using some of the search terms on the database resources. Although the concept of mobile agents was around since the 70s, the fault tolerance aspect and their implementations were closely studied in the past 10 years. Studies discussing security and exception handling are out of the scope of this study. Faults originating from security are not in the scope of this study. Exception handling is a curative form of fault tolerance but the focus of this study is on the preventive aspect. Studies that discuss formal models are difficult to understand and transform their result into the descriptive model that is comparable to other prototype implementations.

2.2.2.5 Quality Assessment
The selected research articles will be evaluated against a number of checklist questions. The definition of quality in this context is a measure of the presence of detailed and supporting information in the studies that aids in answering the research questions. For example, studies that support their findings with experimental data are considered to be more concrete than model simulations. The assessment is a form of internal validation so as to minimize bias. Both authors will conduct the assessment first separately then check together to reach consensus on the differences. A description of the checklist questions and evaluation method for the assessment are given below. Each question has three options to choose from with different weight for each option. The weights for the options are: Yes=1, Partial=0.5 and No=0. The maximum score a certain publication can get is 8.

Table 2.4: Quality Assessment Checklists

<table>
<thead>
<tr>
<th>Quality Question</th>
<th>Choice (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Article discusses a fault tolerance model</td>
<td>Y: yes (discusses the model for detecting and recovering)</td>
</tr>
<tr>
<td></td>
<td>P: partly (discusses the model for either detecting or recovering)</td>
</tr>
<tr>
<td></td>
<td>N: no</td>
</tr>
<tr>
<td>Q2: Article consists of real experiment rather than simulation</td>
<td>Y: Experiment</td>
</tr>
<tr>
<td>Q3: Article consists of experimental data to validate and demonstrate the performance of the model</td>
<td>Y: yes (measures performance)</td>
</tr>
<tr>
<td>Q4: Article discusses about design approach for achieving fault tolerance</td>
<td>Y: yes (discusses the design for detecting and recovering)</td>
</tr>
<tr>
<td>Q5: The model has been implemented for large scale use</td>
<td>Y: yes</td>
</tr>
<tr>
<td>Q6: The assumptions of the model are stated</td>
<td>Y: yes (clearly stated probably in separate section)</td>
</tr>
<tr>
<td>Q7: The platform used along with its support is discussed</td>
<td>Y: yes (platform and its support are discussed)</td>
</tr>
</tbody>
</table>

The inclusion and exclusion criteria were strictly adhered to. Based on the used sources, Table 2.5 below shows the list of selected Journals and Conferences papers considered in the systematic review.

**Table 2.5:** List of Selected Publications
<table>
<thead>
<tr>
<th>Reference ID</th>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
<th>Q. Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>[26]</td>
<td>2005</td>
<td>Leung, Kwai Ki</td>
<td>A fault-tolerance mechanism for mobile agent systems</td>
<td>6</td>
</tr>
<tr>
<td>[27]</td>
<td>2003</td>
<td>Lyu, M.R.</td>
<td>A progressive fault tolerant mechanism in mobile agent systems</td>
<td>5.5</td>
</tr>
<tr>
<td>[29]</td>
<td>1998</td>
<td>Assis, Silva, F.M.;</td>
<td>An Approach for Providing Mobile Agent Fault Tolerance</td>
<td>3</td>
</tr>
<tr>
<td>[31]</td>
<td>2006</td>
<td>Meng, Xuejun</td>
<td>An efficient fault-tolerant scheme for mobile agent execution</td>
<td>5</td>
</tr>
<tr>
<td>[32]</td>
<td>2008</td>
<td>Jong-Shin Chen</td>
<td>An efficient forward and backward fault-tolerant mobile agent system</td>
<td>3</td>
</tr>
<tr>
<td>[33]</td>
<td>2005</td>
<td>Marin, Olivier</td>
<td>DARX - A Self-healing Framework for Agents</td>
<td>8</td>
</tr>
<tr>
<td>[34]</td>
<td>2004</td>
<td>Lyu, Michael R.</td>
<td>Design and evaluation of a fault-tolerant mobile-agent system</td>
<td>4.5</td>
</tr>
<tr>
<td>[35]</td>
<td>2006</td>
<td>Youhei Tanaka</td>
<td>Fault-Tolerant Distributed Systems in a Mobile Agent Model</td>
<td>3</td>
</tr>
<tr>
<td>[38]</td>
<td>1998</td>
<td>Rothermel, Kurt</td>
<td>Fault-tolerant protocol for providing the exactly-once property of mobile agents</td>
<td>3.5</td>
</tr>
<tr>
<td>[40]</td>
<td>2004</td>
<td>Taesoon Park</td>
<td>Lazy Agent Replication and Asynchronous Consensus for the Fault-Tolerant Mobile Agent System</td>
<td>5.5</td>
</tr>
<tr>
<td>[41]</td>
<td>2004</td>
<td>Taesoon Park</td>
<td>Low Overhead Agent Replication for the Reliable Mobile Agent System</td>
<td>6</td>
</tr>
<tr>
<td>[44]</td>
<td>2005</td>
<td>Yang, Jin</td>
<td>Parallel algorithms for fault-tolerant mobile agent execution</td>
<td>5</td>
</tr>
</tbody>
</table>
2.2.2.6 Data Extraction Strategy

The objective of the data extraction process is to consistently and accurately record the available information that is obtained from primary studies [23]. The reporting of multiple publication of the same result is a potential threat to unbiased reporting of findings. If a duplication effect is discovered, then the publication with the most complete and latest information would be used.

Based on how these studies were to be selected, we considered it vital to develop data extraction strategy that document how the information required from each primary study will be obtained. The extraction strategy was developed inline with the research questions defined in Section 2.2.2.2, quality assessment checklist, general information associated with the study identification and certain common characteristics in the studies. The major attributes to be collected for each study through the systematic review are listed below in the data extraction form.

Table 2.6: Data Extraction Form

<table>
<thead>
<tr>
<th>No.</th>
<th>Extraction Fields</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Extract document information</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Title</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Authors</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Publisher</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Published in:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i. Journal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ii. Conference</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iii. Workshop</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Year</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Research Methodology:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i. Model based</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ii. Real Experiment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iii. Simulation</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Scheme</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Fault model</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Assumptions</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Detection</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Recovery</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Fault Tolerance execution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i. Transactional</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ii. Non-transactional</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Agent types</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Communication:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i. Asynchronous</td>
<td></td>
</tr>
</tbody>
</table>
Factors affecting the performance of the proposed model and experiment variables:

- Platform type
- Platform support
- Challenges

Summary of the paper

Quality assessment score

### 2.2.2.7 Data Synthesis

The data synthesis stage involves the collection and summarization of the findings from the associated primary studies [23]. This study follows a descriptive (qualitative) synthesis approach to synthesize the extracted data due to the heterogeneous nature of the information gathered through systematic review [23]. The concepts in the findings of the primary studies would be integrated by using descriptive language. The main challenge in the area of mobile agent’s fault tolerance is that researchers combine a variety of core protocols and use different naming and descriptions when describing their findings. Therefore, the authors of this study will state the integrated result with respect to the research questions given in the review protocol and following the repetition pattern of findings in the high quality studies. The authors believe the effect of low quality studies that were excluded in the review process in the final outcome would be negligible. This is because excluded publications such as the ones that use formal method do not contribute much to the data synthesis part since the information they contain is not similar in nature and comparable to the other primary studies.

### 2.3 Conducting the Review

The basic steps associated with conducting the actual review using the above described protocols is given below:

#### 2.3.1 Identification of Research

Systematic review is a research method for objectively reviewing a subject matter in order to produce an accurate, unbiased, and state-of-the-art summary of the research on that subject [23]. This involves searching several primary studies that are related to the research question using an unbiased search strategy defined in the review protocol.

The search strategy was developed based on our research skills experiences from related courses such as research methodology and from feedback offered from the BTH librarian. The search strategies adopted are iterative in nature. However, to define the search terms, the research questions are divided into individual facets like comparison, outcomes, context, study designs a similar approach adopted in [23]. Also, the key words, synonyms and abbreviations of the search terms were also considered. Other terms based on the subject headings used in journals and databases were considered. Boolean phrases, AND and Or, were used in constructing the search strings or terms to obtain better search results.
2.3.2 Selection of Primary Studies

The selection of the primary study adopted a multi-stage process consisting following the guidelines stated in Kitchenham et al. [23] using different selection criteria:

i. In the first stage, a preliminary set of studies was conducted by the reading of title and abstract of all the papers selected in accordance with the inclusion and exclusion criteria defined in the review protocol. This was performed by running the search string on the selected sources and discarding irrelevant, non-English or papers not clearly linked to any aspect of the research question were rejected. The first stage of the systematic review was conducted into two reviewing sub-processes. In the first sub-process, the search sources were divided among the two authors of this study and all the search terms were run separately. In the second sub-process, the authors merged the individually generated results list and once more reviewed the title and abstract together.

ii. In the second stage, the exclusion criteria were based on the following reasons: inaccessibility, formal or mathematical description, and exception handling. At this stage, all the papers, excluded were kept aside for future references and did not form part of the review papers. The main reason for excluding studies describing mathematical descriptions is because the results in such studies are difficult to compare and contrast with the other studies. Furthermore, the studies with mathematical descriptions we came across do not contain implementations to support the proofs.

iii. In the third stage, the selection process was based on detailed research questions while the exclusion criteria was based on issues of repetition, application of mobile agents in a different study area, mobile agent’s platforms papers.

Moreover, a list of articles considered not meeting the inclusion and exclusion criteria and which had been excluded as a result was created in accordance with [23]. Accordingly, the list did not include those papers considered completely irrelevant that were clearly excluded in the first stage in accordance with the exclusion criteria.

2.3.3 Selection Execution

Here the review process is documented in details in accordance with [23]. Based on the multi-stage process our studies assumed, the selection execution produced various lists of studies output for each stage of the selection procedure. The information collected for each stage of the studies selection procedure is presented as follows:

i. During the running of the first searching sub-process, a total of 7,001 publications were skimming through their title or abstract to produce 113 potential publications. In the second sub-process, the authors reviewed the 113 potential publications together by going through their title and abstract to accept 86 of the 113 potential publications. This list is not included in this report due to its size.

ii. The second stage was applying the exclusion criteria to the selected publication by the end of the first stage. The exclusion criteria were applied together by both authors to produce a list of 55 publications out of the 86 publications selected in the first stage. In this stage, the abstract, the conclusion and in some cases the introduction was reviewed to apply the exclusion criteria. This stage produced all the studies that satisfy any one of the exclusion criteria. These lists found in the Appendix of this work and are made up of the excluded article with their exclusion reason.
iii. The third stage was thoroughly reviewing the publications according to their suitability for addressing some of the research questions. This stage was again conducted by both authors together on the list of publications that were selected at the end of the second stage. The third stage produced as output a list of 26 unique studies that are suitable for the systematic review. In this stage, similar studies by the same authors were rejected by taking one representative study that is the most complete and usually the latest by the authors was taken. The list of rejected papers along with the reason for rejection is included in the Appendix. However, some of the excluded studies were used in filling gaps during data extraction and during the write-up. For example, studies that were considered to be repetitions provided additional information to the main study during data extraction. This is because some of the studies discuss different details of their experiment in different article but the core model they used can be the same.

**Table 2.7: Selection Execution Results Count**

<table>
<thead>
<tr>
<th></th>
<th>Accepted</th>
<th>Rejected</th>
<th>Total Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1</td>
<td>86</td>
<td>27</td>
<td>113</td>
</tr>
<tr>
<td>Stage 2</td>
<td>55</td>
<td>31</td>
<td>86</td>
</tr>
<tr>
<td>Stage 3</td>
<td>26</td>
<td>29</td>
<td>55</td>
</tr>
</tbody>
</table>

**Figure 2.1: Studies by year of publication**

We can notice from Figure 2.1, that the distribution of the publications by year shows sharp increase between 2003 and 2004. Furthermore, more than 65% of the selected studies were conducted between 2004 and 2006. The reason for the spike can be attributed to the increased interest in the area of mobile agents from the research community. FIPA, the mobile agent standardizing body, adopted robust models that persist even when technology progresses and included mapping of commonly used technologies in 2002 [37]. However,
the trend seems to drop recently, resulting in only 23% of the studies being conducted in the last three years, but the presence of studies in this period can be regarded as an indication to the research field being still active. A possible explanation for the drop can be related to the problem of trust in mobile agents. The reception in industry was shadowed by the fact that mobile agents closely resemble to network worms. The popularity of mobile agents did not continue to grow due to the increasing threat from viruses and network worms.

2.3.4 Study Quality Assessment

The primary studies quality assessment was carried out using the checklist of questions shown in Table 2.4 in order to strengthen the inclusion/exclusion criteria. Although the detail score for each study is not shown in this research work, we only present the normalized quality score per selected source and year in Figure 2.2.

![Publications by Quality Assessment Score](image)

**Figure 2.2:** Publications by Quality Assessment Score

Quality in systematic review context is an attribute that is difficult to measure. It is merely the relative comparison of the amount of detail in the selected studies based on the extracted data. The lack of information in a particular section of the study does not necessarily mean that the study has not been done for that specific part. To that respect, we tried to assess related studies by the same authors to gather the missing parts.

More than 75% of the selected publications scored 4.5 of more. Publications that score more than 4 in the quality assessment generally have most of the vital information needed in the data extraction form. They are characterized by having a detailed description of a model and some form of proof in the form of a result from a real experiment or simulation to support their findings. Majority of the primary studies discuss a fault tolerance model and supported their claim with an experiment or simulation. Half of the primary studies have built prototypes and conducted experiment, while 31% conducted simulation (see Figure 2.3). The higher number of experiment based primary studies increases the credibility of the results and supports the validity of the data in this study.


Figure 2.3: Experiment Methods in Primary Studies

2.3.5 Data Extraction and Execution

The review in this research work took place in June 2009 and only search outputs covered by the years 1998-2008 were selected and analyzed in the study. Table 2.8 shows a summary of the studies selected and analyzed for each source in accordance with selection procedure. The data extraction phase was executed by following the strategy and using the forms designed in the review protocol.

Table 2.8: List of Selected Publications by Publisher and Methodology

<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Year</th>
<th>Publisher</th>
<th>Published in</th>
<th>Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>[26]</td>
<td>Leung, Kwai Ki</td>
<td>2005</td>
<td>IEEE</td>
<td>Conference</td>
<td>Model and Simulation</td>
</tr>
<tr>
<td>[27]</td>
<td>Lyu, M.R.</td>
<td>2003</td>
<td>IIIS</td>
<td>Conference</td>
<td>All</td>
</tr>
<tr>
<td>[28]</td>
<td>Sehl Mellouli</td>
<td>2007</td>
<td>Springer</td>
<td>Conference</td>
<td>Model and Experiment</td>
</tr>
<tr>
<td>[31]</td>
<td>Meng, Xuejun</td>
<td>2006</td>
<td>IEEE</td>
<td>Conference</td>
<td>Model and Experiment</td>
</tr>
<tr>
<td>[33]</td>
<td>Marin, Olivier</td>
<td>2005</td>
<td>Springer</td>
<td>Workshop</td>
<td>Model and Experiment</td>
</tr>
<tr>
<td>[34]</td>
<td>Lyu, Michael R.</td>
<td>2004</td>
<td>IEEE</td>
<td>Journal</td>
<td>Model and Simulation</td>
</tr>
</tbody>
</table>
The authors of this study developed the data extraction strategy in accordance with the review protocol. During the course of the extraction phase, first the authors individually filled the data extraction forms and quality assessment forms. The process was repeated at least two more times in order to ensure relevant information were not missed out. The strategy adopted certifies that all important information such as experimental methodology, recognized fault tolerance techniques/approaches, influencing factors, platform supports and challenges in mobile agent’s fault tolerance was extracted from the research articles. Once the authors finish filling the data extraction forms individually, both authors compared the results and reached a consensus for the different outcomes. Once both authors agreed on the differences, the data extraction forms were divided among the two authors and each author conducted a validation on the extracted data so as to fulfil inter-study consistency.

2.3.6 Data Synthesis

The data synthesis process consists of the collection and summarization of the outputs of the underlying primary studies [23]. The extracted data undergoes synthesis to specifically address the stated research questions in the review protocol. Data syntheses can either be descriptive synthesis, quantitative synthesis or qualitative synthesis [23]. However, considering the nature of the systematic review coupled with the nature of the software engineering area, this study adopted the qualitative data synthesis. Descriptive qualitative data synthesis is suitable method in such heterogeneous data format.

2.4 Reporting the Review

Following the guidelines that surrounded the systematic review, reporting the result of the systematic review phase comes immediately after the actual SR conduction phase. The report is expected to be transparent and reproducible to reveal the basic distinction between a systematic review and the traditional review as well as showing the reliability and relevancy of the review findings. In this work, the results of the systematic review are carefully reported in this phase in accordance with the underlying research questions as defined in the review protocol. The relevant information answering the research questions were collected during the course of the systematic review using the data extraction forms. Finally the
collected data were synthesized and reported using the adopted qualitative data synthesis approaches in order to enhance and comprehension and future decision making. The main characteristics of the current state in research are reported as a result of the qualitative data synthesis along with findings from the specific information that addressed the research questions.
CHAPTER 3

SYSTEMATIC REVIEW RESULTS

This section presents a summary of the results obtained in the systematic review along with the analysis of the data collected. Based on the result, in order to identify the current approaches state-of-the-art of mobile agent’s fault tolerance, a generic fault tolerance framework and recommendations which can be used to future research in the area was proposed.

With the completion of the extraction execution phase, the study considered it vital to get rid of multiple publications of the same approach from the data analysis. Accordingly, articles that were found in multiple sources were accounted for only once. After the replicated papers had been eliminated, the different papers that contain same data or approach were grouped together in order to avoid bias of the data synthesis results posed by duplicate reports [23]. All these information are shown in subsequent sections.

3.1 Basic Concepts

Based on the increased demand for better system performance and dependability, hardware or software components are threatened by faults which in turn deteriorate system reliability. Faults bring the normal execution state of a system into error state, which in turn results in system failure. Specifically, faults in software are more difficult to handle than in hardware because of their complexity and logical nature, which is hard to visualize, classify, detect, and correct [2]. In order to protect systems against these threats, reliability mechanisms have been designed with a variety of techniques to tolerate these faults while still delivering an acceptable level of service.

Fault tolerance techniques allow a system to continue to provide service in spite of hardware or software faults that remain in the system after its development [2], [9]. It is a technique aimed at increasing the dependability of a system by masking the software or hardware faults in the system. Particularly, software fault tolerance process has four main activities [2], which are:

I. Detection: identification of error
II. Diagnosis: assessment of the damage due to the error
III. Containment: preventing propagation of error
IV. Recovery: correction of the error state

The first three activities of the fault tolerance process deal with proper identification of the problem before going into the fixing stage. The recovery process, which is the fixing stage, is the core and most complex stage in fault tolerance process. There are two types of recovery; backward or forward recovery [2]. In backward recovery strategies, the system is rolled back to a previously saved state using the checkpointing scheme, while the forward recovery strategy is aimed at finding a new state to compensate the damage caused by the error based on some algorithm that is specific to the application [2]. Forward recovery uses redundancy scheme to come up with an alternative state and requires precise knowledge about the error.
The efficient design of fault tolerance mechanisms must balance between reliability, transparency, portability, low overhead, and scalability. Fault tolerance has been used in many application domains such as nuclear safety, military and nautical systems, electronic banking and commerce, aircraft and air traffic control, medical devices, space mission, automated manufacturing, etc [2].

3.1.1 Agent-centric Approach

The two general approaches of providing fault tolerance mechanisms in mobile agents systems are agent and system-centric [39]. In agent-centric approach, the design and implementation of the fault tolerance strategy is built into the user defined agents. This means the user defined agents and the application consists of code that is relevant for the fault tolerance. The developer in this case has to explicitly call the fault tolerance methods. Agent-centric approaches have a benefit of tailoring fault tolerance needs to the application domain and the fault tolerances strategy is controlled within the application. However, this came with the pitfall of increased complexity and overheads in size and computation, taking agent’s size and execution time into consideration respectively [39]. Furthermore, the developer has a burden of maintaining consistency of the processes and the data.

3.1.2 System-centric Approach

The system-centric approach, on the other hand, integrates the fault tolerances strategy into the agent platform, making it transparent for the user defined agents. The agent platform in this case may need to be modified to have the capabilities for fault tolerance. The system-centric approach has a benefit of abstracting the fault tolerance strategy, making the fault tolerances process transparent. The transparency elevates the developer’s burden in maintaining fault tolerances strategies into the application. The disadvantage of the system-centric approaches is that the strategies may require modification of the underlying platform, which makes maintaining compatibility very difficult especially during new platform version releases [16].

Figure 3.1: Backward and Forward Recovery
3.2 Fault Models

There are several types of faults or abnormal situations that can affect mobile agents or their environments [49]. Failures affecting mobile agent’s execution can be classified based on a fault classification scheme that is often used to categorize faults having the same features. These categories are used to model faults and devise methods for fault prevention and detection. However, several classifications of failures have been proposed [50].

Although the existing fault tolerant implementations refer to the fault classes using different terms, the basic idea behind the terms generally fall into one of the three generalized classes of failures: namely communication, crash and agent software failure. For instance, some implementations use the term node failure while others state it as host failure, where both refer to crash failure. Most of the existing mobile agent’s fault tolerance implementations are designed to either tolerate one of the stated failures or multiple of them in all situations. Based on the results obtained from the systematic review process, Table 3.1 shows the implementations and the failure type they tolerate. All three fault models are supported in [44], [27], [38], [30], [31], [32], [42], [36], [45]. However, very few of the implementations considered the problem of network partitioning. Only Guessoum et al. [44], Rothermel et al. [38] and Osman et al. [30] stated a form of solution to the network partitioning problem in their fault tolerance design.

Table 3.1: Failure Types

<table>
<thead>
<tr>
<th>References</th>
<th>Communication</th>
<th>Crash</th>
<th>Agent/Agent Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[26]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[5]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[27]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[28]</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>[29]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[30]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[31]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[32]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[33]</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>[34]</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[42]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[7]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[35]</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[36]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[38]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[39]</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>[40]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[41]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[14]</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[44]</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[46]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[47]</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>[43]</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Regardless of the permanency, intermittent or transient nature of the failure, we have generalized the abnormal situations of mobile agent’s execution as initially occurring in three distinct system boundaries. The three classes are communication, crash failure and agent software failure [51]. Although both communication and crash fault classes may eventually lead to agent failure, the classification is mainly based on the origin of the fault rather than the effect it produces in the end.

![Figure 3.2: Classes of Failures](image)

Below is a brief description of the categories of common failures associated with each class which might affects agents execution. These include:

### 3.2.1 Communication Failure

Communication failure is a class of failure that may affect agent’s execution in terms of movement and communication. Link failure directly affects the transport of agents and messages. However, faulty communication links may cause message failure due to out of sequence, duplication or message corruption [49], [32], [50].

### 3.2.2 Crash Failure

Crash failure is a class of failure that occurs when the place hosting the mobile agent fails, e.g. server failure. The cause of the crash can be hardware failure, operating system failure or agent platform failure resulting in baring the agent from communicating with other agents and the agent home server [51]. Crash failure eventually leads to agent system failures since either partial failure of agent places due to breaking down of components of agent places such as damaged communication units; or complete server crash due to system software failure, security attack or even deadlock situation disrupts the normal agent execution [51].
3.2.3 Agent/Agent Software Failure

Agent software failure class refers to failures that stop the execution of an agent due to unforeseen circumstance, defective software or bug existing within the mobile agent. The faults in software mainly cause out of order computation, invalid inputs, exception, denial of access to service because of heavy load and situation such as deadlock or live-lock characterized by locking [51]. Agent failures can occur in transit or in the middle of execution. Various fault tolerance approaches use transaction-based agent migration to tolerate agent failure in transit [26], [30], [36], [5], [38].

3.3 Fault Tolerance Schemes

As we stated earlier, the proposed fault tolerance schemes, which handle the sources of system failures and recovery, generally fall under replication-based, checkpoint-based or a hybrid of both schemes [16], [9], [49]. Moreover, some literatures identified additional preventive schemes such as software rejuvenation, return validation, secure coprocessor, third party rendezvous, enforced fault tolerance scheme and exception handling [42], [36]. Each scheme has its own strengths and weaknesses that call for their suitability/unsuitability in a given application area. These techniques can either be independently designed or incorporated into the available mobile agent’s platforms to provide fault tolerance for agent execution.

Our study shall focus on check-pointing and replication schemes since they provide better fault tolerance mechanisms and are widely used in this study area. The distribution of the publications by the two fault tolerance scheme in relation to execution mode is presented in Table 3.2 below. We can notice that the number of publications using either of the two schemes is fairly balanced at around 50% but with replication being used slightly more than checkpoint-based scheme.

Table 3.2: Publication by Fault Tolerance Scheme and Execution Mode

<table>
<thead>
<tr>
<th></th>
<th>Transactional</th>
<th>Non-transactional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Checkpoint</strong></td>
<td>[5], [38], [36], [27], [30], [26], [42]</td>
<td>[34], [45], [48], [46], [32]</td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>[29], [39], [7], [47], [35]</td>
<td>[43], [40], [41], [33], [44], [25], [28]</td>
</tr>
<tr>
<td><strong>Both</strong></td>
<td>[14], [31]</td>
<td></td>
</tr>
</tbody>
</table>
3.3.1 Check-pointing Scheme

Check-pointing scheme is a fault tolerance approach that incorporates two strategies; checkpoint and rollback recovery [30]. It focuses on saving a program’s or agent intermediate state to a stable storage at certain time intervals during failure-free execution, which can be used to recover the agent states to the latest checkpoint when failure occurs [8]. Each saved state is known as checkpoint and restoring the previous checkpoint upon recovery is called rollback recovery [30] (see figure 3.4). The checkpointed information helps the agent to resume computation from saved state (consistent state) without restarting from the beginning thereby avoiding domino effect (i.e. restarting from the initial state of the computation and losing all the work performed before a failure) [30].

Figure 3.3: Fault Tolerance Schemes Distribution

Figure 3.4: Checkpointing
Check-pointing scheme is assumed to have lower computation overhead when compared to replication schemes since agent’s replication and communication costs for synchronization are ruled out. However, check-pointing is not without penalty since excessive checkpointing could lead to performance degradation and slow recovery overhead [8]. Moreover, the scheme may block after a crash failure since the checkpoint may not be available until the host recover [45]. The checkpoint scheme is known to prevent the loss of agents but incurs the possibility of violating the non-blocking property of mobile agent’s execution [16], [8].

3.3.2 Replication Scheme

Replication-based scheme is one of the most widely used techniques to achieve redundancy [2]. It is a vital scheme used to achieve fault tolerance in distributed systems since it can ensure an agent’s service reliability, survivability and increased availability in the face of failure. In this scheme, groups of agents designated as replicas of leader agent exist in multiple places, who are candidates to act as the leader agent to continue computation in the events of its failure [9] (see figure 3.5).

Replication scheme tends to increase the system reliability because with $K$ replicas, it can tolerate up to $K-1$ failures if it is organized as simple primary-backup method. However, if the replicas are organized as majority consensus-based replication methods then $2K + 1$ replicas can tolerate up to $K$ failures (where $K = 0, 1, 2, \ldots$).

The replication scheme is known for preserving the non-blocking property of mobile agent’s fault tolerance execution property, but have the side effect of compromising the exactly-once property, which is considered undesirable especially for transaction based applications [16], [8]. In order to maintain the exactly-once execution property, the replicas have to agree on one execution site for each stage and the redundant execution of other sites ought to be undone by using either the distributed transaction, the consensus protocol or the leader election protocol to solve the agreement problem [16]. Hence, consensus and reliable communication forms the two main building blocks for replication scheme [7].

Replication scheme has a lot of optimization approaches to provide reliable and low performances over-headed. Among the optimization are synchronous, asynchronous, and lazy (enhanced with consensus) [40], [41].

![Figure 3.5: Agent Replication (adapted from S. Pleisch [12])](image-url)
The illustration in Figure 2.3, shows the replication scheme used in providing fault tolerance. We considered agent \( a_i \) as being executed by a group of places \( G_i = \{P_i^0, P_i^1, P_i^2, \ldots \} \) at stage \( S_i \). In stage \( S_i \), for instance, if place \( P_i^0 \) fails then the agent \( a_i \) is not lost because other places in \( G_i \) have also copies of \( a_i \) to start execution with and recreates the lost agent at the next stage [16].

There are two distinct design approach of replication for fault tolerance; namely passive and active replication. Replication scheme is perceived as being too expensive in terms of performances cost. This is due to the higher cost involved in maintaining replicas, which is characterised by consumption of huge execution time in the time required to replicate, migrate and to perform the consensus among replicas [40].

### 3.3.2.2 Active Replication

In active replication mode, there is a single leader designated as 'primary' or 'master' and one or more secondary replicas designated as 'backups' or 'slaves' [14], [16]. All of the replicas communicate with the primary or each other to synchronize and agree on results [14]. The main benefit of having active replication mode is low recovery time since all replicas are up-to-date with the execution state [33], [16], [14]. In the event of failure, any one of the active replicas can take over and resume the execution. The disadvantage in active replication is that it has high computation overhead due to simultaneous processing of messages and synchronization by all replicas [14]. When a primary failure is detected, one of the backup replicas has to be elected to take the place of the failed primary. Alternatively, the backup replicas can be made in ordered list to replace the failed primary but one possible disadvantage of the ordered list could be that there can be waiting delay until the next replica takes over. This is especially true if the migration of the agents is in asynchronous mode in which case other lower priority agents may have arrived but wait for the higher priority agent to take over [41].

### 3.3.2.1 Passive Replication

Like the active replication, the passive replication has also a single primary and many backups or replicas but the backup replicas do not process the assigned task or incoming messages concurrently. The primary executes the task operations and communicates/synchronizes with the rest of the replicas or backups by periodically updating them with the latest process information. In the event of failure of the primary, one of the backups is elevated to take over the primary’s responsibility based on some form of consensus such as voting [16]. Passive replication approach has the benefit of lower overhead since only one replica actively processes the assigned task while the other replicas follow inactively. However, the recovery process is slow due to the delay as a result of restoring the recovery information from the stored check-pointed state of the failed agent [33], [14]. Although the synchronization process in passive replication has lower overhead, maintaining the check-pointing process introduces additional computation cost.

A hybrid of the two replication types has been proposed in the existing fault tolerance implementations as semi-active replication type [33], [7], [39]. In semi-active replication, the backup replicas are grouped and assigned a single proxy that is responsible for communication with the primary on behalf of the replica agents. The replica agents in semi-active replication are passive in nature. When a failure is detected, one of the replicas will be chosen to take up the place of the primary agent.
3.4 Fault Tolerance Protocols

Protocols form a vital part of mobile agents fault tolerance implementation model. They are basically integrated into the models that achieve the two execution properties of fault tolerance mobile agent’s: exactly-once and non-blocking [11]. The protocols in the fault tolerance implementations are mostly achieved by effective message passing (i.e. synchronous or asynchronous message) in order to coordinate local activities and ensure reliable agent failure detection and recovery. However, one issue about categorizing these protocols is that it is almost difficult to put a classification on them since some of them exist in combinations with other protocols [11]. In this section, the fault tolerance protocols are organized with respect to the two execution properties, namely the exactly-once or non-blocking properties, but this does not necessarily mean the protocol can only be used to address one of the properties at a time. The available fault tolerance protocols are:

a.) Exactly-once Protocols
   1. Consensus
   2. Transaction-based

b.) Non-Blocking Protocols
   1. Rollback Recovery
      i. Checkpoint-based
      ii. Message Log-based
   2. Replication

3.4.1 Exactly-once Protocols

The protocols for addressing the exactly-once property aim to ensure that an agent must execute the desired action not more and not less than once in a host. Replication based approaches have the potential to violate the exactly-once property since multiple clone agents can be executing the same task in parallel. The consensus protocol and the transaction based execution protocol aim at avoiding duplicate execution of agents in case of a primary agent failure.

3.4.1.1 Consensus Protocol

The consensus protocol is a process of achieving agreement between the primary agent process and the replicas processes. The consensus is usually done by making a vote on a certain outcome of computation that is conducted by all the participants. In the consensus approach, the agreement step is usually initiated by the primary agent through the $CON_{begin}$ message. The rest of the consensus process is through periodic status notification messages sent to the observers or replicas using a timeout [40]. The simplest consensus by voting is concluded when all the participants agree on one outcome. The majority voting protocol is an election protocol that is concluded when the majority of the participants agree on a certain outcome. In this case, the majority of the replicas agree on the result of an election either to choose a new leader or on a concurrent execution result. In both cases, the result that is supported by the majority will be chosen as an elected result.
**Figure 3.6: Consensus Process**

The messaging process is as follows:

1. Primary/worker sends $\text{CON}_\text{begin}$ message to every replica/observer.
2. Every replica receives and sends back replies with $\text{CON}_\text{ack}$ message. If it already sent, no reply is needed.
3. Primary/worker then reply back with $\text{CON}_\text{confirm}$ which terminates the consensus step when majority of $\text{CON}_\text{ack}$ messages is received. If no majority $\text{CON}_\text{ack}$ messages are received, the primary undoes the executed task. However, the voting protocol is used and the next stage is executed only when majority votes are received. Nevertheless, only one primary can obtain the majority votes for a stage task completion [40], [16].

A common application of the consensus protocol when a failure is detected by replica agents is through timeout. Accordingly, when a message timeout expires during waiting for acknowledgment by the replica, it is then assumed that the primary has failed and immediately the backup replicas initiate the consensus for electing a new primary.

A replica can become the new primary through its own consensus for either being the first to detect or having the highest priority when the primary fails [40]. The existing mobile agent’s fault tolerance implementation that relies on this protocol is stated. Both Park et al. [40] and Park et al. [41] uses asynchronous consensus where the execution and consensus of an agent may proceed asynchronously with the agent replication. That is, agent begins its execution without waiting for the other replicas, or ends the stage before its replicas are ready for the consensus.

Optimization to this approach is introduced by applying selective agent replication and voting aimed at minimizing overheads in the system [14]. Another improvement is by using replication group proxies of agent that is used to reduce message processing overhead since the primary sends message to the replication group proxies rather than to individual replicas [39].
3.4.1.2 Transaction-Based Protocol

Similar to the consensus protocol, the transaction commit is used to reduce the effect of failures on the availability of operational sites. In distributed transaction, there are several operations running at multiple sites and transaction is carried out by a set of processes known as resource managers (RMs) running at each site [16]. Such a transaction is terminated when a commit/abort request is issued by one of the RMs. Commit and abort are both irreversible and unlike consensus, transaction can only be committed if all participating sites vote or agree to commit, otherwise, the transaction is aborted [16].

The goal of the transaction-based approach is the attainment of global atomicity [16]. In the transaction-based studies we considered, some of them have demonstrated the use of either 2-phased commit protocol by Rothermel et al. [38] or 3-phased commit protocol by Silva et al. [29]. The 2-phase and 3-phase commit protocols are effective in ensuring exactly-once property but they are computationally expensive processes [29]. Most of the transaction-based mobile agent fault tolerance approaches use the basic transaction-commit protocol. For instance, Leung et al. [26] uses the centralized transaction commit, while Lyu et al. [27], Osman et al. [30], Tanaka et al. [35], Silva et al. [36], Pleisch et al. [7], Kaneda et al. [47] use the transaction commit protocol.

Some of the implementations demonstrated a combination of both consensus and transaction-based protocol. Silva et al. [29] use a majority vote election protocol (Consensus) with replicating recovery information and 3-phase transaction commit. Both Kaneda et al. [47] and Pleisch et al. [7] use combined transaction commit and consensus while Rothermel et al. [38] uses consensus voting and 2-phase transaction commit.

3.4.2 Non-Blocking Protocols

Protocols used for overcoming blocking problems can be classified into two. One form of avoiding the blocking aspect of an execution is to restart the failed process again. Another alternative is to use duplicate agents that are able to take over in time of primary agent failure. The protocols under rollback recovery protocols relate to the restarting aspect of the protocol while the replication-based protocols relate to the duplication of agents. The two approaches are described in detail in the following sections.

3.4.2.1 Rollback Recovery Protocol

Rollback recovery is a technique that requires a process to periodically record its consistent state, known as checkpointing, in a stable storage. The idea behind this approach is to reduce the amount of loss in computation by avoiding the restarting of computation from the beginning. Most of the existing rollback recovery approaches are based on message passing aimed at proper coordination of activities of local process. Two sub-protocols are found in this approach, checkpoint-based and message logging-based [53], [7].

3.4.2.1.1 Checkpoint-based

The checkpoint-based approach relies strictly on regularly saving of checkpoints (i.e. agent’s process states and code) to a stable storage for future restoration or recovery in the event of failure [53]. Protocols that fall under the checkpoint-based scheme consist of the coordinated, uncoordinated, communication-induced, lazy and the timer-based protocol [53], [7]. Most of the rollback-based protocols in this study use checkpoint-based protocol to provide fault tolerance behaviour. Osman et al. [30] use the communication-induced checkpointing while Hamidi et al. [42] use independent checkpointing enhanced with
receiver-based logging. Also Taesoon et al. [46] the checkpoint-based scheme to restore agents processes back to its consistent state in the event of failure and lastly, Tosic et al. [45] uses checkpoint-based with reliable publisher/subscriber messaging layer. The messaging layer is a platform independent layer for ensuring reliable delivery of messages and failure detection.

### 3.4.2.1.2 Message Log-based

The log-based approach involves logging of non-deterministic actions preset as determinants in combination with check-pointing to achieve fault tolerance recovery behaviour. Examples of protocols under the log-based approach are the pessimistic and optimistic logging protocol [7]. Pessimistic execution ensures that changes are applied only if no agent crashes and there is no erroneous result. In optimistic execution the place modifications can be immediate and transparent to other agents but undoing modifications is a complex task [7]. A good application of the rollback recovery by message logging approaches is by logging messages along with the process state (checkpoints) for future use in the recovery of agents. The existing fault tolerance mobile agent’s approaches that use the message log-based recovery mechanism are discussed by Chen et al. [32], while Lyu et al. [27] achieve recovering transactional agents with message logging.

Checkpointing and logging approaches offer a considerable benefit to application programmers in preserving execution states. However, they come with a penalty of overheads in terms of computation and storage. The approach requires a careful choice of checkpoint frequency and efficient garbage collector so as not to adversely affect the normal execution of the application [30].

### 3.4.2.2 Replication-based Protocols

Replication, as discussed in section 3.3.2, is a way to overcome blocking by using duplicates. Checkpoint-based approaches are usually prone to blocking since they rely on a single primary and the backup is created after a failure of the primary is detected. In replication-based approaches on the other hand, there is a live backup agent, either as a duplicate of the primary or as a standby, even before failure is detected.

Although replication is suitable to avoid blocking, it requires synchronization of replicas with the primary, which is very expensive both in computation and communication [14]. The synchronization is usually achieved through the exchange of multiple messages that introduce computation and communication overhead.

Various optimizations have been suggested in the studies we considered here. Fedoruk et al. [39] formed a group of replicas and a proxy that communicates with the primary on behalf of the multiple replicas. Olivier et al. [33] introduced a dynamic adaptive replication scheme that is similar to other replication based approaches but the number of replicas is determined dynamically during execution. The sliding window protocol discussed by Summiya et al. [25] is a technique that controls the number of backup or replicated agents in order to minimize bandwidth consumption.

### 3.5 Implementation Design Models

Making a mobile agent fault-tolerant is an important issue stemming from the root of reliability. Fault tolerance guarantee consistent agent’s execution even in the face of system failure, i.e. errors can be detected and recovered. The recent development in mobile agent’s fault tolerance has shown that a number of design approaches used in implementing a
reliable fault tolerance for mobile agents exist. Some of the models/approaches either achieve fault tolerance combining two or more protocols. These approaches are either incorporated into the underlying program code or the agent’s platforms to make it more effective and achieve the desired goals. However, most of the current state-of-the-art fault tolerance approaches are hybrid of a variety of the protocols but some also include preventive techniques such as software rejuvenation. The design approaches are organized as follows:

1. Primary-Backup Approach
   a. Traditional Primary-Backup/ Task replication/Data redundancy
   b. Rear Guard with Backup, Pipe-line Mode

2. Primary-Witness Approach
   a. Rear Guard with Witness
   b. Collaborative Agents

3.5.1 Primary-Backup Model

Primary-Backup Model is a well used fault tolerance design that is based on the replication of the system components (agents or server) resulting in two entities namely the primary component (worker component) and one or more observer components (backups/replicas component) [6], [40]. The observers in this case are usually clones of the primary agent that have the capability to resume the execution of tasks if the primary fails for some reason. The primary component takes charge of execution while the backup components monitor the primary’s computation for any possible failure.

3.5.1.1 Traditional Primary-Backup/Task replication/Data redundancy

The traditional primary-backup model relies on the replication of system components, task, data etc in order to achieve fault tolerance in mobile agents. For example, multiple servers are used to implement the fault-tolerant service involving client-server [7], [28]. In this approach, the entire state information of the service and data becomes fully replicated at each server. One of the resulting servers is installed as the primary while the others become backups. Following the primary-backup principle, in the event of the primary’s failure, one of the backups takes over as the primary as well as notifying the clients so that later requests will be directed to new primary. One approach in the existing approaches is by Mellouli et al. [28] where task and agent is replicated using a handshake messaging protocol to achieve overall fault tolerance behaviour. The scalability of the traditional primary/backup is very poor since the replication overhead will be very high for large volume of data, task and agent.

3.5.1.2 Rear Guard Agents as Backup

The Rear Guard approach is mainly based on the primary-backup principle but the backup agent resides on the previously visited host. The backup is responsible to monitor the primary agent and perform recovery actions to resume the computation when it detects failure of the primary agent [43]. With this approach the network is stage-based partition.

The placement of the primary and backup agents in different hosts mainly stems from the necessity to withstand crash failures that could have resulted in the loss of both the primary and backup agent had they both been running on the same host. Reliable failure detection is a challenge in this approach since it is difficult to distinguish crash and network partitioning of the primary agent host [43]. However, several broadcasting algorithms that aim at multicasting agent code and computation results as agent completes its host task have been
proposed. For instance, the broadcasting technique operates by combining ping or heartbeat messages and timeouts to detect proper host failures [43].

Various forms of rear guards with backup approaches are discussed in existing fault tolerance implementations. For instance, in [5], multiple rear-guard enhanced with decision scheme designed reduce communication overheads and prevents blocking as well as ensuring the exactly-once execution property in the mobile agent execution using actual, rear guards and witness agents, while Yang et al. [44] uses parallel processing of replicas using the Reverse Mobile Agent Algorithm (RMAA) that uses forward and backward moving agents. Other rear guards approaches are based on combination of other protocols. An example of such optimizations is by Johansen et al. [43] that is based on rear-guard protocol and reliable broadcast protocol with election protocol. Chaining of rear guard agents throughout the execution path is also an extension of this approach. The pipe-line mode discussed by Pleisch et al. [7] leaves a rear guard in previously visited hosts and if the latest rear guard fails to recover then the next one will try recursively.

### 3.5.2 Primary-Witness Model

Primary-Witness Model is very similar to the above primary-backup approach but the backup agent in this case is not a duplicate of the primary agent rather it is a different type of agent usually for monitoring and creating a backup agent when the primary agent fails. The main difference here is that the backup agent that replaces the failed primary agent is created only after the detection of a failure.

#### 3.5.2.1 Rear Guard Agents as Witness

The rear guard as the witness approach is also very similar to the rear guard as backup approach discussed in section 3.5.1.2. Again the difference here is that the witness agent is a different type of agent that can not on its own take over as a primary during recovery. The primary agent is in charge of the natural execution while the witness is responsible for monitoring and recovering, which in this case involves creating a new primary and restoring normal execution.

Rear guards as witness approaches are discussed in existing fault tolerance implementations. For instance, in Meng et al. [31] when the Monitor Agent (MA) detects failure of Execution Agent (EA), it creates Repair Agent (RA) for fixing the error in the EA host. Another similar implementation is by Lyu et al. [27] and Lyu et al. [34] that uses actual and witness agents but creates a probe agent for recovering log during recovery.

#### 3.5.2.2 Collaborative Agents Model

Collaborative Agents Model is an extension of the Rear Guard as Witness technique discussed in section 3.5.2.1 above, but uses three or more types of agents working together for achieving fault tolerance. The different types of agents have designated responsibility in the detection and recovery processes. In this technique, the different types of agents work collaboratively to provide an overall fault tolerant action with a clear division of labour. The primary agent again is in charge of the execution. The other types of agents participate for specifically task such as monitoring, tracking, checking path, recovering, creating another type of agent, etc, however, they can not take the place of the primary agent during recovery.

Various forms of collaborative agent implementations are discussed in existing fault tolerance approaches. For instance, in Summiya et al. [25], three types of agents, namely
observer agents, ping agents and transaction agent, are used for monitoring, path checking and executing transactions respectively. Another similar implementation is by Leung et al. [26] that uses three types of agents, namely worker, monitor and tracker.

3.6 Addressing the Research Questions

3.6.1 Systematic Review Question 1

What is the state-of-the-art in research in the recognised mobile agent’s fault tolerance?

What we can understand from this systematic review is that the mobile agent fault tolerance field of study is actively being studied in today’s state of research. The years between 2004 and 2006 has shown a remarkable increase in number of publications in this area, though the trend seems to be declining in the last three years. Over the years, the number of studies that are supported by simulation or experiment has improved.

The main attribute of any state-of-the-art fault tolerance approach is the ability to observe the agent and detect failure. All of the existing approaches consist of a mechanism to detect failures that is based on the results of the observation module that corresponds to the fault model. Fault models define which set of observations are categorised are failure and which are acceptable operation modes.

The existing fault tolerant mobile agent implementations have tried to address at least one of the three fault models (see Figure 3.7). Fault models for communication failures were least address throughout the studies. Most of the studies that do not address communication failures assume that the network is reliable or eventually resumes service even if the network disconnects at times. Crash and agent software fault models seem to be supported in most of the existing implementations. However, only 35% of the existing implementations support all three fault models. Only 12% of the existing implementations stated that they have considered network partitioning to be a problem and have suggested a form of solution to it. See challenges in chapter 4 for more on network partitioning.
Another equally important attribute of any state-of-the-art fault tolerance approach is the ability to recover from failure. The state of recovery may not be absolute as resuming execution at exactly the point of failure but generally falls under forward or backward recovery. In backward recovery some processes may be executed twice or in the case of forward recovery some processes may skip certain segment of the complete execution. In order to have an acceptable recovery state, the recovery process requires careful planning. The planning is mainly to bring about a compromise in recovery delay with respect to the specific application domain at hand.

As can be seen from the evidence about the protocols in section 3.4, the two types of schemes, checkpointing and replication, are commonly used in today’s mobile agent fault tolerance implementations. Both checkpointing and replication-based schemes are used almost equally over the years of consideration (see Figure 3.8). However, the trend of distancing away from active replication-based schemes is quite visible. The majority of the replication-based primary studies tend to favour inactive replication due to the lower computation overhead.
The majority (72%) of replication-based approaches follow either passive or semi-active replication. The semi-active to passive ratio within the passive replication-based category is 3:7 indicating semi-active techniques are not used that often either. The synchronization of agents in active replication incurs a high computation and communication cost, hence most implementation do not favour active replication (see Figure 3.9). The computation overhead in active replication is highlighted even among the implementations that support both replication types.
**Figure 3.9: Replication Types Distribution**

The distribution of transactional and non-transactional executions is somehow balanced (see Figure 3.10). Transaction-based fault tolerance executions are slightly higher than non-transaction-based modes in the existing implementations. We have not noticed any shift in the trend over the years so both methods are used over the years. However, transactional executions are more reliable in maintaining the exactly-once property of agent execution. Non-transaction fault tolerance executions can also achieve some level of consistency with lower computation overhead. However, it is recommended to follow transactional execution for application domains that require higher level of consistency. A possible explanation for the balance in execution mode usage indicates that it is all about trade off between reliability and performance overhead. Transactional mode execution is reliable but has computation overhead than non-transactional.

![Execution Modes](image)

**Figure 3.10: Execution Modes**

The mode of communication in the existing fault tolerant mobile agent implementation is mostly asynchronous (see Figure 3.11). This is mainly due to the characteristics of a mobile agent environment where agents are autonomous and migrate usually in open networks with latency. Furthermore, synchronous communication has high performance overhead when compared to asynchronous communication. There is no implementation that solely works on synchronous communication but the approaches by Park [14] and Olivier et al. [33] have the capability to operate both asynchronously and synchronously. The obvious shift to asynchronous mode of communications demonstrates that asynchronous communication is the most efficient mode in agent communication.
3.6.2 Systematic Review Question 2

What available fault models are considered in designing fault tolerance protocols?

The available fault models in the existing fault tolerant mobile agent implementations have been discussed in detail in sections 3.2.

3.6.3 Systematic Review Question 3

What are the available approaches and their design elements in the available recognised mobile agent’s fault tolerance in current state of research?

The available protocols and approaches in the existing fault tolerant mobile agent implementations have been discussed in detail in sections 3.3, 3.4 and section 3.5.

The remaining systematic review questions will be answered in the next chapters.
CHAPTER 4

FAULT TOLERANCE INFLUENCING FACTORS

4.1 Performance Factors

This section presents factors that may affect the performance of fault tolerance implementations. Some of the selected primary studies have experimental data showing some of the factors that affect the performance of their individual implementations. The focus here is on the factors rather than the actual figures from the experiments of the individual implementations. The experimental result figures from the individual experiments have unique scenarios and measurements that are specific to each case; hence it cannot be quantitatively compared across the studies.

4.1.1 Agent Size

The size of the agent has been seen as having a considerable impact in the performance of fault tolerant implementations. The agent size is affected by the size of the code and the payload data it takes with itself during mobility [15]. It has been demonstrated in numerous experiments from literatures that the execution time of the agents linearly increases with the increase in size of the agent [14], [40], [41], [42], [36], [46], [7]. The increase in agent size also introduces an overhead to the agent replication process [7], [40]. More than 50% of the experimental studies reported agent size as a performance factor.

4.1.2 Number of Replica

More than 50% of the replication based schemes reported that the number of replica has an implication on performance. The adverse effect of increasing replica agents is exaggerated mostly in the synchronous replication schemes [40]. In designs that consist of a consensus process, the increase in number of replica agents will result in longer time being lapsed during the consensus stage [41]. Some of the existing fault tolerance implementations have reported number of replica/witness factor in their findings [40], [41], [36], [46], [39], [28], [34].

In replication-based fault tolerance schemes, the number of replica or witness agents has a direct impact on the performance of the scheme. An increase in the number of replica agents or witnesses adversely affects the performance of the replication process; hence replication based schemes in general [36]. The performance effect of number of replica is attributed to the computational overhead associated with the synchronization of clones. When the number of the clones increases, the number of synchronization messages and the computation process needed to send and receive the synchronizing messages increases.

4.1.3 Message Size

More than 30% of the experimental primary studies consider message size as a factor influencing performance. Messages are basically used for communication and
synchronization between mobile agent processes. The size of the message that is exchanged between mobile agent processes has a significant impact on the performance of the fault tolerant scheme, which in turn contributes significantly to network traffic. This adverse impact emanates from the fact that the time spent for sending a particular message increases as the size of the messages increase. The cost of message size on performance is relatively higher in synchronous communication models than in asynchronous model [54]. Some of the existing fault tolerance implementations have reported the message size factor in their findings [14], [38], [45], [34].

4.1.4 Number of Messages

Similar to message size, the number of messages also affects the performance of fault tolerance schemes especially as uncontrolled number of messages can overwhelm fault detectors. Generally replication based schemes have communication overhead due to increased communication cost in agent forwarding and consensus [16], [39]. Naturally the response time of a distributed mobile agent systems improves when more agents collaborate and work to achieve a certain task. However, the number of messages exchanged increases as the number of agents in the system increases, which creates an additional overhead on the communication and migration process of the system [39]. It has been demonstrated that reducing number of messages produces performance gain [7]. Some of the existing fault tolerance implementations have reported the number of messages factor in their findings [44], [39], [31], [34].

4.1.5 Number of Hops/Hosts

Mobile agents typically migrate from host to host to perform a certain task. Hopping can be characterized as the migration process of an agent that is going from a current execution host to the next host in its task execution path. A mobile agent’s survivability decreases as number of servers in the agent’s itinerary increases [34]. The risk can be attributed to the increased possibility of reaching a timeout when the execution time takes a long time. Correspondingly if the scheme relies on timeout then it may initiate an unnecessary recovery process that would affect the performance of the system negatively. In addition to timeouts, reliable agent migration is a complex process especially if it involves maintaining backup agents on previously visited hosts. Some of the existing fault tolerance implementations have reported the number of host factor in their findings [7], [26], [27], [44].

4.1.6 Frequency of Checkpointing

Another important factor that affects the performance of fault tolerance mobile agents is the degree at which checkpointing are taken. Although checkpointing based agent execution does not guarantee loss of an agent, it is open to blocking [6], [5]. This happens when an agent server crashes and the checkpoint is not available for a long time period until the agent server is recovered. Although taking checkpoints periodically helps guard against loss of computation results during failure by restarting from the latest checkpoint, the frequency at which it is executed brings about additional overheads. An increase in the frequency of checkpointing increases overhead while infrequent checkpointing brings about much re-computation in the event of recovery [6], [5].
4.2 Platform Support

4.2.1 Fault Tolerance Components

The agent platforms provide executing environment for mobile agents. Fault tolerance mechanisms embedded in the mobile agent platform is assumed to have better dependability than custom made fault tolerance mechanisms that sit on top of the platform. The world of mobile agent’s fault tolerance is currently inundated with varieties of agent’s platforms overwhelmed by the popularity of Java programming language [5]. However, only few of these platforms provide partial or incomplete facilities for fault tolerance mechanisms [8], [5]. For an agent platform to completely provide support for fault tolerance execution, certain key functionalities such as process state capture/restore (mobility and checkpointing), reliable inter-agent communication (synchronous and asynchronous), and agent replication/cloning etc. irrespective of the language supported are of considerable importance. These functionalities are discussed below.

4.2.1.1 Process State Capturing/Restoration

Mobility, a vital characteristic of mobile agents, is achieved through an evolution of state capture/restore that allows the execution state of an agent to be captured, transferred and then restores at the destination host (checkpointing). The mobility of agents either strong or weak refers to the migration of agent execution but each differs in the level of process information that is captured and restored. In strong mobility, the full process state (execution state, code and data) along with the execution thread is captured, moved and restored at the destination host. While in the case of weak mobility, the code is migrated and linked to a new execution thread at the destination host [55]. Systems that support strong mobility offer the possibility of capturing and restoring of a process at any point in its execution time but the transparency does place a high system overhead [56]. However, not all mobile agent platforms provide support for the two classes of process state mobility. Strong mobility offers better flexibility, however most agent platforms support only weak mobility due to its complexity and in the inadequate support to deal with execution context migration by the present Java Virtual Machine (JVM) based platforms [55], [36].

4.2.1.2 Reliable Communication

Reliable communication is a sin qua non to effective fault detection and recovery especially among agents or collaborating agent. It is one of the bases for agent’s platform to realize reliable fault tolerances. With reliable communication, it is possible to track the mobile agent or to route messages to its mailbox during migration either using synchronous or asynchronous mechanism. Although synchronous interaction plays a vital role in effective fault detection especially in the client-server paradigm, the distributed nature of mobile agents system along with unreliable link such as the Internet has shifted the communication protocols of the agent platforms to provide support for asynchronous interaction [54]. Currently all agents platforms support some form of communication, but the platforms that support asynchronous communication provide better communication performance over synchronous communication [54].

4.2.1.3 Replication and Cloning

Replication/cloning form an important platform support feature for agent fault tolerance. Replication is process of copying agent, server or system component information from one location to another and maintaining its consistency to provide overall fault tolerance. Cloning on the other hand is a form or replication applicable to mobile agents where new
agents are created from the current state of the original agent by copying the private address stack of the original agent to the new agent [57]. This feature allows multiple agents to exist with the same execution state and to continue its execution in the event of some form of system failure.

4.2.2 Available Agent Platforms

Several academic and commercial systems present agent-based computing platforms out of which to mention some are the Agent TCL, ARA, Mole, Tacoma, Voyager, Aglets, Concordia, Voyager, NOMAD, JAMES, JADE, Naptel, MadKit, FIPA-OS, Grasshopper etc [2], [5], [53]. These platforms differ in their features, architecture and implementations, but they more or less offer common facilities for the support of agent migration, inter-agent communication, various forms of security and programming or interpreted languages etc [15]. However, these differences have created an adverse effect on the interoperability and the speedy proliferation of agent technology.

Based on the findings of this study, we present a qualitative comparison of the current agent platforms that are used in recognised mobile agent’s fault tolerance implementations. Table 4.1 below shows the various platforms used in the primary studies reviewed in the systematic review and their nature of fault tolerance support.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Prog. Lang.</th>
<th>Mobility</th>
<th>Communication</th>
<th>Fault Tolerance Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aglet [58]</td>
<td>Java</td>
<td>Weak</td>
<td>Asynchronous, Synchronous, Proxy</td>
<td>NA</td>
</tr>
<tr>
<td>Concordia [59]</td>
<td>Java</td>
<td>Weak</td>
<td>Asynchronous</td>
<td>Yes (Checkpoint, Transactional message queue, Proxy)</td>
</tr>
<tr>
<td>FIPA-OS [60]</td>
<td>Java</td>
<td>Weak</td>
<td>Asynchronous</td>
<td>Yes (Replication(clone) Transactional message queue, Proxy)</td>
</tr>
<tr>
<td>Grasshopper [61]</td>
<td>Java</td>
<td>Weak</td>
<td>Synchronous, Asynchronous, Multicast, Dynamic method invocation</td>
<td>NA</td>
</tr>
<tr>
<td>JADE [62]</td>
<td>Java</td>
<td>Weak</td>
<td>Asynchronous</td>
<td>NA</td>
</tr>
<tr>
<td>JAMES [21]</td>
<td>Java</td>
<td>Weak</td>
<td>JavaSpace</td>
<td>Yes (Checkpoint)</td>
</tr>
<tr>
<td>MadKit [63], [64]</td>
<td>Java</td>
<td>Weak</td>
<td>asynchronous message passing</td>
<td>Yes (congestion management, agent monitoring mechanisms)</td>
</tr>
<tr>
<td>MOLE [65]</td>
<td>Java</td>
<td>Weak</td>
<td>Asynchronous, Synchronous, Sessions</td>
<td>Yes (Transactional message queue)</td>
</tr>
<tr>
<td>Naptel [66]</td>
<td>Java</td>
<td>Weak</td>
<td>asynchronous message passing</td>
<td>Yes (agent monitoring mechanisms, cloning)</td>
</tr>
<tr>
<td>Tacoma [68]</td>
<td>C/C++, ML, Perl, python</td>
<td>Weak</td>
<td>Asynchronous, Synchronous</td>
<td>Yes (Rear guards)</td>
</tr>
<tr>
<td>Voyager [69]</td>
<td>Java, C#, C++</td>
<td>Weak</td>
<td>Asynchronous, Synchronous, Multicast, Proxy</td>
<td>NA</td>
</tr>
</tbody>
</table>

Note: NA (Not Applicable) implies that either the information or the feature is unavailable.
4.3 Challenges

Given the autonomous and distributed nature of mobile agents, it has been proved that there are lots of challenges that affect fault tolerance implementations. The challenges limit the total efficiency or direct application of fault tolerance strategies in building especially, mobile applications [6]. For instance, the complexity of fault tolerance strategies is cumbersome to tackle by the agent platforms in their core releases partly on grounds that the technology has not matured and partly on reasons of diversity in reliability requirements of potential agent applications [53]. Below we discuss the various challenges that confront the available implementations and the complete realization of reliable fault tolerance execution.

4.3.1 Reliable Fault Detection

Mobile agents are difficult to track centrally once they are injected into the distributed system. The reliability of fault tolerance strategies is highly dependent on the ability to correctly detect failures [25]. One major challenge in fault detection comes when a fault tolerance process wrongly detects fault and acts upon it. In such cases, an agent is wrongly assumed failed and a replacement agent is created in place of the failed agent. This leads to the duplication of execution, which in turn leads to the violation of the highly desirable “exactly once” property of agent execution [25].

The existing fault tolerance schemes, seen in this study, rely on techniques such as timeout, periodical exchange of heartbeat message, call backs, etc for reliable detection of faults [36], [44]. However, none of these techniques is absolute in detecting the occurrence of failures [5]. A known challenge in reliable fault detection between different hosts is especially sited when trying to distinguish between remote host failure and communication link failure [43]. Although only around 20% of the primary studies specifically mention the problem of reliable fault detection, most of the studies state it as very critical attribute. The criticality of fault detection process necessitates for a fault tolerant framework implementation to have a dedicated component to reliably detect faults.

4.3.2 Network Partitioning

Another important challenge in some of the fault tolerant mobile agent implementation is network partitioning, which results from communication failure [49]. Some fault tolerant mobile agent implementations execute in stages in which the internal network is partitioned into stages or domains [7], [35]. Each stage consists of sets of nodes capable of independent failure. Within a given partition, a specific node is designated as worker while others are the observer agents. However, in the event of communication failure, the various stages will be unable to communicate either to advance to the next stage or complete the assigned task, which results in blocking [43].

In the case of same stage partitioning, both the worker and observer are in the same stage but two distinct partitions are formed. The worker agents continue to work independently without the knowledge of what is going on in the other network partition. But if the observer agent happens to be in the other partition and takes recovery action while the workers in the partitioned network continue to operate, then this will result in violating the exactly-once property when the network resumes service.
The effect of network partitioning may not be critical for application domains that employ forward recovery. However, the monitoring component in a fault tolerant framework should have a mechanism to take care of runaway agents that potentially are alive when the partition resumes service. Only around 12% of the primary studies specified network partition as a challenge while most regard it as a temporary problem.

![Network Partitioning Diagram](image)

**Figure 4.1:** Network Partitioning

### 4.3.3 Lack of Platform Support

Agent platforms can play a vital role in achieving fault tolerance. Much of the complexity in implementing fault tolerant mobile agents can be been wrapped in the platforms. In this section, we will consider some of the essential features that can be put in the platforms so as to assist the fault tolerance implementations. The challenges coming from the agent platforms will be too complex to be addressed in the fault tolerant framework.

#### 4.3.3.1 Lack of Full Process State Capture/Restore

Most of the agent platforms today are Java-based systems and they use the Java Virtual Machine (JVM) underneath the platforms. The Java-based platforms, around 90% of the agent platforms, do not allow the capturing and restoring of the full execution state of a process. As a result, achieving strong mobility of agents is impossible in Java-based agent systems. An additional challenge in relation to the strong mobility is the complexity and difficulty of capturing the full state of collaborating processes. The context of execution, in the case of collaborating agents, involves multiple processes, each with their own process context that extends to other processes recursively. Most of the agent platforms have no way of mapping dependencies and contexts of execution. The difficulty to recover from failures in distributed collaborative agents’ environment also stands when global checkpoint states need to be maintained. The highly distributed nature of mobile agent systems makes it complex to maintain global checkpoints between processes. The available approaches made no clear consideration about how a single agent’s failure can affect the consistency of the global state of collaborative agent applications.
4.3.3.2 Lack of Interoperability

Interoperability of agents is another important challenging issue in the execution of agents in multiple platforms. The lack of interoperability originated due to the increase in incompatible Application Programming Interfaces (APIs). As a result, fault tolerant agents developed for one agent platform cannot be ported easily to a different agent platform [67]. This lack of interoperability affects the adoption and realization of fault tolerance from existing implantations. Although standards such as Foundation for Intelligent Physical Agents (FIPA) and Mobile Agent System Interoperability Facility (MASIF), have taken steps to end this dilemma, it is yet to be realized in the future [2]. FIPA standard aims to bring interoperability between agents and agent-based systems so as to allow reuse and ease of upgrade but it lacks to consider fault tolerance in the interoperability. On the other hand, MASIF, the first mobile agent standard by Object Management Group (OMG), aims to bring interoperability between agent platforms by integrating Remote Procedure Calls (RPC) paradigms in mobile agent platforms [57]. Most of the RPC standardization efforts use interface description language (IDL) definitions to allow interoperability.

4.3.4 Full Transaction Support

Transaction support is vital part in realising the exactly-once property of agent execution that has a distributed nature. Distributed transaction involves many operations running at several sites, where commit/abort request is used to terminate transaction by sites. Moreover, transaction can only be committed only if all sites are willing to commit it. This operation requires complete atomicity to ensure exactly-once execution. However achieving this atomicity property in various types of failure situations pose a tough challenging situation that is worsened by the extremely dynamic nature of mobile agent systems which is somewhat incompatible with the concept of transactions that forces several agents to remain in the same transaction as long as it does not commit or abort [6], [70], [29]. Other effective transaction protocols such as the 2-phase and 3-phase commit protocols are too expensive to maintain [29]. The complexity and computational overhead of managing transaction will have an implication on the performance of any fault tolerant framework. Therefore, transaction support in a fault tolerant framework can be incorporated either as an external component or a platform service that provides transaction service during agent execution.

4.3.5 Scalability

In the investigation of the fault tolerant mobile agent, all of the implementations we came across are in academia. The existing implementations mostly conduct experiments and simulations on a small scale setup, although some of the implementations promise scalability [26], [33], [41], [44]. The unavailability of large scale experiments in existing implementations curtails the dependability of the promised scalability. Considering the currently existing implementations, it will be very difficult to say anything about the reliability and effective of the approaches in real world applications. However, a fault tolerant framework should consider the scalability of the solution when number of agents grows. In the majority of real case scenarios, the number of agents progressively increases with the complexity of the task that is supposed to be executed via mobile agents.

4.3.6 Lack of Transparency

Transparency of a fault tolerance strategy increases flexibility to the developer. The transparency hides the details of a process into the underlying component of a system, usually the platform. The abstraction of fault tolerance protocols into the underlying platform
permits introduction of fault tolerance mechanism without making modification to the rest of the application code [25]. Achieving transparent fault tolerance protocol in agent platforms has been a great challenge since it requires the modification of the underlying platform to accommodate the protocols. Most of the fault tolerance platforms seen in our study are not transparent to the developer. Thus, developers are required to come up with fault tolerance protocols in the applications, which make the application code complex and hard to maintain. Transparency can not be thought of as a challenge to be addressed in a fault tolerant framework but rather a desired feature for the sake of flexibility.

4.4 Addressing the Research Questions

4.4.1 Systematic Review Question 4

What factors influences mobile agent’s fault tolerance execution?

The factors that influence the execution of fault tolerant mobile agents in the existing implementations have been discussed in detail in sections 4.1.

4.4.2 Systematic Review Question 5

How much supports are offered by the mobile agent’s platforms used in implementing the fault tolerance features?

The platform used in the existing fault tolerant mobile agent implementations and the kinds of fault tolerance support they provide has been discussed in detail in sections 4.2.

4.4.3 Systematic Review Question 6

What challenges exist and how do they affects the implementation of the fault tolerance in mobile agents?

The challenges in the existing fault tolerant mobile agent implementations have been discussed in detail in sections 4.3.

4.5 Summary

Mobile agent’s fault tolerance is affected by lots of factors and the field is faced with many unaddressed challenges. In this chapter, the factors have been discussed as well as the challenges. A fault tolerant mobile agent design must take into consideration the attributes mentioned as factors in order to achieve an acceptable level of performance. Investigating deeply in the existing challenges is also a good starting point for considering pressing issues. Unfortunately some of the challenges mentioned in this study can not be incorporated in the framework we are going to look at in the next chapter. Especially challenges in the underlying agent platform are complex to address in the framework, which is far from the platforms. In the next section, we are going to propose a generic framework for fault tolerance mobile agent that will assist in developing a reliable fault tolerance suitable for all failure scenarios.
CHAPTER 5

FAULT TOLERANCE FRAMEWORK

5.1 Fault Tolerance Execution Properties

As stated earlier, there are generally two basic properties a mobile agent’s fault tolerance execution must adhere to in order to be reliable and achieve their design goals. These are the non-blocking and exactly-once execution properties [16], [9].

Sequel to agent execution, the non-blocking property stipulated that the deviation of any infrastructural component (e.g. machine, place, communication link, agent, etc) from its specified services should not impede the execution of the agent. For instance, if a component’s failure may disrupt the agent’s execution while executing on a given place, this might thwart the progress of the execution and during that period, execution cannot go on and is said to be blocked. However, the non-blocking execution property dictates the inclusion of fault tolerance measures so that the execution is continued in spite of failure due to loss of agents. The execution can only progress when the failed place recovers from the failure with the aid of available recovery mechanisms.

On the other hand, the exactly-once property specify that multiple executions of the agent are forbidden. That is, the code of an agent must be subject to only and exactly once execution [16]. The vitality of this property is seen as emanating from certain operations surrounded with adverse consequences especially in the transaction based applications such as banking systems where agent is sent to update accounts.

5.2 Fault Tolerance Framework Execution Cycle

The dynamic nature of mobile agents tends to make fault tolerance mechanisms complex in nature. Equally, the autonomy in the fault tolerance strategies correspond to the fundamental robotic control procedure sense-plan-act, which maps to monitoring, planning and execution stages of the fault tolerance framework. As already recommended by J. Briot et al., fault tolerance techniques should have automatic detection and diagnosis [10]. Considering the mode of operation and observation of the existing fault tolerant mobile agent implementations, we have come up with a three stages/phases cyclic framework (see figure 5.1). The framework process cycle constitutes structure of operation that is used to plan, decide and execute fault tolerance mechanism. The framework execution phases include:

1. Monitor/Detection Phase
2. Action Planning Phase, and
3. Execution of action and continue service Phase

The fault tolerance processing stages can form an abstraction of the fault tolerance framework in mobile agents. These phases apply for any fault tolerance measure in mobile agent execution at any specific stage in the agent’s itinerary and as well form a cycle that starts with the monitoring activity. The phases are discussed as follows.
5.2.1 Monitor/Detection Phase

Every fault model has a monitoring element to detect the presence of the specific fault that needs to be modelled. Depending on the fault tolerance scheme the monitoring activity can be done in a variety of ways. Exchanging heartbeat messages between the agent and the monitoring component is the classic form of monitoring in many of the fault tolerant mobile agent systems. However, other novel approaches for mobile agent failure detection were also introduced in today’s current state of research. For example, some of the existing agent-centric approaches include attaching a probe component with the agent that guides the itinerary of the agent and reports if it detects faults. Another way of monitoring is attaching a monitoring agent that travels with the main agent while monitoring the liveliness of the agent.

In the case of system-centric approach, the monitoring activity is done in the platform and the recovery scheme relies on the information it gets from the platform. A simple example can be a normal exception handling by the platform and notification to the component responsible for the recovery. Also, synchronization of processes with reliable communication protocols is important throughout the monitoring process [7]. This is especially true for replication based fault tolerance schemes since the replicas need to be synchronized in order to successfully takeover when the main agent fails.

5.2.2 Planning Phase

Once the monitoring component of the fault tolerance scheme successfully detects a fault, the next stage is to plan the recovery process. The planning phase of the execution is context based but the context awareness is done in the monitoring phase. The strategy in hand and the context of the application matter for the kind of decision made in the planning phase. The context of the environment in terms of resource availability, nature of execution and application domain must be known to make a planning on the recovery.

Before the planning of the recovery process, the fault needs to be contained and the detection process should map the fault with the available fault models (protocols) designed into the fault tolerant system. Although the planning phase is context-based, there are fundamental activities in employing fault tolerance protocol. In replication based schemes, the planning stage will involve voting and reaching consensus between replicas. The recovery time depends on the effectiveness of the voting and consensus strategy, which is the main element in the planning stage. In checkpoint-based schemes on the other hand, the plan will identify
the latest checkpointed state and messages. For systems that have transactional processes, the planning phase may involve identification of the incomplete transactions for rollback. In the case of an agent-centric approach, all the planning activities such as verifying and loading checkpoint data will be dealt in the planning phase.

### 5.2.3 Execution Phase

The execution phase is the main and most complex phases of fault tolerance process in mobile agent. Depending on the strategy the fault tolerant system has in place, the execution phase may involve a simple retransmission of message if the failure is caused by message corruption or delivery failure of message or a complex reconfiguration of components especially if the type of failure requires that the failed component be replaced. The complex process may involve a series of processes for dynamically creating and reconfiguration of a new agent or reconfiguration of a replica agent until the failed state is brought back to work normally. If the type of failure can not be confined without rolling back some of the execution steps then a process of rolling back of incomplete transactions will be needed.

System-centric approaches may have better flexibility for the developer in the execution phase. This is because activities such as the process state loading and restoration are better accessed via the platform. In the agent-centric approaches, the developer has to take care of the synchronization and communication of processes throughout the recovery process. During the rollback process, effort is geared towards avoiding *domino effect* through resumption from a *consistent state* [30].

The last step in this phase is verifying and resuming of the services. The verification activity is mainly to make sure that the planning and execution phases are completed and exactly once property is fulfilled before the service is continued. Once the service is properly restored, the fault tolerance scheme moves back to the monitoring phase.

### 5.3 Proposed Fault Tolerance Framework

Based on the result of the systematic review in which we highlighted the execution cycle of the current fault tolerance framework shown and discussed in Section 5.2, we propose a generic fault tolerance framework that utilizes mechanism from existing approaches and considers existing challenges. The proposed fault tolerance framework is a form of generic model that does not specifically incline to any of the two major schemes, replication or checkpointing. The framework is for general guidance on where to place the fault tolerance protocols when constructing a fault tolerant mobile agent system. A certain implementation may choose to use one or more of the fault tolerance protocols. In this framework, all options are placed but it is up to the actual implementer to use a certain desirable protocol.

Like the framework process cycle, the proposed framework correspondingly has three core components, namely the Monitoring/Detection, Planning and Execution. The three main components are clearly marked in the proposed framework in figure 5.2. A rather new component in the proposed framework is the Backup Manager, which is a reusable component that is accessed by all the other three components (see Figure 5.2).
5.3.1 Monitor/Detection Component

The Monitoring/Detection component has two main sub-components, namely the monitoring and detection components. The monitoring and detection components are derived from the state-of-the-art feature of a fault tolerant implementation. The Monitoring component is mainly responsible for monitoring the liveness of an agent. It mainly gathers agent activity information and passes it onto the Backup Manager and the detector. Under normal circumstances, the Monitoring component passes recovery information to the Backup Manager. However, if the monitoring activity is disrupted for some reason, the Monitoring component passes notification information to the Detector component.

The Detector component is mainly responsible for resolving failure types based on the information it gets from the Monitoring component. It may use additional sources such as the underlying platform and operating system to clearly resolve the failure types into communication, crash and agent software failure. The resolution of the failure into the three failure types helps for planning a suitable recovery. Once the failure type is resolved, the information is passed on to the Planning component.

5.3.2 Planning Component

The Planning component is mainly responsible for laying out a recovery plan before its execution. The plan may also include fault confinement strategies so that the failure does not propagate throughout the system. Depending on the choice of the fault tolerance approach, the recovery can be switching to readily available replica agents or from stored process state information of the failed agent. The planning for the two types of approaches is different, hence the need for the two subcomponents, namely the Consensus and the Recovery Info Manager components. As the names indicate, the Consensus component is responsible for
conducting election protocol to choose one replica among the available active or passive standby replicas. The Recover Info Manager, on the other hand, is to appropriately determine the latest recovery information and prepare before the execution of the recovery process.

5.3.3 Execution Component

The Execution component is mainly responsible for executing the planned recovery action. The recovery process may start by applying corrective rollback activity, which is an application specific action. The application developer determines the point at which the recovery can be applied. Once this point is determined the planned recovery action will start to execute. Again depending on the fault tolerance approach, either a new replica agent will be created or an already available standby replica is activated to work in place of the failed agent. In case of creating a new agent, the latest available recovery information will be applied to the newly created agent. Activating a passive replica agent also involves applying the latest recovery information to the agent that is going to be activated. However, for active replicas, the activation and resuming of the service is smooth since the active replica is considered to be in a consistent state with the failed agent. Once the recovery execution is completed, an application specific validation may need to be applied to verify the capability to resume service.

5.3.4 Backup Manager Component

The Backup Manager is a reusable component in all of the three fault tolerance execution phases. Its primary purpose is to manage standby agents and process states for recovery. The observer in the Monitoring/Detection component passes either synchronization information for replica agents or process state of the agent for checkpointing through the Backup Manager. In the monitoring/detection phase, the main task of the Backup Manager is to synchronise replicas and store checkpoint of the observed primary agent. The checkpoint data can be in the form of message logging or capturing the process state of the primary agent, hence the subcomponents Message Log and Process State components handle the storage and retrieval of the respective types of recovery information.

During the planning phase, the Backup Manager should provide information about the available backup information. This includes the status of the synchronized replicas and the latest checkpoint state. Additional information about the individual replicas may also be requested during the planning phase. For instance, the Planning component may request for the priority of the replicas before it conducts the consensus process.

In the execution phase, the Backup Manager is mainly responsible to provide the latest recovery information in the form of either the elected replica or process state checkpoint data. The Execution component requests the Backup Manager for the latest recovery data during creation of a replacement agent from the checkpoint data. But if the recovery strategy is to activate a replica agent, then the Backup Manager is supposed to hand in the chosen replica among the available standby agents.

The Backup Manager component is put as a separate element for the sake of transparency and separation of concerns. The individual services of the Backup Manager can be incorporated in the three main stages so as to avoid having a separate entity for managing standby agents and checkpoints, especially if having the Backup Manager separately introduces performance issues. Most of the services of the Backup Manager are consumed by the Monitoring component. The Planning and Execution components only use few of the Backup Manager services during recovery.
5.4 Framework Validation

We do not have the implementation of the proposed model, therefore the validation can not be done straight forward. However, we strongly recommend that any one wishing to follow this framework first have an expert opinion before going into the implementation. Most of the existing agent simulation systems focus on the interaction and performance of agents. The existing simulation systems can not be directly applied to validate the framework since they lack features to model failures. However, some of the simulation systems include ways to describe creation, deletion and dynamic configuration of agents in their simulation system. For example, MADESE simulation framework consists of parallel simulator SimulAgent and IBM Aglets, which can be used to simulate the behaviour of mobile agents running in the Internet. But using such simulators for our case requires modification to the underlying structure to model faults and recovery.

The main attributes of the framework came from our observation of the execution of the existing fault tolerance implementations and map it to the three process cycles mentioned in section 5.2. We believe following the three cycles is one good way to start a fault tolerant implementation in agent systems. We will take a cross section of the key studies with higher assessment scores just to demonstrate on the fact that the existing implementations map to the framework execution cycle.

<table>
<thead>
<tr>
<th>Table 5.1: Mapping of execution in a replication-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach Description:</strong></td>
</tr>
<tr>
<td>Park et al. [40] proposed a Lazy Agent Replication and</td>
</tr>
<tr>
<td>Asynchronous Consensus for the Fault-Tolerant Mobile Agent</td>
</tr>
<tr>
<td>System using three agents; Primary, Replica and Consensus</td>
</tr>
<tr>
<td>with the intent to reduce the execution time overheads posed</td>
</tr>
<tr>
<td>by replications schemes while ensuring non-blocking execution.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scheme:</th>
<th>Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol:</td>
<td>Asynchronous consensus (replicas have priority)</td>
</tr>
<tr>
<td>Fault Class:</td>
<td>agent failure, place failure and system failure</td>
</tr>
<tr>
<td>Monitoring:</td>
<td>• Primary agent at new sides sends messages to replicas following a specified time-out.</td>
</tr>
<tr>
<td></td>
<td>• Messages signal that no failure occurs</td>
</tr>
<tr>
<td>Planning Recovery:</td>
<td>• When failure of the is suspected, primary uses fixed consensus agent to perform consensus step without waiting for other replica</td>
</tr>
<tr>
<td></td>
<td>• Consensus ends with majority votes results</td>
</tr>
<tr>
<td>Recovery Execution:</td>
<td>• At the end of consensus, any replica with highest priority or first detect the primary’s failure takes over the primary</td>
</tr>
<tr>
<td></td>
<td>• Using the primary and 2K replicas, it can tolerate up to K failures. If a replica fails on arrival on new site, other replica can still receive majority messages as long as K+1 agent among the primary, replica and consensus agent are alive.</td>
</tr>
<tr>
<td>Assumptions:</td>
<td>The algorithm assumes a fail-stop model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5.2: Mapping of execution in an adaptive replication-based approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach Description:</strong></td>
</tr>
<tr>
<td>J. P. Briot et al. [33] proposed a strategy that monitors roles and</td>
</tr>
<tr>
<td>activities of agents dynamically to determine agent criticality and</td>
</tr>
<tr>
<td>decide on the number of replica. The dependability of an agent is</td>
</tr>
<tr>
<td>dynamically adjusted by analyzing the criticality of the agent. This</td>
</tr>
</tbody>
</table>
approach consists of three agent types: normal agents, agent-monitors, host-monitors.

<table>
<thead>
<tr>
<th>Scheme:</th>
<th>Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault Class:</td>
<td>Agent processor failure, host crash</td>
</tr>
</tbody>
</table>

**Monitoring:**
- Monitor semantic-level information, which are roles of agents that are defined or maintained by the designer in a role library.
- Monitor system-level information that is the activity of an agent, which is measured by CPU time and communication load such as number of messages exchanged.
- Replication group leaders check the liveness of the replicas.
- Software threads are monitored to detect failure.
- Monitoring agents communicate by exchanging heartbeat messages.

**Planning Recovery:**
- Based on the information gathered by the monitoring process, the adaptive module dynamically adjusts the number of replica agents by considering the criticality of agent.
- When an agent group leader fails, the replicas are notified so that the replica with the highest leadership potential (highest degree of consistency) becomes the new group leader. The degree of consistency is computed continuously for each replica.

**Recovery Execution:**
- When a DarX server crashes, the agents hosted on it are considered lost so replica agents on another host replace the failed agents.
- When a leader of a replication group fails, an election for a new leader will be conducted. In active replication, the replica with the highest degree of consistency takes over the group leader role.
- A replica can be added or removed depending on the criticality of the agent, change in the environment, agent crash, etc.

**Assumptions:**
- Failure detection is unreliable, hence all failed agents are recovered.
- Asynchronous distributed system that no bounds on transmission delays.
- Communication channels are reliable. Communication failures are not permanent, hence network partitions do not occur.
- No malicious or Byzantine failures.

### 5.5 Validity Threats

In any research, whether qualitative or quantitative, validity is considered to be a key attribute that plays essential role in strengthening the trustworthiness, legitimacy, credibility,
quality or usability of the results. Speaking from qualitative researchers’ perspectives, validity through the use of triangulation is believed to remove bias and increase the researcher’s truthfulness of a proposition about some social phenomenon. The triangulation according to Creswell [24] is defined to be “a validity procedure where researchers search for convergence among multiple and different sources of information to form themes or categories in a study”. The use of multiple credible sources to converge to a core conclusion is one way of asserting validity.

One of the procedures to assert validity is the careful reporting of methods and results in the research process. Patton [52] proposed some procedures to be undertaken in order to improve the validity and credibility of the qualitative analysis. Among the procedures are careful reporting of methods and results, using a sample carefully selected to maximize the collection of relevant information, minimising researcher effects, using peer review to review findings, and the testing of rival explanations to those derived from qualitative analysis [52]. In our case, we have strictly followed the Kitchenham et al. [23].

In our study, with reference to the systematic review, the Kitchenham et al. [23] guidelines have strictly been followed starting from the planning to the reporting of the review results. We believe a thorough explanation of the procedures that were used have been provided to enhance readers comprehension of these processes. Although not all processes require reporting but it is imperative to have in mind that reporting any systematic literature review is limited to the information provided in the primary studies. Further more, in the results section, we have provided detailed reporting of our results and analysis. Despite every steps were followed, the key threats to validity in this systematic literature review are related to bias resulting from publication, selection of the studies to be included and as well as inaccuracy in data extraction.

5.5.1 Publication Bias

We considered publication bias as a major validity threat. The research questions stated in this study are geared towards knowing the state-of-the-art in fault tolerance mobile agents and unaddressed challenge in terms of approaches, techniques, platform supports, etc. Concentrating only on a particularly techniques, like the performances of replication approach could limit the results; hence a bias threat. Furthermore, there was no restriction on the sources of information to a particular publisher, journal or conference. The sources used in the systematic review are credible and trusted by the research community. The sources that generated a lot of unrelated results were partially considered. However, the authors have conducted trial searches and carefully reviewed the relevance of the results on all sources before deciding on the partial consideration for some of the sources.

5.5.2 Identification of Primary Studies

In section 2.2.2.3.2 of this work, we have considered the seven major sources or digital libraries considered to be rich and recognised by research bodies world wide, which, in turn, taking the Compendex and Inspec for instance include other important electronic sources such as IEEE, ACM, Springer etc. These sources, which are considered to be reliable, were well explored to arrive at all the available articles in this area of study. However, the main validity issue that may affect the systematic review in this direction is the case of whether we have not done much to cover all the relevant primary studies or most relevant primary studies can be hidden in the sources we excluded like CiteSeer, Google and Yahoo search. Furthermore, another validity threat is related to sampling of the publication years we considered in this review. Some relevant primary studies published before 1998 could be missing and we fear that some relevant papers are hidden in them but we believe their
number is not significant. Moreover, fault tolerant mobile agent computing was at its infancy stage in the years prior to 1998.

The set of exclusion criteria may seem rather fierce in our publication selection process but the main aims and objectives of the study forced us to mainly focus on results from experimental evidence in identifying performance factors and implementation challenges. The scope of the studies with formal mathematical method covered in the review process was somehow broad for us to have achieved full understanding. Excluding studies with formal method has a potential to be a validity threat. However, the authors believe the limited number of studies with formal mathematical method may not significantly affect the result.

Regardless of the facts mentioned above, we are aware that achieving totality is impossible in selecting all primary studies. It is inevitable that some relevant papers may have been missed, but we believe that if they do exist, they are not many and their absence has no significant effect to the information gathered in this study.

5.5.3 Data Extraction Consistency

Patton also argued that, in any qualitative research study, the credibility of the data is directly attached to the credibility of the author [52]. As part of the effort to validate the completeness of our systematic review, the search terms/strings formulated were designed in consultation with the BTH Librarian. A multistage process or three-stage process is adopted and applied to ensure that all relevant information is not left out.

The studies considered in the systematic review are all peer-reviewed studies. The two authors of this systematic review have worked collaboratively and all the selected studies have been analyzed by both authors. During the three-stage data extraction process, the authors have verified each other’s data extraction process individually and together. The criteria applicable in each stage have been detailed in Section 2.3.2.

However, the reservation we have in this regard is if the search terms/strings formulated were not sufficient enough or effectively utilized. Any ways, if such exist we believe it has not counter effect to this report.

5.5.4 Framework Validation

The fact that the framework has not been validated with either simulation or implementation would have a major threat towards the credibility of the framework. However, the scope of this study was to propose a framework and there was no other framework to relate to. Although including the simulation results of the framework in the scope of the study could have boosted the validity of the framework, the unavailability of readily available simulating systems made the simulation effort not suitable to our timeframe. Furthermore, the challenge encountered in involving external experts was that we could not find external experts to review our work in due time. We believe this could be a major validity threat for anyone who is planning to implement the proposed framework; hence we strongly recommend having an expert opinion before venturing into the implementation. However, this threat has no major effect on the validity of the data provided in this work.

5.5.5 Generalization of the Framework
Conclusion validity threats may affect the generalization of the framework. As stated earlier, the fault tolerance aspect depends on a number of factors. Each fault tolerance approach has its own pros and cons in the area of the application domains it is applied. The authors believe the nature of the fault tolerance approaches can generally be categorized in the three execution phases discussed in this study. However, depending on the application domain and context of execution, the properties of the three stages may be merged together for the sake of simplicity. But the authors recommend maintaining separation of concerns for the sake of transparency and maintainability. If the application domain or context has very little use of the specific stage, the implementers can incorporate dummy components to separate concerns. It is difficult to say anything about the performance of the framework without proper simulation or experiment. The authors are reserved from guessing on performance implications at this stage.

5.6 Summary

In this chapter, we have discussed the two basic properties that mobile agent’s fault tolerance must adhere to in order that reliable execution is ensured. We also discussed the framework under which the existing approaches operate on comprising of the monitoring, planning and execution/recovery phases. However, this framework is not generic for the fact that it cannot to be use to tolerate fault in all situation (communication, crash and agent). Moreover, not all challenges should be incorporated in this framework as they can be better addressed elsewhere in the mobile agent system architecture hierarchy. As a stepping stone to fulfilling our research objective and in order to get rid of the impending challenge, we have proposed a general framework based on the current approaches under which fault tolerance can operate irrespective of the schemes, failure types, protocols or platforms intended to addressed The proposed framework is considered a complete cycle that cannot be partially applied to achieve the two main properties of fault tolerance. This section brought this study to an end and in the next section conclusion, possible recommendations and future work shall be drawn based on the analysis of the studies we have obtained.
CHAPTER 6

CONCLUSION AND RECOMMENDATION

6.1 Conclusion

This study started by carrying out a comprehensive systematic review on research papers of fault tolerant mobile agents. The review was basically designed to summarize existing findings, to identify gaps in existing research and to provide framework for new research activities. The review ended with the presentation of detailed reporting of the results and analysis.

Based on the analysis and results obtained, the study takes a closer look at the available mobile agent’s fault tolerance strategies and the challenges that affects its realization. Although the fault models in the primary studies are described differently, they generally fall into the three failure types, namely communication, crash and agent software failure. The failure modes described in the primary studies are stated differently but generally refer to the three failure types. A good example is the descriptions stated as place failure, node crash, hardware failure, server failure, host failure, etc in the primary studies all refer to crash failure.

The fault tolerance strategies were studied with respect to faults resulting from communication, agent software and crash failures. It has been evident that fault tolerance strategies can not address all single-point-of-failures in a system. The complexity and cost of addressing all single-point-of-failures makes the fault tolerance process virtually incomplete. Thus, single-point-of-failures are inevitable in mobile agent and no approach is considered the best in all failure situations. The result obtained from the systematic review shows that there are variety of ways for achieving a specific aspect of fault tolerance in mobile agent systems. For example, a fallback agent can be maintained in the form of a standby replica agent or by logging and maintaining the internal state of the live agent. Both approaches can be used to achieve the task of having a backup agent. The choice of following either way is up to the application domain and degree of fault tolerance that is expected from the system. In some cases, both alternatives can be combined to benefit the most from each of the alternatives. A good example for combining approaches can be the use of combined checkpointing and replication schemes.

The performance overhead of a fault tolerance strategy is inversely related to recovery time. Fault tolerance schemes with short recovery time generally tend to have higher performance overhead. As a result, any fault tolerance strategy should try to make a balance between recovery time and performance overhead. The balance can be achieved by varying the degree of reliability within the fault tolerance protocols. A developer of a mobile agent system must choose a fault tolerance strategy that is suitable to the fault models of his/her application domain. The application domain dictates the degree of fault tolerance needed at each single-point-of-failure. Prioritizing the failure modes by considering failure frequency, cost of redoing tasks in recovery, cost of skipping tasks in recovery, varying number of retries, etc may help choose the suitable fault tolerance strategy.
Furthermore, the challenges facing mobile agent’s fault tolerance are enormous. Challenges such as lack of adequate support from agent’s platforms as well as lack of resource control capabilities have been seen as the greatest set back to the realization of a reliable agent execution and need to be drastically addressed. However, we can say that there is no fault tolerance scheme that can be considered as the best for all situations since the suitability of an approach heavily depends on the application domain.

Lastly, fault tolerant mobile agent is gathering momentum without a clear general framework in its agent system. The existing fault tolerance designs are designed to handle a particular set of fault models and not faults in all situations. The general framework suggested in this study aims to contribute positively to a high level of system dependability and in addressing the challenges and influencing factors affecting the existing fault tolerance models. It is on this note coupled with the results of the systematic review that the generic framework is proposed.

### 6.2 Recommendations

The shortcoming about the fault tolerance schemes considered in this study is that they have not been applied in a large scale except may be for some of the commercial platforms, which at this point we can not verify their level of acceptance in the real world application. This is because most of the implementations are based on simulation results which we believe is hard to ascertain their capabilities and effectiveness. The maturity of the suggested fault tolerance approaches depends highly on the level of acceptance of the approaches. The more the approaches are used, the more the approaches evolve and develop and proved that they work. More large scale tests in real world applications should be applied to the available fault tolerance schemes so as to better demonstrate their reliability. The effects of very long itinerary, many collaborative agents, many replicas, many uncommitted transactions, etc during scalability would need to be investigated so as to support the dependability of fault tolerance techniques.

The measure for completeness of a fault tolerance approach is difficult to weigh. Some of the existing implementations we considered in this study focus on partial list of failure types by stating assumptions to escape from going into specific details of certain failure types. For instance, most approaches assume network link failures to be temporary and state that network partition may not occur. But in reality, such a phenomenon can occur and most likely affects the normal execution of mobile agent. In addition to network partitioning, disconnected operations and deadlocks during consensus stages should be considered in future approaches since they can be thought of as naturally occurring phenomena.

Although the presence of standards for mobile agents helps to alleviate the problem of interoperability, they have inadequate consideration for fault tolerance. An improvement suggestion to the platforms can be the standardization and inclusion of some of the vital fault tolerance protocols such as cloning and resource monitoring. Moreover, it will be more flexible to detect runaway agents from within the platform than building a separate architecture on top of the platforms.

Another recommendation that may offer better flexibility is if the platforms have a mechanism to selectively apply a fault tolerance protocol to a specific agent or agent place as needed. It has been evident that not all fault tolerance protocols are needed at all times; therefore, developers of agent systems should be able to pick their suitable protocol within their application domain.
Researchers in this field seem to focus on adapting the conventional fault-tolerance methodologies to the agent paradigm without carefully analyzing the unique characteristics of existing agent systems and the applications that are built on them [53]. As a result, certain fault tolerance approaches overlooked important issues that are typical to agent system. The inadequate focus on the typical issues has resulted in selection of unsuitable methodology for the recovery of the applications running the agent from faults that affects agents’ execution, migration, and interaction.

Lastly, application programmers should put additional fault tolerance features in order to ensure better reliability. Furthermore, the application of fault tolerance approach in the real world applications could introduce a different set of challenges that have never been thought of. Alternative design approaches that consider scavenging existing resources such as using existing code in previously visited hosts instead of re-transporting code improve the reusability of code.

6.3 Future Work

This study proposed a fault tolerance framework for mobile agents but the scope does not include the validation and implementation of the framework. The study can have several potential future work areas that will strengthen the contribution to the research community.

6.3.1 Framework Validation and Implementation

Simulation of the framework can be a good starting point for the validation of the framework. However, the available mobile agent simulation systems are not suitable to model the fault models described in this study. The simulation systems require some modification or need additional modelling layer for the fault models. Having an expert opinion will contribute to identification of issues the authors might have overlooked and also assists in the validation process. Once the proper validation is conducted, the implementation of the framework will help in the actual validation and verification of the properties of the framework.

6.3.2 Investigate challenges outside the framework

The proposed framework does not address all the challenges mentioned in this study. Some of the challenges such as capturing full process state of agents can not be addressed within the framework since the challenge is deep in the nature of the underlying agent platforms. A potential future work can be investigating deeper in to the challenges coming from the agent platforms, which are not addressed in the proposed framework.

6.3.3 Investigate fault tolerance in other areas

The fault tolerance aspect in this study is limited to the mobile agent paradigm. There are several fault tolerance techniques in other areas of studies that may be adapted to the mobile agents’ paradigm. Although investigating fault tolerance generally is too wide, the investigation of generic fault tolerance techniques may help to extend some of the available fault tolerance approaches mentioned in this study.
REFERENCES


72


# Appendix A

## List of Rejected Studies

<table>
<thead>
<tr>
<th>ID</th>
<th>Year</th>
<th>Source</th>
<th>Authors</th>
<th>Title</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2005</td>
<td>IEEE</td>
<td>Mohammadi, K.</td>
<td>A new approach for mobile agent fault-tolerance and reliability using distributed systems</td>
<td>FATOMAX repetition</td>
</tr>
<tr>
<td>2</td>
<td>2005</td>
<td>WSEAS</td>
<td>Mohammadi, K.</td>
<td>Modeling fault-tolerant and reliable mobile agent execution in distributed systems</td>
<td>Repetition by Mohammadi, K.</td>
</tr>
<tr>
<td>3</td>
<td>2001</td>
<td>IEEE</td>
<td>Pleisch, Stefan et al.</td>
<td>FATOMAS - A fault-tolerant mobile agent system based on the agent-dependent approach</td>
<td>FATOMAS repetition</td>
</tr>
<tr>
<td>4</td>
<td>2005</td>
<td>WSEAS</td>
<td>Mohammadi, K.</td>
<td>A new approach to fault-tolerant mobile agent execution in distributed systems</td>
<td>Repetition by Mohammadi, K.</td>
</tr>
<tr>
<td>5</td>
<td>2005</td>
<td>WSEAS</td>
<td>Mohammadi, K.</td>
<td>A new approach for evaluation fault-tolerant mobile agent execution in distributed systems</td>
<td>Repetition by Mohammadi, K.</td>
</tr>
<tr>
<td>6</td>
<td>2006</td>
<td>Kluwer Academic</td>
<td>Xu, Jie</td>
<td>A dynamic shadow approach to fault-tolerant mobile agents in an autonomic environment</td>
<td>Exception Handling</td>
</tr>
<tr>
<td>7</td>
<td>2005</td>
<td>WSEAS</td>
<td>Mohammadi, K.</td>
<td>Modeling fault-tolerant and secure mobile agent execution</td>
<td>Repetition by Mohammadi, K.</td>
</tr>
<tr>
<td>8</td>
<td>2007</td>
<td>IEEE</td>
<td>Yun, Wang-Xiao</td>
<td>Improvement of temporal-replication mechanism in mobile agent system fault-tolerant model</td>
<td>optimization of [27]</td>
</tr>
<tr>
<td>9</td>
<td>2000</td>
<td>IEEE</td>
<td>Pleisch, Stefan</td>
<td>Modeling fault-tolerant mobile agent execution as a sequence of agreement problems</td>
<td>FATOMAS repetition</td>
</tr>
<tr>
<td>10</td>
<td>2003</td>
<td>IEEE</td>
<td>Park, Taesoon</td>
<td>Fault-Tolerant Execution of Collaborating Mobile Agents</td>
<td>inaccessible</td>
</tr>
<tr>
<td>11</td>
<td>2003</td>
<td>CSREA</td>
<td>Ling, Anthony Wong Kee</td>
<td>Mobile agent-based fault recovery service for distributed environment</td>
<td>inaccessible</td>
</tr>
<tr>
<td>12</td>
<td>2005</td>
<td>WSEAS</td>
<td>Patel, R.B.</td>
<td>Fault-tolerant mobile agents computing</td>
<td>inaccessible</td>
</tr>
<tr>
<td>13</td>
<td>2005</td>
<td>IEEE</td>
<td>Qu, Wenyu</td>
<td>Performance modelling of a fault-tolerant agent-driven system</td>
<td>formal method</td>
</tr>
<tr>
<td>14</td>
<td>2005</td>
<td>IEEE</td>
<td>Qu, Wenyu</td>
<td>Mobile agent-based execution modelling</td>
<td>formal method</td>
</tr>
<tr>
<td>No.</td>
<td>Year</td>
<td>Journal/Book Publisher</td>
<td>Author(s)</td>
<td>Title</td>
<td>Summary</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>------------------------</td>
<td>-----------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>15</td>
<td>2004</td>
<td>IEEE</td>
<td>Qu, Wenyu</td>
<td>Analysis of mobile agents' fault-tolerant behavior</td>
<td>formal method</td>
</tr>
<tr>
<td>16</td>
<td>2000</td>
<td>IEICE</td>
<td>Assis Silva, F.M.D.</td>
<td>Mobile agent-based transactions in open environments</td>
<td>inaccessible</td>
</tr>
<tr>
<td>17</td>
<td>2005</td>
<td>IEEE</td>
<td>Lu, Zhaoxia</td>
<td>Enable efficient stage construction for replication based fault-tolerant execution of mobile agent</td>
<td>no fault tolerance</td>
</tr>
<tr>
<td>18</td>
<td>2005</td>
<td>Springer</td>
<td>Ahn, JinHo</td>
<td>Fault-tolerant and scalable protocols for replicated services in mobile agent systems</td>
<td>protocol improvement</td>
</tr>
<tr>
<td>19</td>
<td>2006</td>
<td>IEEE</td>
<td>Yang, Jin</td>
<td>CIC: An integrated approach to checkpointing in mobile agent systems</td>
<td>checkpoint optimization</td>
</tr>
<tr>
<td>21</td>
<td>2002</td>
<td>IEEE</td>
<td>Park, Taesoon</td>
<td>The performance of checkpointing and replication schemes for fault tolerant mobile agent systems</td>
<td>evaluation of checkpoint and replication</td>
</tr>
<tr>
<td>22</td>
<td>2008</td>
<td>John Wiley and Sons</td>
<td>Yang, Jin</td>
<td>Efficient global checkpointing algorithms for mobile agents</td>
<td>inaccessible</td>
</tr>
<tr>
<td>23</td>
<td>2005</td>
<td>Springer</td>
<td>Yang, Jin</td>
<td>A Framework for Transactional Mobile Agent Execution</td>
<td>focus in transactional execution and not fault tolerance</td>
</tr>
<tr>
<td>24</td>
<td>1998</td>
<td>IEEE</td>
<td>Lee, Keith K.S.</td>
<td>New replication strategy for unforeseeable disconnection under agent-based mobile computing system</td>
<td>no mobile agent focus</td>
</tr>
<tr>
<td>25</td>
<td>2003</td>
<td>IEEE</td>
<td>Mishra, Shivakant</td>
<td>Interagent communication and synchronization support in the DaAgent mobile agent-based computing system</td>
<td>communication aspect of DaAgent</td>
</tr>
<tr>
<td>26</td>
<td>2006</td>
<td>IEEE</td>
<td>Beheshti, S.</td>
<td>Fault tolerance in mobile agent systems by cooperating the witness agents</td>
<td>optimization of [34]</td>
</tr>
<tr>
<td>27</td>
<td>2003</td>
<td>IEEE</td>
<td>Xu, Jie</td>
<td>A dynamic shadow approach for mobile agents to survive crash failures</td>
<td>Exception Handling</td>
</tr>
<tr>
<td>28</td>
<td>2003</td>
<td>ACTA Press</td>
<td>Pleisch, S.</td>
<td>Execution atomicity for non-blocking transactional mobile agents</td>
<td>FATOMAS repetition</td>
</tr>
<tr>
<td>29</td>
<td>2002</td>
<td>ACTA Press</td>
<td>Yanphanich, W.</td>
<td>A fault tolerant pipelined cluster model</td>
<td>Cluster fault tolerance</td>
</tr>
<tr>
<td>30</td>
<td>2002</td>
<td>IEEE</td>
<td>Pleisch, S.</td>
<td>Non-blocking transactional mobile agent execution</td>
<td>FATOMAS repetition</td>
</tr>
<tr>
<td>#</td>
<td>Year</td>
<td>Publisher</td>
<td>Author(s)</td>
<td>Title</td>
<td>Category</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
<td>-------------</td>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>32</td>
<td>2006</td>
<td>IEEE</td>
<td>Yang, Jin</td>
<td>Checkpoint Placement Algorithms for Mobile Agent System</td>
<td>checkpoint optimization</td>
</tr>
<tr>
<td>35</td>
<td>2006</td>
<td>Kluwer Academic Publishers</td>
<td>Dragoni, N.</td>
<td>Crash failure detection in asynchronous agent communication languages</td>
<td>formal approach</td>
</tr>
<tr>
<td>37</td>
<td>2000</td>
<td>IEEE</td>
<td>Gendelman, E.</td>
<td>An application-transparent, platform-independent approach to rollback-recovery for mobile agent systems</td>
<td>no fault tolerance</td>
</tr>
<tr>
<td>38</td>
<td>2005</td>
<td>Springer</td>
<td>Park, Taesoon</td>
<td>Mobile Agent Based Fault-Tolerance Support for the Reliable Mobile Computing Systems</td>
<td>use of MA for FT</td>
</tr>
<tr>
<td>39</td>
<td>2003</td>
<td>IEEE</td>
<td>Simon Pears et al.</td>
<td>Mobile agent fault tolerance for information retrieval applications: an exception handling approach</td>
<td>Exception Handling</td>
</tr>
<tr>
<td>40</td>
<td>2006</td>
<td>ACM</td>
<td>Sapna E. George et al.</td>
<td>Movement-based checkpointing and logging for recovery in mobile computing systems</td>
<td>no mobile agent focus</td>
</tr>
<tr>
<td>41</td>
<td>2006</td>
<td>Springer</td>
<td>Gu Su Kim et al.</td>
<td>Domain-Based Mobile Agent Fault-Tolerance Scheme for Home Network Environments</td>
<td>specific application in home network</td>
</tr>
<tr>
<td>45</td>
<td>2004</td>
<td>Springer</td>
<td>Ahn, JinHo</td>
<td>Fault-Tolerant and Scalable Communication Mechanism for Mobile Agents</td>
<td>optimization of communication</td>
</tr>
<tr>
<td>48</td>
<td>1998</td>
<td>Springer</td>
<td>Joachim Baumann et al.</td>
<td>The shadow approach: An orphan detection protocol for mobile agents</td>
<td>orphan detection</td>
</tr>
<tr>
<td></td>
<td>Year</td>
<td>Publisher</td>
<td>Author(s)</td>
<td>Title</td>
<td>Subject</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>49</td>
<td>1999</td>
<td>IEEE</td>
<td>Uwe G. Wilhelm et al.</td>
<td>A pessimistic approach to trust in mobile agent platforms</td>
<td>Security</td>
</tr>
<tr>
<td>50</td>
<td>2002</td>
<td>Springer</td>
<td>Holger Pals et al.</td>
<td>FANTOMAS Fault Tolerance for Mobile Agents in Clusters</td>
<td>Cluster fault tolerance</td>
</tr>
<tr>
<td>51</td>
<td>2006</td>
<td>Springer</td>
<td>Sebnem Bora</td>
<td>A Fault Tolerant System Using Collaborative Agents</td>
<td>replication of data and servers</td>
</tr>
<tr>
<td>52</td>
<td>2002</td>
<td>IEEE</td>
<td>Hyunjoo Kim</td>
<td>The cost of checkpointing, logging and recovery for the mobile agent systems</td>
<td>FT evaluation and challenges</td>
</tr>
<tr>
<td>53</td>
<td>2002</td>
<td>IEEE</td>
<td>Komiya, T.</td>
<td>Mobile agent model for distributed systems</td>
<td>Repetition of [47]</td>
</tr>
<tr>
<td>54</td>
<td>2000</td>
<td>ACM</td>
<td>A. Silva et al.</td>
<td>Reliability requirement in mobile agent system</td>
<td>no fault tolerance</td>
</tr>
</tbody>
</table>