Palestine Polytechnic University

College of Engineering and Technology
Computer and Electrical Engineering Department

Graduation Project

Fault Tolerance of Mobile Agent (Replication _Based Technique)

Project Team
Hala Al-Jbareen Suhad Al-Awawdeh Thikrayat Al-Jawa'deh

Project Supervisor
Eng. Amal Al-Dweik

Hebron – Palestine

May, 2006

جامعة بوليتكنيك فلسطين
الخليل – فلسطين
كلية الهندسة و التكنولوجيا
دائرة الهندسة الكهربائية و الحاسوب

اسم المشروع

Fault Tolerance of Mobile Agents
Replication-based Technique

أسماء الطلبة

ذكريات يوسف الجواعدة سهاد حسن العواودة هالة نايف الجبارين

بناء على نظام كلية الهندسة و التكنولوجيا و إشراف و متابعة المشرف المباشر على المشروع و موافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية و الحاسوب و ذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص أنظمة الحاسوب.

توقيع المشرف

.....

توقيع رئيس الدائرة

.....

الإهـــــــــــــــــــــــــــداء

إلى من علمونا كيف نتعلم, إلى من تعبوا وسهروا وقاموا ليلهم من أجلنا, من أحاطونا بحبهم و رعايتهم وحنانهم, من أثقلتهم هموم الحياة فأبوا إلاّ أن يقدموا لنا ويعطونا بلا انتظار مقابل.....

إلى من شاركونا أفراحنا و آلامنا, من تحملونا وصبروا علينا في لحظات تعبنا وسهرنا, من تمنّوا لنا التوفيق والنجاح بقلوب صادقة, من سرنا معاً درب الحياة يداً بيد, نعلمهم ونتعلم منهم وإن كانوا أصغر منا سناً.....

إلى من اخترناهن فأحببناهن, من ساعدننا و وقفنّ بجانبنا دائماً, من كنّ أخوات لنا لم تلدهنّ أمهاتنا فكنّ أخوات كما لو ولدنّ معنا, من شاركننا في الدمعة والبسمة....

إلى كل من نعرفهم ولا نعرفهم.....

إلى من يعجز القلم أن يعبر لهم ولا تكفي الدفاتر لتشكرهم....

إلى الوالدين الغاليين..... الإخوة الأحبّة..... الصديقات العزيزات.....

إليكم نهدي تعبنا وجهدنا ومشروعنا وثمرة دراستنا وتعبكم وسهركم ودعمكم لنا, نهديه لكم ليس رداً للجميل فجميلكم كبير, ولكن شكر قليل لتعب كثير....

Acknowledgment

We complete our project, with the help of our Blessing Allah, so thanks Allah for everything….

Then we want to thank our teacher and supervisor "Eng. Amal Al-Dweik", who supports all the time, so thanks a lot for her….

Also we want to thank our teacher "Eng. Mazen Zalloum" for everything….

Finally, we want to thank and dedicate our project to all our teachers and colleagues in our university, the University of Martyrs and Hero's, Palestine Polytechnic University…

Thanks a lot for all of them, and thanks for everybody who helps us…..

Abstract

Fault Tolerance of Mobile Agents (Replication-based Technique) is a system used to tolerate faults in a case of crash of machines, platforms, and agents. The system will use JADE framework to simulate our work. The work mainly uses the fault tolerance tool available at JADE to be tested and work on. The main application to be used here is information retrieval application; where some specific information will be gathered and retrieved from several hosts. In case of failure, several techniques could be used to solve this degradation and loss of performance and reliability.

Replication-based techniques would be one of the possible solutions to solve the problem of system failure. We apply the existing mechanism and start working on our suggested technique.

يتحدث مشروعنا عن متجاوز أو متحمل الأخطاء (Fault Tolerance) الخاص بالوكيل أو العميل المتحرك باستخدام آلية إعادة الرد أو بالأحرى آلية استرجاع نسخة مطابقة حيث هو عبارة عن نظام مستخدم ليتجاوز عن الأخطاء ويتحملها في حالة تحطّم الآلات، المنصات، أو العناصر أو حتى العملاء سيستخدم النظام هيكل باستخدام لغة البرمجة جافا (Java) أداء عملنا.

العمل في الغالب سيطبق على التطبيقات التي بها استرجاع معلومات, حيث بعض المعلومات الخاصّة ستجمع تسترجع من عدّة مضيفين.في حالة إخفاق أو فشل النظام، تتوفر عدّة تقنيات قد تستخدم لحلّ هذا الإخفاق و الخسارة في الأداء و الوثوقية.هذه التقنية المستخدمة ستكون تكون أحد الحلول الممكنة لحلّ مشكلة إخفاق النظام, سنستخدم بعض الآليات الموجودة, نقارن بينهم, نحلل النتائج , وفي النهاية قد يقدر نظامنا أن يكون متكاملا ضمن بيئتنا الحالية.

Table of Contents

Chapter One: Introduction

1.1 Overview..... 1
1.2 General Idea of the System..... 1
1.3 Literature Review about Agent..... 1
1.4 Development Organization..... 3
1.5 Estimated Cost..... 3
1.5.1 Development Cost..... 3
1.5.2 Implementation Cost..... 5
1.6 Project Scheduling..... 5
1.7 Project Management..... 9
1.7.1 Human Resources..... 9
1.7.2 Allocation of rules of system..... 9
1.8 Risk Management..... 10

Chapter Two: System Requirement

2.1 Overview..... 12
2.2 Theoretical Background..... 12
2.2.1 Distributed System..... 12
2.2.2 Models of Distributed System..... 13
2.2.2.1 Client server model..... 13
2.2.2.2 Remote Procedure calls..... 14
2.2.2.3 Code on Demands..... 15
2.2.2.4 Message Passing..... 15
2.2.2.4.1 Message passing features..... 15
2.2.2.4.2 Message Processing..... 16
2.2.2.5 Software agent Model..... 16
2.2.2.5.1 Forms of software agent..... 16
2.2.3Mobile agent..... 17
2.2.3.1 Security..... 18

2.2.3.2 Agent Platforms..... 19
2.2.3.3 Agent Services..... 20
2.2.3.4 Representative mobile agent systems..... 20
2.2.4 Fault Tolerance..... 21
2.2.4.1 What is fault tolerance?..... 21
2.2.4.2 Basic concepts..... 22
2.2.4.3 Types of fault..... 22
2.2.4.4 Classification of failure module..... 23
2.2.4.5 Scope..... 23
2.2.5 Replication..... 23
2.2.5.1 What is Data Replication?..... 23
2.2.5.2 Data Replication Benefits..... 24
2.2.5.3 Data Replication Techniques..... 24
2.3 Java Agent Development Framework (JADE)..... 25
2.3.1 Containers and platforms..... 26
2.3.2 JADE features..... 27
2.3.3 FIPA (Foundation for Intelligent Physical Agents)..... 28
2.3.3.1 FIPA-Agent-Management ontology..... 28
2.3.4 The ACL language..... 28
2.3.4.1 ACL message..... 29
2.4 Software Requirements Specification..... 31
2.4.1 System Definition..... 31
2.4.2 Functional Requirements..... 31
2.4.3 Non-Functional Requirements..... 32

Chapter Three: Architecture Design

3.1 Overview..... 33
3.2 System Objectives..... 33
3.3 System Block Diagram..... 34
3.3.1 The Agent Application Specification..... 34
3.3.2 Fault Tolerance (FT) Module..... 35
3.3.2.1 Fault Tolerance Model..... 35
3.3.2.2The Cause-and –Effect Relationship..... 37
3.3.3 Jade System..... 37
3.3.3.1 JADE Features..... 37

3.3.3.2 Why Using JADE?..... 39
3.3.4 Output Model.....43
3.4 System Architectures.....44
3.5 How Does the System Work?.....45
3.5.1 JADE Tools..... 47
3.5.2 Replication-based Technique..... 48
3.6 System Modeling.....51
3.6.1 Use-Case.....51
3.6.1.1 Agent-To-Agent Relationship..... 51
3.6.1.2 Agent-To-Container Relationship.....52
3.6.2 Sequence Diagram.....53

Chapter Four: Detailed System Design

4.1 Overview..... 55
4.2 Design Models..... 55
4.2.1 The agent application specification module..... 55
4.2.2 JADE Module..... 56
4.2.3 Fault Tolerance Module..... 57
4.3 JADE Component..... 58
4.4 Graphical User Interface of the system..... 61
4.5 General Algorithms and Flowcharts..... 65
4.5.1 Algorithm 72

Chapter Five: Implementations and Testing

5.1 Overview..... 73
5.2 Implementations..... 73
5.2.1 The code was built..... 73
5.2.2 Command lines..... 75
5.2.2.1 -mtp jade.mtp.iiop.MessageTransportProtocol..... 75
5.2.2.2 –name –port..... 80
5.2.2.3 –port –container..... 81
5.2.2.4 –backupmain..... 85
5.3 Testing..... 88
5.3.1 Work Results..... 88

Chapter Six: Conclusion and Future Work

6.1 Future Works..... 98
6.2 Conclusions.....98

Table of Figures

1. Figure (2.1) 13
2. Figure (2.2) 18
3. Figure (2.3) 19
4. Figure (2.4) 19
5. Figure (2.5) 26
6. Figure (2.6) 30
7. Figure (3.1) 34
8. Figure (3.2) 34
9. Figure (3.3) 35
10. Figure (3.4) 36
11. Figure (3.5) 37
12. Figure (3.6) 44
13. Figure (3.7) 49
14. Figure (3.8) 49
15. Figure (3.9) 50
16. Figure (3.10) 51
17. Figure (3.11) 52
18. Figure (3.12) 52
19. Figure (3.13) 52
20. Figure (3.14) 53
21. Figure (3.15) 54
22. Figure (3.16) 54
23. Figure (4.1) 55
24. Figure (4.2) 56
25. Figure (4.3) 56
26. Figure (4.4) 57
27. Figure (4.5) 58
28. Figure (4.6) 61
29. Figure (4.7) 62
30. Figure (4.8) 62
31. Figure (4.9) 63
32. Figure (4.10) 63
33. Figure (4.11) 64
34. Figure (4.12) 64

35.Figure (4.13)..... 65
36.Figure (4.14)..... 65
37.Figure (4.15)..... 66
38.Figure (4.16)..... 67
39.Figure (4.17)..... 68
40.Figure (4.18)..... 69
41.Figure (4.19)..... 70
42.Figure (4.20)..... 70
43.Figure (4.21)..... 71
44.Figure (4.22)..... 71
45.Figure (4.23)..... 72
46.Figure (5.1)..... 73
47.Figure (5.2)..... 74
48.Figure (5.3)..... 74
49.Figure (5.4)..... 75
50.Figure (5.5)..... 76
51.Figure (5.6)..... 76
52.Figure (5.7)..... 77
53.Figure (5.8)..... 77
54.Figure (5.9)..... 78
55.Figure (5.10)..... 78
56.Figure (5.11)..... 79
57.Figure (5.12)..... 79
58.Figure (5.13)..... 80
59.Figure (5.14)..... 80
60.Figure (5.15)..... 81
61.Figure (5.16)..... 81
62.Figure (5.17)..... 82
63.Figure (5.18)..... 82
64.Figure (5.19)..... 83
65.Figure (5.20)..... 83
66.Figure (5.21)..... 84
67.Figure (5.22)..... 84
68.Figure (5.23)..... 85
69.Figure (5.24)..... 86
70.Figure (5.25)..... 86

71.Figure (5.26)..... 87
72.Figure (5.27)..... 87
73.Figure (5.28)..... 89
74.Figure (5.29)..... 89
75.Figure (5.30)..... 90
76.Figure (5.31)..... 90
77.Figure (5.32)..... 91
78.Figure (5.33)..... 91
79.Figure (5.34)..... 92
80.Figure (5.35)..... 92
81.Figure (5.36)..... 93
82.Figure (5.37)..... 93
83.Figure (5.38)..... 94
84.Figure (5.39)..... 94
85.Figure (5.40)..... 95
86.Figure (5.41)..... 95
87.Figure (5.42)..... 96
88.Figure (5.43)..... 96
89.Figure (5.44)..... 97
90.Figure (5.45)..... 97

List of Tables

1. Table (1.1) 3
2. Table (1.2) 4
3. Table (1.3) 4
4. Table (1.4) 4
5. Table (1.5) 5
6. Table (1.6) 7
7. Table (1.7) 8
8. Table (2.1) 23
9. Table (3.1) 46

Chapter One
Introduction

Overview
General Idea of the System
Literature Review about Agent
Development Organization
Estimated Cost
Project Scheduling
Project Management
Risk Management

Chapter One

Introduction

1.1 Overview

In this chapter we will discuss the literature review about agents, development organization, estimated cost, project scheduling and the risk management for our system (fault tolerance of mobile agents).

1.2 General Idea of the System

The work reported herein deals in particular with fault-tolerance of mobile agents, that is, on how to ensure a service up to fulfilling the system's function even in the presence of "faults", namely, events having their origin inside or outside the system boundaries and possibly introducing unexpected deviations of the system state.

The system specifies fault-tolerant mobile agent execution using the replication technique. Several approaches are used, we will implement some. The analysis of these techniques is concluded at the end of the work.

1.3 Literature Review about Agents

Many previous literature reviews are done about agents. Some are:

- Hyacinth S. Nwana and Divine T. Ndumu at the Applied Research & Technology Department in British Telecommunications Laboratories made a research about "A Perspective on Software Agents". They talked about software agents and some related topics such as multi-agent systems (promises and

reality), and personal and information agent. This research sat out, ambitiously, a brief reappraisal of software agents. They mentioned that however some five years after the word 'agent' came into vogue in the popular computing press, it is perhaps time the efforts in this fledgling area are thoroughly evaluated with a view to refocusing future efforts. They do not pretend to have done this in their research. The research contains some strong views not necessarily widely accepted by the agent community.

- Hyacinth S. Nwana for the Intelligent Systems Research for the Advanced Applications & Technology Department worked on "Software Agents". The author wrote about software agents and other related topics. This research presented a typology of agents; it placed agents in context, defined them use them made an overview for critically the rationales, hypotheses, goals, challenges and state-of-the-art demonstrators of the various agent types in their typology.

- Philip R. Cohen & Adam Wang wrote their paper about "An Open Agent Architecture" in Stanford University they build a system that support distributed execution of a user requests, interoperability of multiple application subsystems, addition of new agent, and incorporation of existing application.

- Stefan Pleisch & Ander' Schiper they wrote about the "Modeling Fault-Tolerance Mobile Agent Execution as a Sequence of Agreement Problem". This paper talked about the fault tolerance that it is fundamental to the future development of mobile agent application, and fault tolerance prevent a partial or complete loss of the agent and ensure that the agent arrives at its destination by using simple approach such as check pointing, and talk about a novel approach to

fault tolerance mobile agent execution which based on modeling agent execution as a sequence of agreement problem.

1.4 Development Organization

The team consists of three persons, and all the team members' work together to complete this project. The team works in parallel without explicit distribution for the activities. Each member participates in each activity of this project. We started the work step by step and activity-by-activity.

1.5 Estimating Cost

This section lists the estimated costs for this system including the development costs and the implementation costs.

1.5.1 Development Cost

Hardware cost

The following table shows the hardware cost.

Note: - these computers will be used for other applications.
- The estimated cost is considered according to that most of the equipments (PC, printer, and so ..) are rented not owned or purchased.

Table (1.1): development hardware cost.

Hardware Component	Cost
PC with Pentium 4-2800 MHz	300$
Printer HP Laser	100$
Flash Memory	20$
Floppies and CD'S	10$
Total	430$

Software Cost

The following table shows the software cost.

Table (1.2): development hardware cost.

Software Component	Cost
Windows XP professional	200$
J-Builder	100$
Internet	200$
Simulator(JADE)	Free (open source)
Total	500$

Human Resources Cost:

The following table shows the human resources cost.

Table (1.3): human resources cost.

No.	Work hours\developer	Cost\hour	Total human cost
3 Developers	(20hours)*(4weeks)*(6months) = 480hours	4$	(480)*(3developer)* (4$) = 5760$

Total Development Cost:

The following table shows the total cost of the system.

Table (1.4): total development cost.

Hardware development cost	Software development cost	Hunan development cost	Other development cost	Total
430$	500$	5760$	1000$	7690$

1.5.2 Implementation Costs:

We need a PC which costs 300$.

Total system cost:

The following table shows the total cost of the system.

Table (1.5): total development cost.

Development Cost	Implementation Cost	Total System Cost
8890$	300$	9190$

1.6 Project Scheduling

The project time needed is thirty two weeks, there is one week backup for the project and the work in project began on 1/10/2005, and will finish on 31/5/2006.
The main activities in our system are:

- **(T1) Collecting information and theoretical issues**

At the beginning we must collect the requirements from the stakeholders and then we will write a feasibility study to determine if we can continue or not with our budget.

➔ **(MI)** The milestone will be a feasibility report.

- **(T2) Analyze the concepts**

In this activity we must study and analyze the requirements very well to make decisions regarding what the new system will be according to users and stakeholders defining user and system requirements,……etc .

➔ **(M2)** The milestone will be a document.

- **(T3) System modeling and design**

After collecting the requirements and writing the feasibility study, we will model the system by using the dataflow because we fing it the most appropriate representation.
➔ (**M3**) The milestone will be Architecture model, design.

- **(T4) System implementation**

In this stage and after the designers and programmers complete their work, we must check for validity of the system by implementation it.
➔ (**M4**)The milestone will be a code report.

- **(T5) System testing**

This will include component, integration, the defect testing, and to be sure that the system work as good as we want.
➔ (**M5**)The milestone will be a report.

- **(T6) System documentation**

This phase will start with the project and end with it
The milestone will be a complete project delivered at the end of the semester.

Table (1.6): The system development time scheduling

Activities(tasks)	Symbol	Duration(Week)	Dependent
Collect the info. & the requirement and writing feasibility study.	T1	3	M1
Analyze the info. & the requirements	T2	3	T2(M2)
Design	T3	6	T2 (M3)
Implementation	T4	10	T3(M4)
Testing	T5	6	T4(M5)
Backup week	T6	2	
Documentation	T7	3	T1,T2,T3,T4,T5 Complete SW project
Final Project		32 weeks	

Table (1.7) : system plan

Weeks / Tasks	1-3	4-6	7-13	14-23	24-29		
Collect the info. & the equirement							
Analyze the info. & the requirements							
Design							
Implementation							
Testing							
Documentation							

1.7 Project Management

Explain the team work and the relations between team members

1.7.1 Human Resources

System development work personnel and their supervisor :

1. Hala Naief Jabareen
2. Suhad Hasan Awawdeh
3. Thikrayat Yousef Jawa'deh

1.7.2 Allocation of Roles of System Developers

There is no constant situation for any team member job, because we will do the whole work together. But in general we put the allocation of roles of system developers as follows:

1. **Leader (Suhad)**: responsible of planning, scheduling and controlling flow of system development processes.
2. **Programmer (Hala)**: responsible of the system programming, implementation testing also her nuts has enough experience in the java development.
3. **Software engineer (Thikrayat)**: responsible for the documentation and tracing of the development stages of the software.

1.7 Project Management

Explain the team work and the relations between team members

1.7.1 Human Resources

System development work personnel and their supervisor :

1. Hala Naief Jabareen
2. Suhad Hasan Awawdeh
3. Thikrayat Yousef Jawa'deh

1.7.2 Allocation of Roles of System Developers

There is no constant situation for any team member job, because we will do the whole work together. But in general we put the allocation of roles of system developers as follows:

1. **Leader (Suhad)**: responsible of planning, scheduling and controlling flow of system development processes.
2. **Programmer (Hala)**: responsible of the system programming, implementation testing also her nuts has enough experience in the java development.
3. **Software engineer (Thikrayat)**: responsible for the documentation and tracing of the development stages of the software.

1.8 Risk Management

In this project the team may face the following risks:

1. Some activities may not be made on time for some reasons, such as one of the team members might become ill.
2. The system may fail in the operation stage.
3. The user may not be able to use the program.
4. The time required to develop the software is underestimated.
5. The costs of the project may exceed the estimated costs, so the budget of the project will not cover this cost.
6. Political situation may affect the time schedule. Like investment and suspensions in university.

Table (1.7) System Risk Analysis.

Objectives :	JADE (Java Agent Development Framework) will be used to stimulate the fault tolerance of mobile agent (replication-based technique)
Constraints:	Delivering the system in three months. Maintain the flexibility of the simulation (JADE) Maintain the highest level of security. Acceptance by the audience.
Risk:	**Nonfunctional risk :** Required time more than the estimated time. New cost may appear. Technology, which is used in simulator, may be changed. Operational risk : Incompatibility between systems with environment. **Functional risk:** Such as requirements changes. Operational risk : Some applications don't perform as estimated such as the system interface is not comfortable or acceptable users. Appearance of new requirement after or during development stage.
Risk resolution:	The simulator must work in environments that have fewer specifications. Study the planning process and take right steps to deliver the system in the desired state. Study all the requirement and identify the importance of each one. Training the users to operate this simulation . Apply the scheduling table.
Results :	Creation of simulator by using J-Builder

Chapter Two
System Requirement

Overview
Theoretical Background
Java Agent Development Framework (JADE)
Software Requirements Specification

Chapter Two
System Requirements

2.1 Overview

In this chapter we are going to introduce all the requirements of the system. These will be described using different notations and models.

2.2 Theoretical Background

2.2.1 Distributed System

Distributed system is a collection of independent computers that appears to its users as a single coherent system.
There are many advantages of the distributed system some of these:

1. Over the centralized ones as follows:
    - Economics: microprocessor offers a better price and performance than mainframe.
    - Speed: a distributed system may have more total computing power than mainframe.
    - Inherent distribution: some application involves specially separated machines.
    - Reliability: if one machine crashes, the system as a whole can still survive.
    - Incremental growth: computing power can be added in small increments.

2. Over personal computers as follows:
   - Data sharing: allow many users to access to a common data base.
   - Device sharing: allow many users to share expensive peripherals.
   - Communication: make human-to-human communication easier.
   - Flexibility: spread workload over available machines in most cost effective way.[1]

2.2.2 Models for Distributed Systems:

This section will describe the models of distributed system, the client server model, remote procedure call, code on demand, message passing and the software agent model.

2.2.2.1 Client-Server Model:

Client Server-Model is a base topic for topics describing models of interacting Clients and Servers. In the Client Server-Model, a Server waits for requests from Clients, and after receiving such a request, the Server processes it and sends a response. The communication between the two peers is based on a Protocol, which is defining the possible interaction patterns and the information being exchanged. Client Server is a general description of a networked system where a client program initiates contact with a separate server program for a specific function or purpose. The client exists in the position of the requester for the service provided by the server. [1]

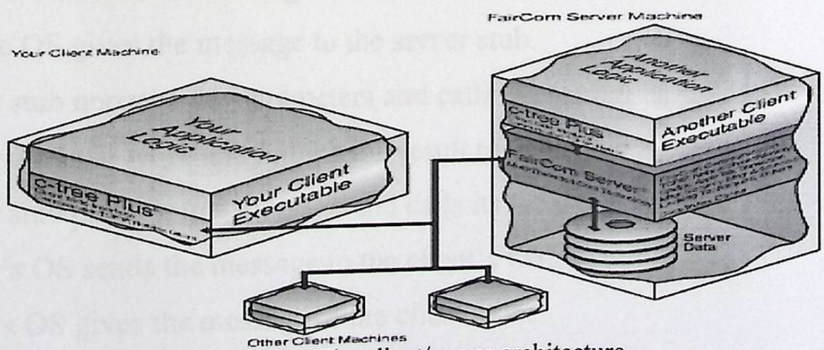


Figure 2.1 The client/server architecture.

2.2.2.2 Remote Procedure Calls (RPC)

Remote Procedure Calls (RPC) is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure.

The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports. RPC makes the client/server model of computing more powerful and easier to program.

When combined with the ONC RPCGEN protocol compiler clients transparently make remote calls through a local procedure interface.[2]

A remote procedure call occurs in the following steps:-

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client. [2]

2.2.2.3 Code on Demand (CoD)

Mobile agent paradigm has been proposed as a promising solution to facilitate distributed computing over open and heterogeneous networks. Mobility, autonomy, and intelligence are identified as key features of mobile agent systems and enabling characteristics for the next-generation smart electronic commerce on the Internet.

However, security-related issues, especially integrity protection within mobile agent technology, still hinder the widespread use of software agents: from the agent's perspective, mobile agent integrity should be protected against attacks from malicious hosts and other agents.

2.2.2.4 Message Passing

A message is a structured piece of information sent from one agent to another over a communication channel. Some messages are requests made to one agent by another, other messages deliver data or notification to another agent. A message consists of a message identifier and, if needed, a set of message arguments. The message identifier tells the receiver the purpose or type of the message. The arguments to the message contain additional information that is interpreted based on the type of message. They may contain the object of an action, or they may contain information used to carry out a request .

2.2.2.4.1 Message passing features

The types of messages:

1. Informative: send information for the object to update itself.
2. Interrogative: ask an object to reveal some information about itself
3. Imperative: take some action on itself, or another object

2.2.2.4.2 Message processing

When you're developing an agent that will be exchanging messages, it's important to think about how message processing will integrate with the rest of the objects making up the agent. Ideally, you'd like to:

- Isolate communications details from application details
- Provide a structured way to link messages to method calls on application. [3]

2.2.2.5 Software Agent Model

A Software Agent (or Autonomous Agent or Intelligent Agent) is a computer program which works toward goals (as opposed to discrete tasks) in a dynamic environment (where change is the norm) on behalf of another entity (human or computational), possibly over an extended period of time, without continuous direct supervision or control, and exhibits a significant degree of flexibility and even creativity in how it seeks to transform goals into action tasks.

2.2.2.5.1 Forms of Software Agents

As there is no hard definition of a software agent, here are the various forms of software agent that identified, to date:

- **System Agent**: a continuously running background task or daemon that typically has a specialized, pre-programmed purpose.
- **User Agent**: the client-side user interface for a sophisticated network application. For example, the web browser for accessing the World Wide Web (WWW).
- **Client Agent**: the client-side user interface for a server-based application such as a search engine or deal-seeking. The server side is usually simply performing a

database lookup on a database that is independently maintained by some form of web crawler that monitors changes to web sites.

- **Intelligent Agent (IA)**: a program that performs a task or pursues goals with minimal specific direction, using intelligent or heuristic techniques to the effect that the user is very impressed that a computer could be so smart. An IA does not need to be mobile since the vast range of information on the web can be accessed remotely.
- **Mobile Agent (MA)**: an autonomous program that migrates between host systems in the process of pursuing one or more goals. The MA does not need to be truly intelligent, but have enough flexibility to deal with an environment in which things can change or be inaccessible at any time.
- **(Closed) Multi-Agent System**: a tightly integrated application environment in which portions of the application are parceled out to mini-programs that pursue sub-goals.
- **Open Multi-Agent System**: a loosely integrated, distributed application environment in which independently constructed programs (agents) can participate in structured interactions in which the agents have a significant degree of common interest (community).[4]

2.2.3 Mobile Agents

Mobile agents are programs that can migrate from host to host in a network, at times and to places of their own choosing. The state of the running program is saved, transported to the new host, and restored, allowing the program to continue where it left off.

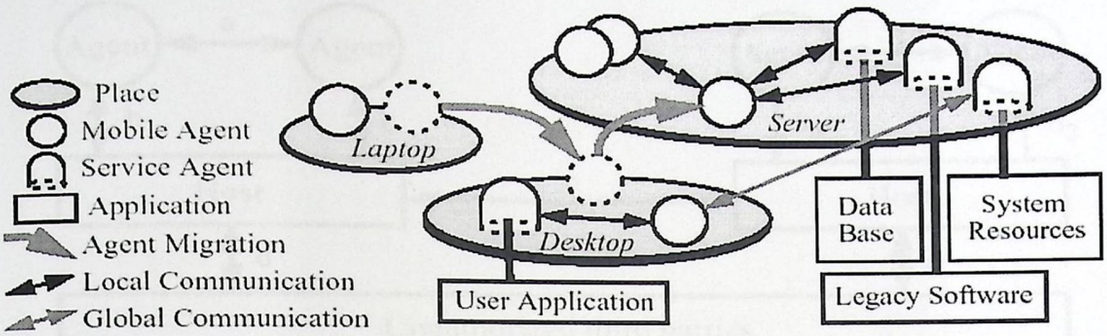


Figure 2.2 A Mobile Agent System

A mobile agent contains:-

- *Code*: the program (in a suitable language) that defines the agent's behavior.
- *State*: the agent's internal variables etc., which enable it to resume its activities after moving to another host.
- *Attributes*: information describing the agent, its origin and owner, its movement history, resource requirements, authentication keys, for use by the infrastructures.[5]

2.2.3.1 Security:

The vision of mobile agents as the key technology for a wide range of networked applications, such as electronic commerce or systems management, can only become reality if all security issues are well understood and the corresponding mechanisms are in place. As illustrated in Figure (2.3), security issues in mobile agent systems can be subdivided into four areas, namely (a) inter-agent security, (b) agent-host security, (c) inter-host security, and (d) security between hosts and unauthorized third parties.

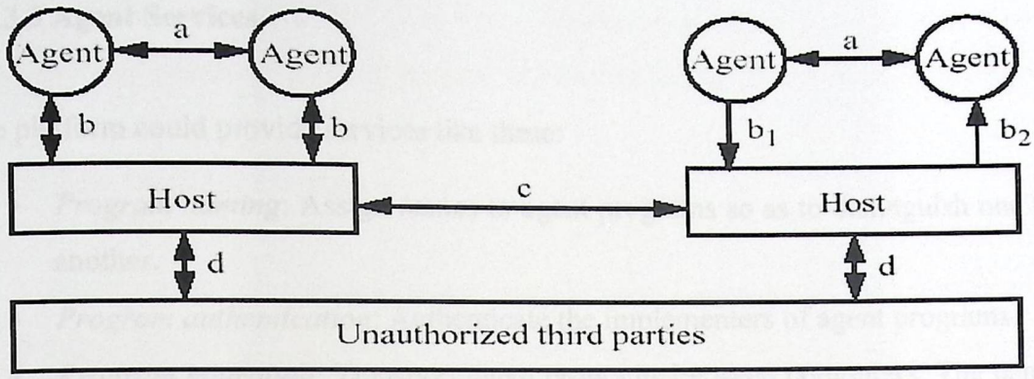


Figure 2.3 Security Issues in Mobile Agent Systems

2.2.3.2 Agent Platforms

Agents will be written in a variety of programming languages. This creates a need for a *common agent platform*, a language-independent means for agent interaction and mobile agent migration.

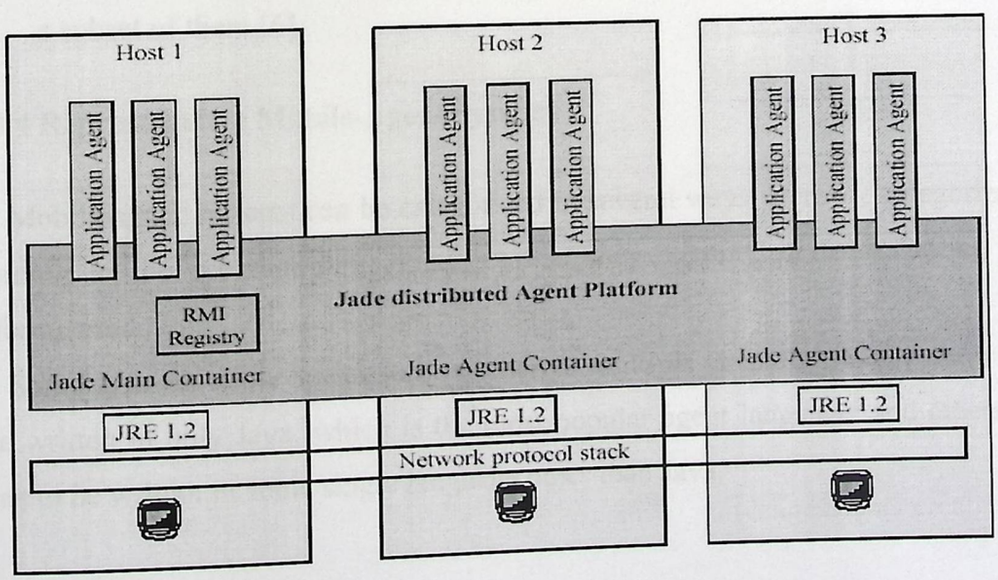


Figure 2.4 Agent Platform

2.2.3.3 Agent Services

The platform could provide services like these:

- *Program naming*: Assign names to agent programs so as to distinguish one from another.
- *Program authentication*: Authenticate the implementers of agent programs.
- *Program migration*: Transport agent programs between computers. The name of a program might be used to detect that it was already present at its destination.
- *Agent naming*: Assign names to agents so as to distinguish one from another.
- *Agent authentication*: Authenticate the authorities of agents.
- *Agent migration*: Transport agents between computers. The mobile agent's destination might be specified indirectly as the computer that hosts a specified stationary agent.
- *Agent creation*: Let agents create other agents locally and remotely. A new agent might have the authority of the existing agent and either the same permissions or a subset of them.[6]

2.2.3.4 Representative Mobile-Agent Systems

Mobile-agent systems can be categorized in several ways. Here we categorize them according to the programming languages that they support, or to according to the platforms used.

Some systems allow agents to be written in multiple languages; many allow agents to be written in only Java, which is the most popular agent language; and others allow agents to be written in some single language other than Java.

Some several ways to categorize Mobile-agent systems according to the programming languages that they support, or to according to the platforms used are:-

1) Multiple-language systems

A few systems attempt to support mobile agents outside of the context of any specific programming language, and indeed support more than one programming language.

Examples :-

1. **Ara**: Ara supports agents written in Tcl, Java and C/C++. The C/C++ agents are compiled into an efficient interpreted bytecode called MACE.

2. **D'Agents:** The mobile-agent system D'Agents, which was once known as Agent Tcl, supports agents written in Tcl, Java and Scheme, as well as stationary agents written in C and C++.

3. **Tacoma:** Tacoma supports agents written in C, C++, ML, Perl, Python and several other languages. Unlike Ara and D'Agents, Tacoma does not provide automatic state-capture facilities.[7]

2.2.4 Fault Tolerance (FT)

2.2.4.1 What is fault-tolerance?

Fault tolerance (FT) is the ability of a system to respond gracefully to an unexpected hardware or software failure. There are many levels of fault tolerance, the

lowest being the ability to continue operation in the event of a power failure. Many fault-tolerant computer systems mirror all operations that is, every operation is performed on two or more duplicate systems, so if one fails the other can take over.

2.2.4.2 Basic Concepts

Fault Tolerance is closely related to the notion of "Dependability". In Distributed Systems, this is characterized under a number of headings:

- *Availability* – the system is ready to be used immediately.
- *Reliability* – the system can run continuously without failure.
- *Safety* – if a system fails, nothing catastrophic will happen.
- *Maintainability* – when a system fails, it can be repaired easily and quickly (and, sometimes, without its users noticing the failure).

2.2.4.3 Types of Fault

There are three main types of fault:

- Transient Fault: appears once, and then disappears.
- Intermittent Fault: occurs, vanishes, and reappears; but: follows no real pattern (worst kind).
- Permanent Fault: once it occurs, only the replacement/repair of a faulty component will allow the DS to function normally.[8]

2.2.4.4 Classification of Failure Models

Table 2.1:-classification of fault tolerance.

Type of failure	*Description*
Crash failure	A server halts, but is working correctly until it halts.
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests. - A server fails to receive incoming messages. - A server fails to send outgoing messages.
Timing failure	A server's response lies outside the specified time interval.
Response failure *Value failure* *State transition failure*	The server's response is incorrect. - The value of the response is wrong. - The server deviates from the correct flow of control.
Arbitrary failure	A server may produce arbitrary responses at arbitrary times.

2.2.4.5 Scope:

- Fault Tolerance provides functionality to improve system reliability and availability.
- Fault Tolerance has plug-ins for commercial and third party devices.
- Fault Tolerance acquires local data and network wide data via sensors.
- Interface to monitoring for reporting.
- Diagnostic access through sensor system.

2.2.5 Replication

2.2.5.1 What is Data Replication?

Data replication is the ability to replicate an existing file or file structure from one node to another. Depending on your data replication tool, you have the ability to replicate data either one-way or two-way. Most data replication tools give the ability to

schedule when replication should occur and what data should replicate. Data replication can be a solution that could integrate within your current environment and data processes.

2.2.5.2 Data Replication Benefits

Depending on your needs and the data replication tool, you can hope to gain several benefits from using data replication in your environment. Some of the benefits may include are:

- Reduced equipment costs
- Depending on how you setup replication, restores of files can be quicker
- Can provide fail over capabilities almost instantaneous
- Easily implemented within your current environment
- Ease of use
- Provides exact multiple copies of your data.[10]

2.2.5.3 Replication Techniques

The most popular replication techniques, which allow maintaining consistency among the server replicas, can be summarized below.

1. Active Replication: Active replication, also called state-machine replication. Active replication has two limitations: the processing redundancy leads to increased resource usage, and the need for deterministic execution of the client request. Multiple threads, for instance, generally lead to non-determinism, as the thread scheduling is not deterministic.

2. Passive Replication: In contrast to active replication, in passive replication (also called *primary-backup*) only one replica, the *primary*, executes the client request. The backup replicas do not directly communicate with the client; rather, they only communicate with the primary. As only the primary executes the client request, deterministic execution is not needed.

3. Semi-Passive Replication: Similarly to passive replication, semi-passive replication, processes the client request on one server replica (called the *primary*), if no failures occur, and then updates the backup replicas.

4. Semi-Active Replication: Semi-active replication attempts to combine the advantages of passive replication (i.e., handling of non-determinism) and active replication (fast response time despite failures, i.e., fast fail-over). [10]

2.3 Java Agent Development Environment (JADE)

JADE is a middleware that facilitates the development of multi-agent systems. It includes:

- **A runtime environment** where JADE agents can "live" and that must be active on a given host before one or more agents can be executed on that host.
- **A library of classes** that programmers have to/can use (directly or by specializing them) to develop their agents.
- **A suite of graphical tools** that allows administrating and monitoring the activity of running agents.

2.3.1 Containers and Platforms

Each running instance of the JADE runtime environment is called a Container as it can contain several agents. The set of active containers is called a Platform. A single special Main container must always be active in a platform and all other containers register with it as soon as they start. It follows that the first container to start in a platform must be a main container while all other containers must be "normal" (i.e. on-main) containers and must "be told" where to find (host and port) their main container (i.e. the main container to register with).

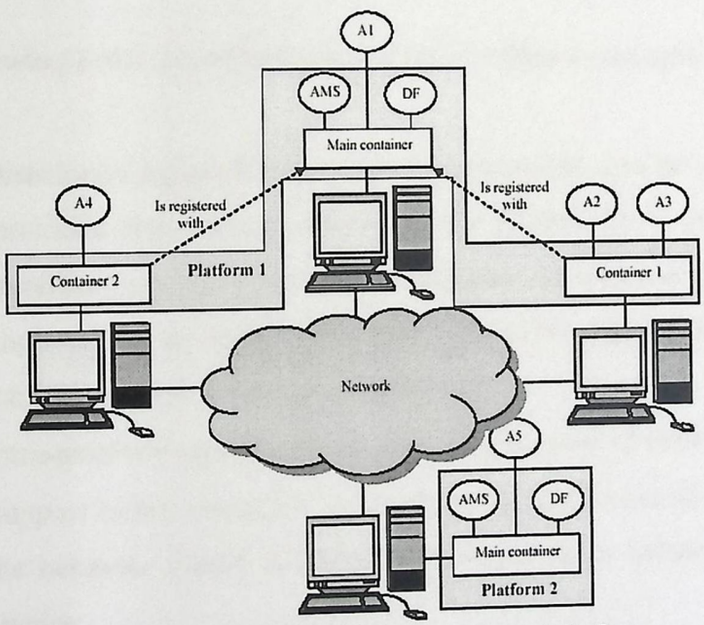


Figure 2.5 Containers and platform

AMS and DF:

Besides the ability of accepting registrations from other containers, a main container differs from normal containers as it holds two special agents (automatically started when the main container is launched).

The AMS (Agent Management System) that provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS).

The DF (Directory Facilitator) that provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

2.3.2 JADE Features

The following is the list of features that JADE offers to the agent programmer:

- Distributed agent platform. The agent platform can be split among several hosts (provided they can be connected via RMI4). Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as Java threads and live within *Agent Containers* that provide the runtime support to the agent execution.
- Intra-platform agent mobility, including transfer of both the state and the code.
- Support to the execution of multiple, parallel and concurrent agent activities via the behavior model. JADE schedules the agent behaviors in a non-preemptive fashion.
- FIPA-compliant Agent Platform, which includes the *AMS* (*Agent Management System*), the *DF* (*Directory Facilitator*), and the *ACC* (*Agent Communication Channel*). All these three components are automatically activated at the agent platform start-up.
- FIPA-compliant naming service: at start-up agents obtain their GUID (Globally Unique from the platform.[11]

2.3.3 FIPA (Foundation for Intelligent Physical Agents)

FIPA is a non-profit making organization that was established in 1996 with the aim of promoting specifications for facilitating interoperability between agent systems. The model proposed by FIPA consists of concrete specifications enabling interoperability, based on a high-level abstract architecture.

2.3.3.1 FIPA-Agent-Management ontology

Every class implementing a concept of the fipa-agent-management ontology is a simple collection of attributes, with public methods to read and write them, according to the frame based model that represents FIPA fipa-agent-management ontology concepts. The following convention has been used. For each attribute of the class, named attrName and of type attrType, two cases are possible:

1) The attribute type is a single value and
2) The attribute type is a set or a sequence of values.

2.3.4 The ACL language

Messages exchanged by JADE agents have a format specified by the ACL language defined by the FIPA (http://www.fipa.org) international standard for agent interoperability. This format comprises a number of fields and in particular:

- The sender of the message.
- The list of receivers.

- The communicative intention (also called "performative") indicating what the sender intends to achieve by sending the message. The performative can be REQUEST, if the sender wants the receiver to perform an action, INFORM, if the sender wants the receiver to be aware a fact, QUERY_IF, if the sender wants to know whether or not a given condition holds, CFP (call for proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, if the sender and receiver are engaged in a negotiation, and more.

2.3.4.1 ACL message

The idea of creating a migration using ACL messages came from FIPA's specification regarding mobility, where this type of migration is proposed. However, as mentioned above, this specification only gives a general outline of the ontology's, the protocol, and the life cycle of a mobile agent. It has not been updated due to the lack of implementations and has become obsolete within the FIPA specifications. For these reasons, we have found that there is a need to propose extensions to the specification to cover situations that it does not deal with. These extensions have been approved by the managers at JADE and the FIPA technical committee.

The design for a migration using ACL means that transmission of mobile agents between two agencies will be carried out using the message system between agents. In other words, the agent (both the code forming it as well as the state that it is in) will travel as the content of a message between two agents. Specifically, the agent will travel in the message between the AMS agents of each of the agencies involved in a migration. Because the agencies have mechanisms for sending and receiving messages, using a parallel transmission protocol is not necessary. This is an advantage in interoperability

terms and enables agents to be transmitted using the various message transport protocols (HTTP, IIOP, SMTP, etc). Furthermore, this is achieved in a totally transparent way.

The first logical step in this process is to design the ontology and the protocol that will be used in the exchange of messages between the two agencies. This protocol has the movement of the agent as its final purpose. Defining an ontology basically consists of defining the elements that will form the content of an ACL message to give a common interface between the two parties when extracting the information of the message.

The two possible migration models that are proposed in the initial FIPA specification deal with one migration directed by the agent and another directed by the agency. In implementation, Must have decided to adopt the migration directed by the agency, which is more robust, as it enables the agency to decide which migration requests are accepted and which are not.

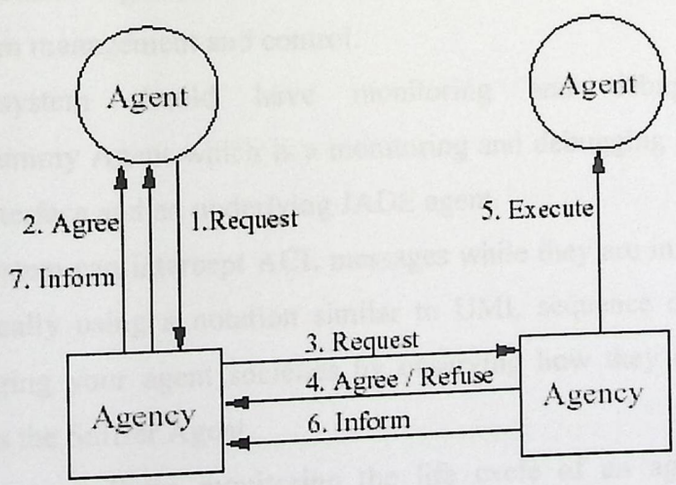


Figure 2.6 Exchange of ACL messages in the migration protocol

2.4 Software Requirements Specification

2.4.1 System Definition

Our system is about simulation for fault tolerance for a mobile agent using a replication technique; this means that the system should recover the faults that agents face by a replication-based technique. Replication may occur in every container the agent visit or only in the main container, for this simulation we will use a JADE simulator.

2.4.2 Functional Requirements

The system to be designed will have the following:

1. The system will provide a good GUI, it is available in JADE using Remote Management Agent, *RMA* for short, which is acting as a graphical console for platform management and control.
2. The system should have monitoring and debugging tools such as The Dummy Agent which is a monitoring and debugging tool, made of a graphical user interface and an underlying JADE agent.
3. The system can intercept ACL messages while they are in flight, and displays them graphically using a notation similar to UML sequence diagrams. It is useful for debugging your agent societies by observing how they exchange ACL messages such as the Sniffer Agent.
4. The system allows monitoring the life cycle of an agent, its exchanged ACL messages and the behaviors in execution. Such as the Introspector Agent.

5. The system allows setting at runtime logging information, such as the log level, for both JADE and application specific classes that use Java Logging such as The LogManagerAgent.
6. The system has agent acting as a bidirectional gateway between a JADE platform and an ordinary TCP/IP connection. Such as The SocketProxyAgent
7. The system should be able to perform the basic operation on agents. These are: kill, create, clone, migrate and suspend agents.
8. The system should be able to make some processing on containers, such as: create, kill containers.
9. The system should be able to shutdown the platform normally.
10. The system should be able to connect more than one platform.[14]

2.4.3 Non-Functional Requirements

The system will have following nonfunctional requirements:

1. Speed: the system shall provide and retrieve the request from agent in acceptable speed.
2. Ease of use: the user interface should help the user to choose services in an easy way, and the system shall have help.
3. The migration of agent should be done easily and without loss of information.
4. System Reliability: the system should work in any environment and should be recoverable.
5. Portability: the system user shall have the ability to transfer system data.
6. Delivery date: the product should be delivered before 31/05/2006.

Chapter Three
Architecture Design

Overview
System Objectives
System Block Diagram
System Architectures
How Does the System Work?
System Modeling

Chapter Three

Architectural Design

3.1 Overview

This chapter will focus on the System objectives, system block diagram, system architecture, system modeling, and fault tolerance approach which will be used.

3.2 System Objectives

The main objective that the system aims is to tolerate the faults in the environment where several crash possibilities could occur. There are some other minor objectives related to the main objective such as:

- FT should maintain the system performance in case of failure
- FT should not have single point of failure.
- Failure of the system should not stop system.
- FT should be distributed (for scalability and tolerance).
- FT should optimize message rate and volume in system → hierarchical.
- All FT activities have to be traceable through monitoring.
- Upgradeable.
- Standardized interfaces to other systems.
- FT keeps current (distributed) system status, with history.

3.3 System Block Diagram

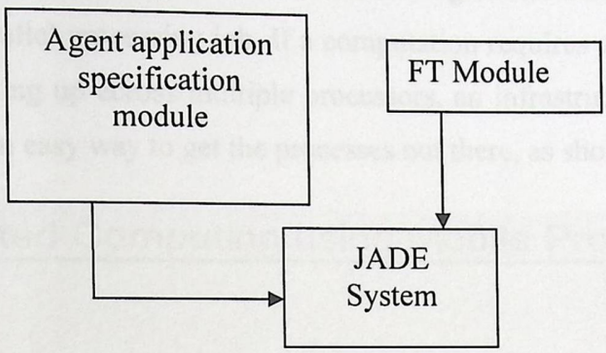


Figure 3.1 Block diagram for the system

3.3.1 The Agent Application Specification

- This module will concern with the setting up for the system, such as: giving agents their services and tasks to be performed. In a case of static approach, this module will specify the itinerary. One of the tasks that this mobile agent could do is Data collection as shown in figure 3.2:-

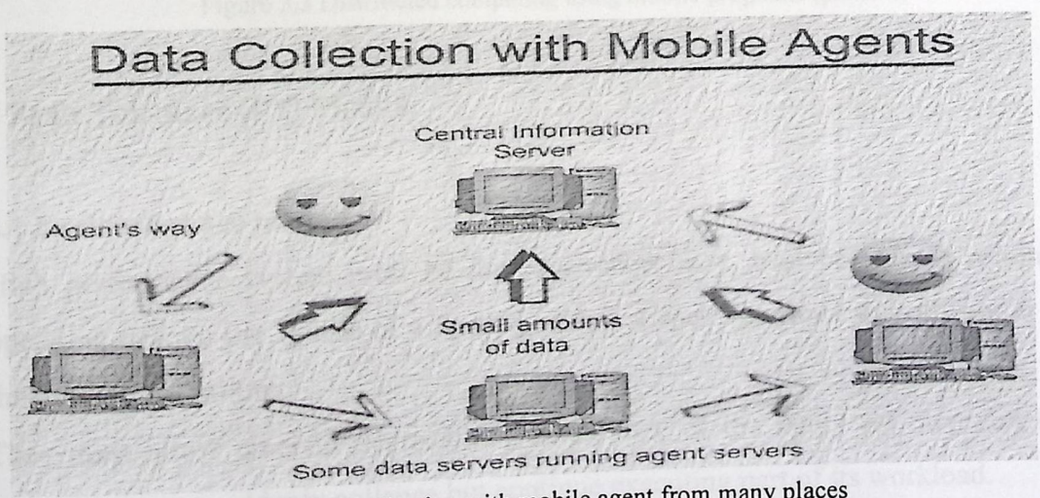


Figure 3.2 Data collection with mobile agent from many places

- Parallel processing, given that mobile agents can move from node to node and can spawn subagents; one potential use of mobile agent technology is as a way to administer a parallel processing job. If a computation requires so much CPU time as to require breaking up across multiple processors, an infrastructure of mobile agent hosts could be an easy way to get the processes out there, as shown in figure 3.3 :-

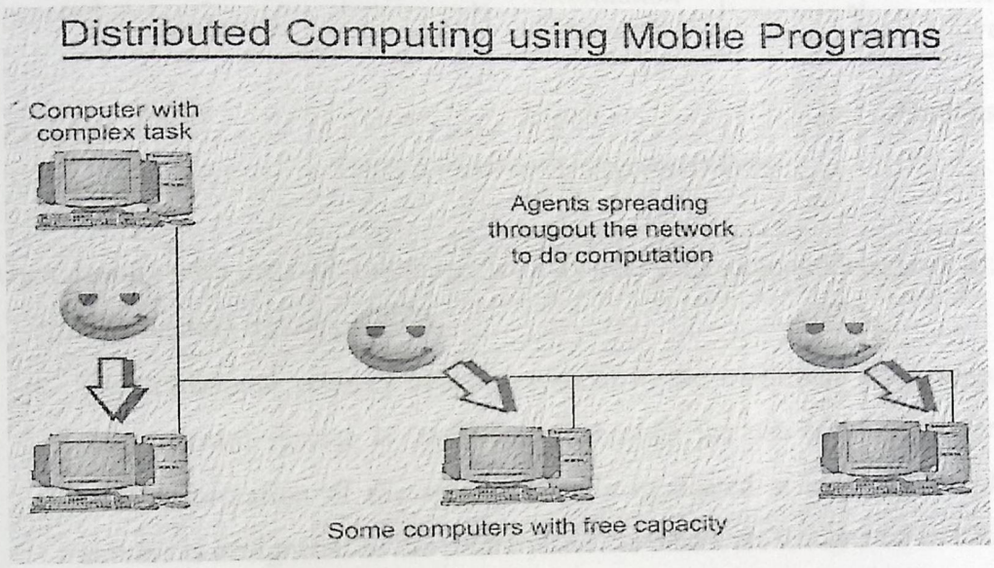


Figure 3.3 Distributed computing using mobile programs (parallel)

3.3.2 Fault Tolerance (FT) Module

3.3.2.1 Fault tolerance module

- High performance system must be fault tolerance: they must be able to continue operating despite the failure of a limited subset of their hardware or software or in the presence of failure.
- They must allow graceful degradation: as the size of the faulty set increases the system must not suddenly collapse but continue executing part of its workload.

Fault ⟶ errors ⟶ failures

- Fault is a physical defect, imperfection, or flow that occurs within some hardware or software component, a fault can be caused by specification mistakes, implementation mistakes, component defects or external disturbance (environment effects).
- An error is the manifestation of a fault.

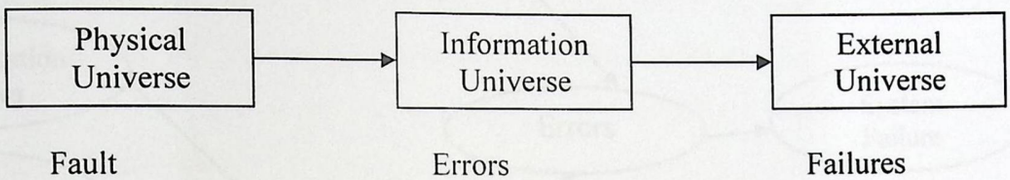


Figure 3.4 Cause-and-effect relationships

When dealing with the fault tolerance, we may deal with some models, such as:-

1. Primary backup models:
   - The traditional primary backup protocol in the traditional back up protocol, a fault-tolerant service is implemented through the use of multiple servers. The state information of the service is fully replicated at each server. One of the servers is designated as the primary and the others are designated as backups.
   - Active clients' primary-backup model.
   - The active clients' primary-backup protocol extends the traditional primary-backup model, each client maintains an ordered list of servers and uses it to detect faulty servers and elect new primary server. They argue that their AC n tolerates three types of failures: crash failures, send-omission failures and receive- omission failures of servers and clients and using minimal replication.

2. Replicated agents with voting: replication and voting can be used to mask the effects of executing an agent on a faulty processor. However, this needs the authentication of the votes since faulty processors that do not execute agents may proof and cast false votes.[17]

3.3.2.2 The cause-and-effect relationship is:-

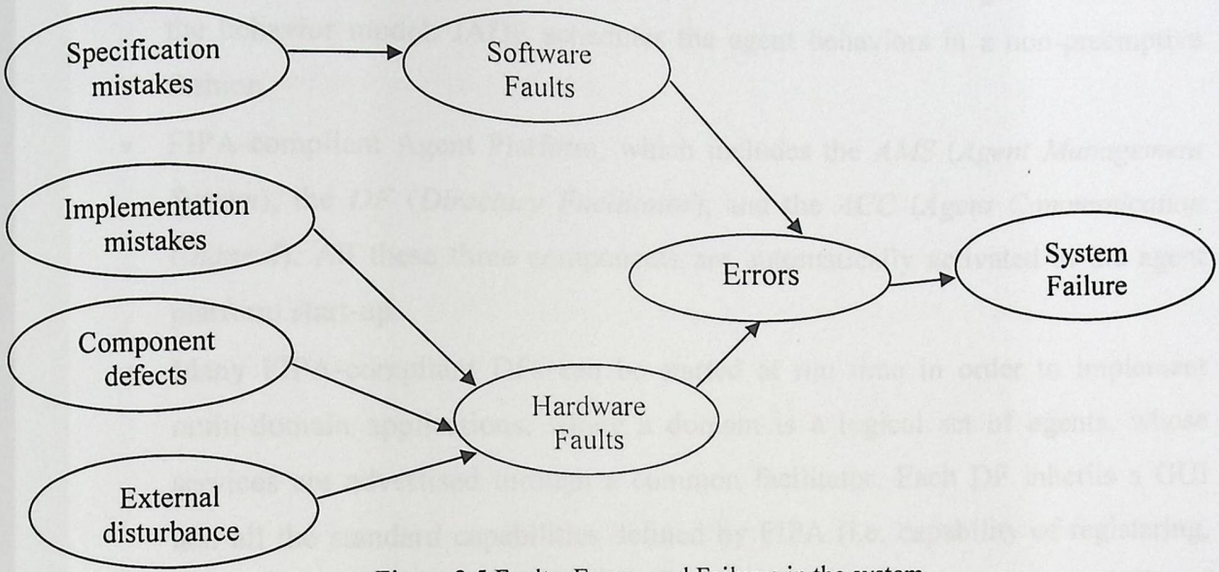


Figure 3.5 Faults, Errors, and Failures in the system

3.3.3 JADE System

3.3.3.1 JADE Features

The following is the list of features that JADE offers to the agent programmer:

- Distributed agent platform. The agent platform can be split among several hosts (provided they can be connected via RMI). Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as Java threads and live within *Agent Containers* that provide the runtime support to the agent execution.
- Graphical user interface to manage several agents and agent containers from a remote host.
- Debugging tools to help in developing multi agents applications based on JADE.

- Intra-platform agent mobility, including transfer of both the state and the code .
- Support to the execution of multiple, parallel and concurrent agent activities via the behavior model. JADE schedules the agent behaviors in a non-preemptive fashion.
- FIPA-compliant Agent Platform, which includes the *AMS* (*Agent Management System*), the *DF* (*Directory Facilitator*), and the *ACC* (*Agent Communication Channel*). All these three components are automatically activated at the agent platform start-up.
- Many FIPA-compliant DFs can be started at run time in order to implement multi-domain applications, where a domain is a logical set of agents, whose services are advertised through a common facilitator. Each DF inherits a GUI and all the standard capabilities defined by FIPA (i.e. capability of registering, deregistering, modifying and searching for agent descriptions; and capability of federating within a network of DF's).
- Efficient transport of ACL messages inside the same agent platform. In fact, messages are transferred encoded as Java objects, rather than strings, in order to avoid marshalling and un marshalling procedures. When crossing platform boundaries, the message is automatically converted to/from the FIPA compliant syntax, encoding, and transport protocol. This conversion is transparent to the agent implementers that only need to deal with Java objects.
- Library of FIPA interaction protocols ready to be used.
- Automatic registration and deregistration of agents with the AMS.
- FIPA-compliant naming service: at start-up agents obtain their GUID (Globally Unique from the platform.
- Support for application-defined content languages and ontologies.
- In Process Interface to allow external applications to launch autonomous agents.[1]

3.3.3.2 Why using JADE

There are many reasons that justify the use of JADE (Java Agent Development Framework), some of these reasons are:-

- JADE allows you to easily build real-world business models without being constrained by the mechanics of storing and accessing data. With JADE, you can store, retrieve and rapidly traverse complex, interconnected object structures as though they are always in memory.
- JADE entirely provides moving objects to and from a database (often referred to as *impedance mismatch*).
- Coupled with a flexible distributed processing model, JADE is ideal for solving complex problems.
- JADE makes it easier to build and run distributed systems that take advantage of modern, high-performance hardware platforms.
- JADE reduces the complexity of software development by delivering for Windows and Linux.
- Reducing Complexity, JADE simplifies programming by giving developers a single data model, seamless persistent data storage and a concise conceptual model for building OO, distributed, systems that manage complex, highly connected data.
- Increased Productivity, JADE's single language is compact, consistent and easy to learn. Combined with JADE's conceptually economic programming model, a proficient developer with a good working knowledge of OO and database concepts.

- With JADE, developers can focus their time and attention on system design and critical business details. Investment in key development staff is maximized as they are able to focus more on the higher-level problem space.

Developers can generally target more complex problems, so development is invariably faster and easier. Compared to other environments, JADE typically allows developers to deal with fewer concepts and higher-grain objects with fewer interfaces. These factors reduce complexity and improve productivity.

- Reduced Administration Costs, because JADE systems are robust. They deliver lower long-term costs for support and maintenance. JADE's robustness is derived from its single technology approach. One technology across all layers means fewer technology seams. Fewer seams make applications stronger and more robust.

As a result, systems built in JADE are resilient, scalable, and generally require less administration than other platforms.

- Interoperability, JADE provides several facilities to interoperate and co-exist with other technologies, the most important of which today is Web Services (XML, WSDL, and SOAP). Web Services are a set of technology standards rapidly emerging as the leading model for encapsulating business processes and sharing them with other systems. JADE's Web Services support provides a seamless interface for JADE developers to build distributed JADE applications that interoperate with other platforms (for example, .NET, J2EE and WebSphere).
- High Performance, the JADE platform can scale from single-server systems to large multi-server systems running critical applications with thousands of users.

- JADE is an open source environment.
- FIPA-compliant Agent Platform, which includes the AMS (Agent Management System), the DF (Directory Facilitator), and the ACC (Agent Communication Channel). All these three components are automatically activated at the agent platform start-up.[2]

As we are working on existing network environment, then we will follow the same architecture used there with all our minor and major changes to the platform in order to achieve our goal.

JADE distributed architecture relies on a special node, named *Main Container*, to coordinate all other nodes and keep together the whole platform. Though most JADE operations are decentralized, there are some essential features that are supported only by the Main Container.

These features are:

- Managing the Container Table (i.e. the set of all the nodes that compose the distributed platform).
- Managing the Global Agent Descriptor Table.
- Managing the MTP table.
- Hosting the platform AMS agent.
- Hosting the platform Default DF agent.

If the Main Container terminates or otherwise becomes unavailable to the other platform containers, all the above features become unavailable and this severely hampers platform operations. To keep JADE fully operational even in the event of a

failure of the Main-Container, support for Main Container replication has been introduced. Using this support, it is possible to start any number of Main Container nodes a "*master*" main container actually holding the AMS, and a number of "*backup*" main containers, which will arrange themselves in a logical ring so that whenever one of them fails; the others will notice and act accordingly.

The Remote Monitoring Agent (RMA) allows controlling the life cycle of the agent platform and of all the registered agents. The distributed architecture of JADE allows also remote controlling, where the GUI is used to control the execution of agents and their life cycle from a remote host.

The Agent Management System (AMS) is the agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. Each agent must register with an AMS in order to get a valid AID.

The Directory Facilitator (DF) is the agent who provides the default yellow page service in the platform. The Message Transport System, also called Agent Communication Channel (ACC), is the software component controlling all the exchange of messages within the platform, including messages to/from remote platforms.

JADE fully complies with this reference architecture and when a JADE platform is launched, the AMS and DF are immediately created and the ACC module is set to allow message communication. The agent platform can be split on several hosts.

Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is a basic container of agents that provides a complete

failure of the Main-Container, support for Main Container replication has been introduced. Using this support, it is possible to start any number of Main Container nodes a "*master*" main container actually holding the AMS, and a number of "*backup*" main containers, which will arrange themselves in a logical ring so that whenever one of them fails; the others will notice and act accordingly.

The Remote Monitoring Agent (RMA) allows controlling the life cycle of the agent platform and of all the registered agents. The distributed architecture of JADE allows also remote controlling, where the GUI is used to control the execution of agents and their life cycle from a remote host.

The Agent Management System (AMS) is the agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. Each agent must register with an AMS in order to get a valid AID.

The Directory Facilitator (DF) is the agent who provides the default yellow page service in the platform. The Message Transport System, also called Agent Communication Channel (ACC), is the software component controlling all the exchange of messages within the platform, including messages to/from remote platforms.

JADE fully complies with this reference architecture and when a JADE platform is launched, the AMS and DF are immediately created and the ACC module is set to allow message communication. The agent platform can be split on several hosts.

Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is a basic container of agents that provides a complete

run time environment for agent execution and allows several agents to concurrently execute on the same host.

The main-container, or front-end, is the agent container where the AMS and DF lives and where the RMI registry, that is used internally by JADE, is created. The other agent containers, instead, connect to the main container and provide a complete run-time environment for the execution of any set of JADE agents.

According to the FIPA specifications, DF and AMS agents communicate by using the FIPA-SL0 content language, the fipa-agent-management ontology, and the fipa-request interaction protocol.

JADE provides compliant implementations for all these components:

- The SL-0 content language. Automatic capability of using this language can be added to any agent.
- Concepts of the ontology (apart from Agent Identifier. The FIPAManagementOntology class defines the vocabulary with all the constant symbols of the ontology. Automatic capability of using this ontology can be added to any agent.
- Finally, the fipa-request interaction protocol.

3.3.4 Output Module

Here, we will talk about the results we can achieve from our project; these results may be a list of all active agents, mobile agents, and containers either in the local or remote platforms within the same JADE system. It could be a list of some specific services collected from specific containers.

3.4 System Architecture

As we are working on existing network environment, then we will follow the same architecture used there with all our minor and major changes to the platform in order to achieve our goal.

JADE distributed architecture relies on a special node, named *Main Container*, to coordinate all other nodes and keep together the whole platform. Though most JADE operations are decentralized, there are some essential features that are supported only by the Main Container.

The Remote Monitoring Agent (RMA) allows controlling the life cycle of the agent platform and of all the registered agents. The distributed architecture of JADE allows also remote controlling, where the GUI is used to control the execution of agents and their life cycle from a remote host.

The standard model of an agent platform, as defined by FIPA, is represented in the following figure, Figure 3.6:-

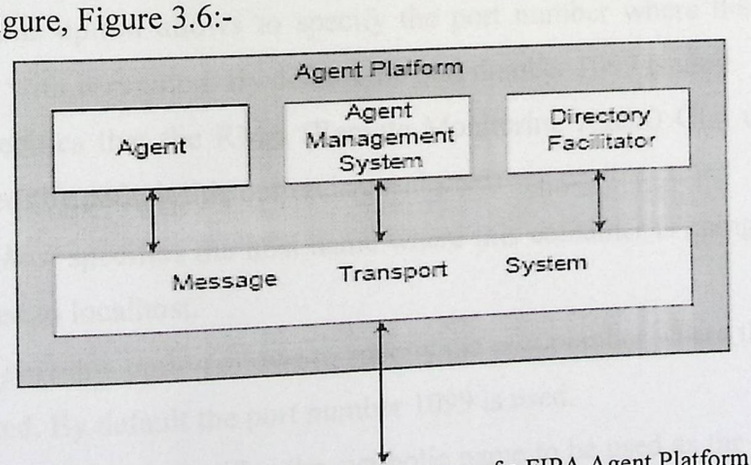


Figure 3.6 Reference architecture of a FIPA Agent Platform

3.4 How does the System Work

Our system use Fault Tolerance of Mobile Agents (Replication-based Technique) to tolerate faults in a case of crash of machines, platforms, and agents. The system will use JADE framework to simulate our work. the work mainly will be applied in an information retrieval application; where some specific information will be gathered and retrieved from several hosts. In case of failure, several techniques could be used to solve this degradation and loss of performance and reliability. Replication-based techniques would be one of the possible solutions to solve the problem of system failure. We will make use of some of the existing techniques in JADE Simulation, compare between them, analyze the results and may be propose our own technique. Then, this could be integrated within our current environment.

And there are some options we use it in JADE like :-

- - *host* specifies the host name where the main container to register with is running; its value is defaulted to localhost. This option can also be used when launching the main-container in order to override the value of localhost.
- - *port* this option allows to specify the port number where the main container to register with is running. By default the port number 1099 is used.
- -*gui* specifies that the RMA (Remote Monitoring Agent) GUI of JADE should be launched (by default this option is unselected)
- - *local-host* specifies the host name where this container is going to run; its value is defaulted to localhost.
- - *local-port* this option allows to specify the port number where this container can be contacted. By default the port number 1099 is used.
- - *name* this option specifies the symbolic name to be used as the platform name; this option will be considered only in the case of a main container; the default is to

generate a unique name from the values of the main container's host name and port number. This might result in non-unique agent names.

- - *container-name* this option specifies the symbolic name to be used as the name of this container.
- - *services* specifies a list of classes, implementing JADE kernel-level services, that must be loaded and activated during startup. Class names are to be separated by semicolon, and must be fully qualified. The special messaging and management services are always activated and need not be specified with this option. If this option is not given, by default the mobility and event notification services are started. [11]

The following table lists the services available with the standard distribution of JADE, as shown in following table:-

Table 3.1 JADE services [11]

Name	Service Class	Features
Messaging	jade.core.messaging.MessagingService	ACL message exchange and MTP management. **Activated automatically**
AgentManagement	jade.core.management.AgentManagementService	Basic management of agent lifecycle. Container and platform shutdown. RMA support. **Activated automatically**
AgentMobility	jade.core.mobility.AgentMobilityService	Agent mobility support. **Active by default**

- *-backupmain* specifies that this instance of JADE is a backup main container and, as such, that it must join with a main-container (by default this option is unselected)
- *-help* print on standard output this help information (by default this option is unselected).

3.5.1 JADE Tools

There are number of tools that we will use in JADE Simulator, these tools about the agent:-

1- **Dummy Agent**: A monitoring and debugging tool, made of a graphical user interface and an underlying JADE agent. Using the GUI it is possible to compose ACL messages and send them to other agents; it is also possible to display the list of all the ACL messages sent or received, completed with timestamp information in order to allow agent conversation recording and rehearsal.

2- **Sniffer:** The system can intercept ACL messages while they are in flight, and displays them graphically using a notation similar to UML sequence diagrams. It is useful for debugging your agent societies by observing how they exchange ACL messages.

3- **Remote Management Agent**: Which acting as a graphical console for platform management and control. A first instance of an RMA can be started with a command line option ("*-gui*"), but then more than one GUI can be activated.

4- **The Introspector Agent**: The system allows monitoring the life cycle of an agent, it's exchanged ACL messages and the behaviors in execution.

5- **The LogManagerAgent**: The system allows setting at runtime logging information, such as the log level, for both JADE and application specific classes that use Java Logging.

6- **The SocketProxyAgent**: The system has agent acting as a bidirectional gateway between a JADE platform and an ordinary TCP/IP connection.

The basic functions applied for each agent, mobile agent, are as follows:

1) Star New Agent →it create new agent.

2) Kill Agent → the agent will be killed.

3) Suspend Agent →the agent will be suspended for a specific time.

4) Resume Agent →retrieval and resume the agent that we suspend before.

5) Send Message →the agent will send message to other agent.

6) Migrate Agent →the agent will move from container to another, or from platform to another one.

7) Clone Agent → copy agent with all its services.

8) Load Agent →it will load an agent.

9) Save Agent →it will save the agent.

10) Freeze Agent →it will stop the agent work.

11) Thaw Agent →kill agent slowly.

3.5.2 Replication – based technique

Data replication is the ability to replicate an existing file or file structure from one node to another. Depending on your data replication tool, you have the ability to replicate data either one-way or two-way. Most data replication tools give you the ability to schedule when replication should occur and what data should replicate.

Data replication can be a solution that could integrate within your current environment and data processes. To keep JADE operational even in the event of a failure of the Main-Container, support for Main Container replication has been introduced. Using this support, it is possible to start any number of Main Container nodes, which will a range themselves in a logical ring so that whenever one of them fails; the others will notice and act accordingly.

Ordinary containers will then be able to connect to the platform through any of the active Main Container nodes; the different copies will evolve together using cross-

notification .As Figure 3.7 shows, without Main Container replication JADE platform has a star topology, end enabling Main Container replication turns the topology into a ring of starts as shown in Figure 3.7:-

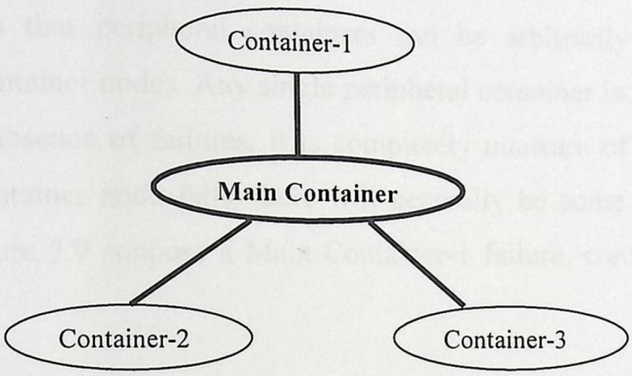


Fig 3.7 Star Topology.

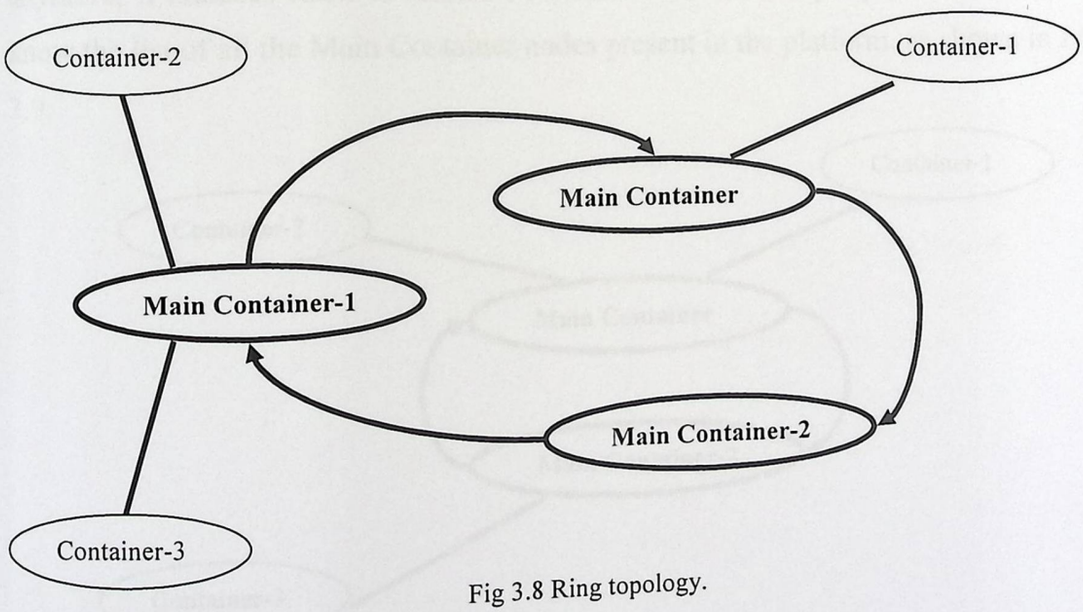
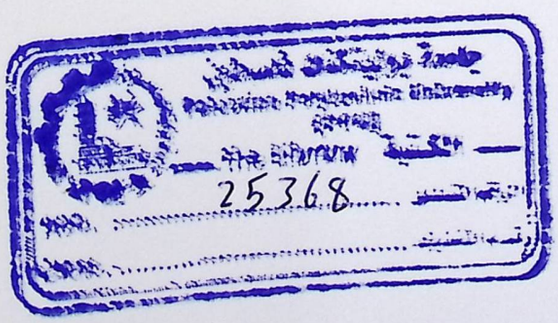


Fig 3.8 Ring topology.

In the fault-tolerant configuration shown on Figure 3.8 , three Main Container nodes are arranged in a ring, and each node is monitoring its neighbor: if the node Main



Container-1 fails, the node Main Container-2 will notice and inform all the other Main Container nodes (in this case just the Main Container one).

Then, a smaller ring will be rebuilt (with just two elements in the figure). The figure also shows that peripheral containers can be arbitrarily spread among the available Main Container nodes .Any single peripheral container is connected to exactly one node and in absence of failures, it is completely unaware of all the other copies When a Main Container node fails, there will generally be some orphaned peripheral containers (in Figure 3.9 suppose a Main Container-1 failure, container-3 will become an orphan).

When an orphan container detects that, its Main Container node is not available anymore, it attaches itself to another, for this to succeed, a peripheral container must know the list of all the Main Container nodes present in the platform, as shown in Figure 3.9:

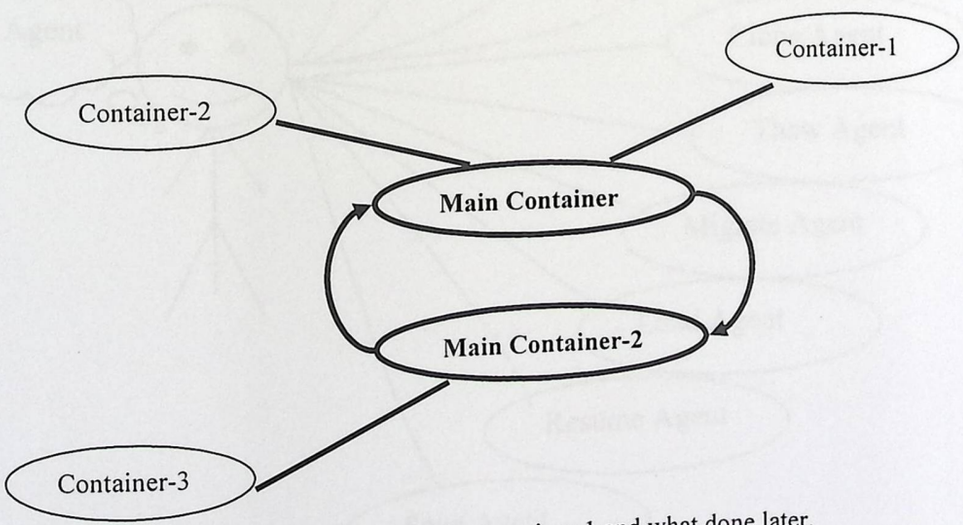


Figure 3.9 Failure on Main Container-1 and what done later.

3.6 System Modeling

In the system modeling, we will talk about the sequence diagram and the use-case; these will be shown in the coming figures.

3.6.1 Use- Case:-

3.6.1.1 Agent-to-Agent relationship

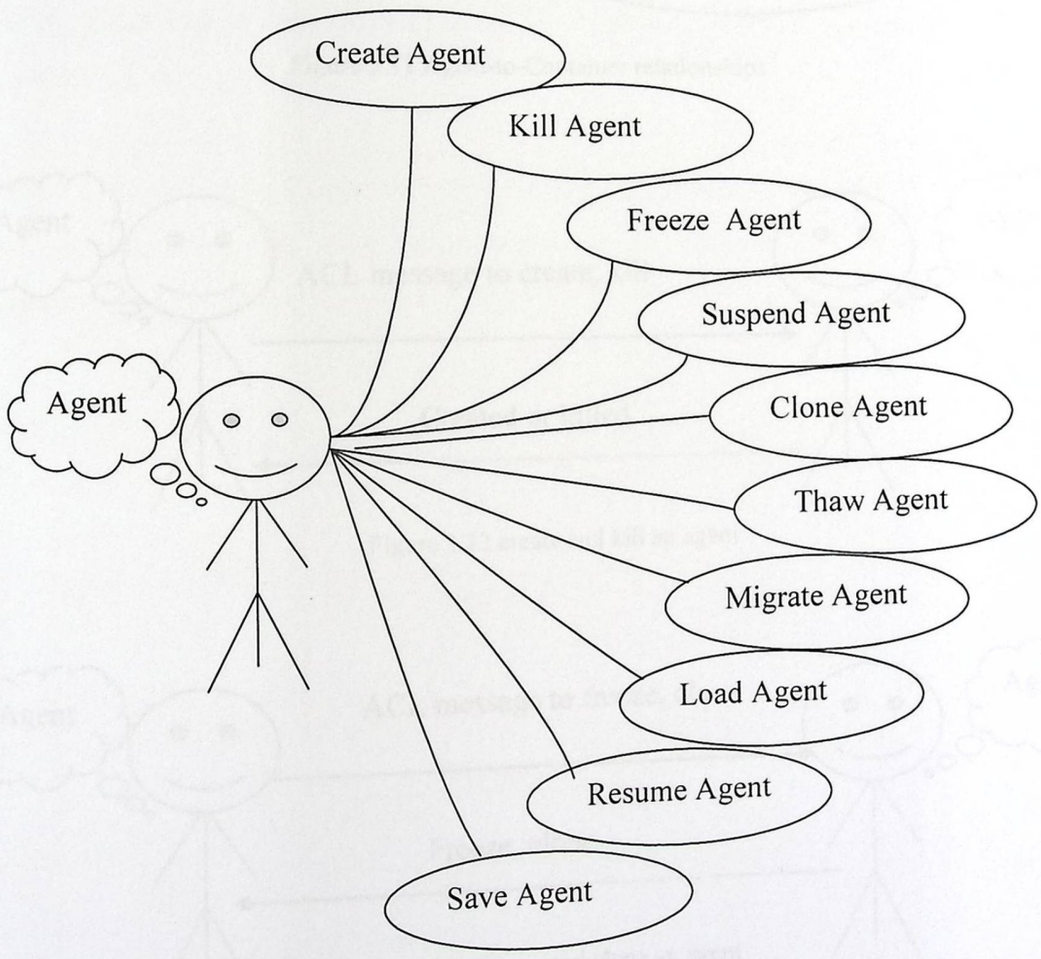


Figure 3.10 Agent-to-Agent relationships

3.6.1.2 Agent –to- Container relationship

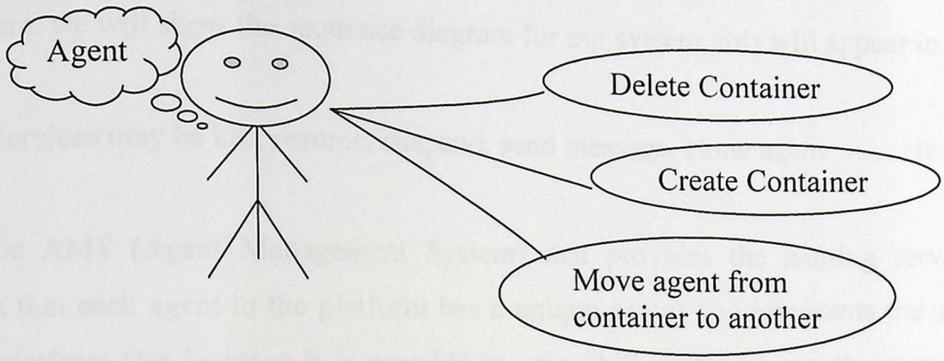


Figure 3.11 Agent-to-Container relationships

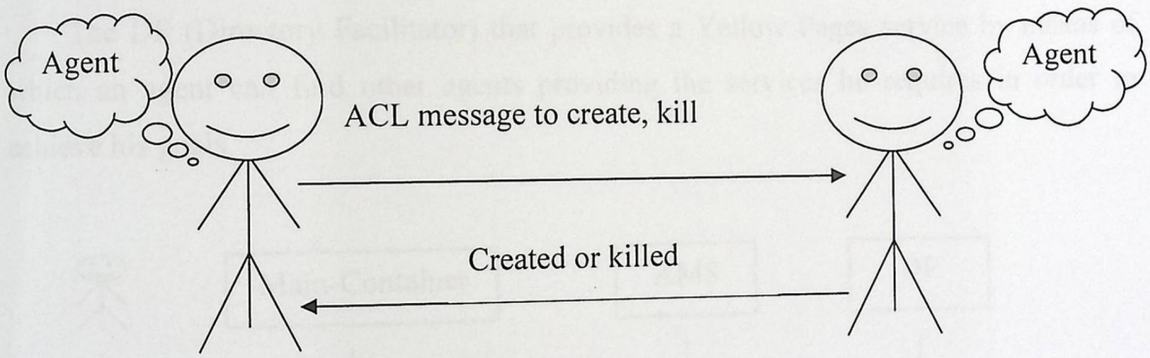


Figure 3.12 create and kill an agent

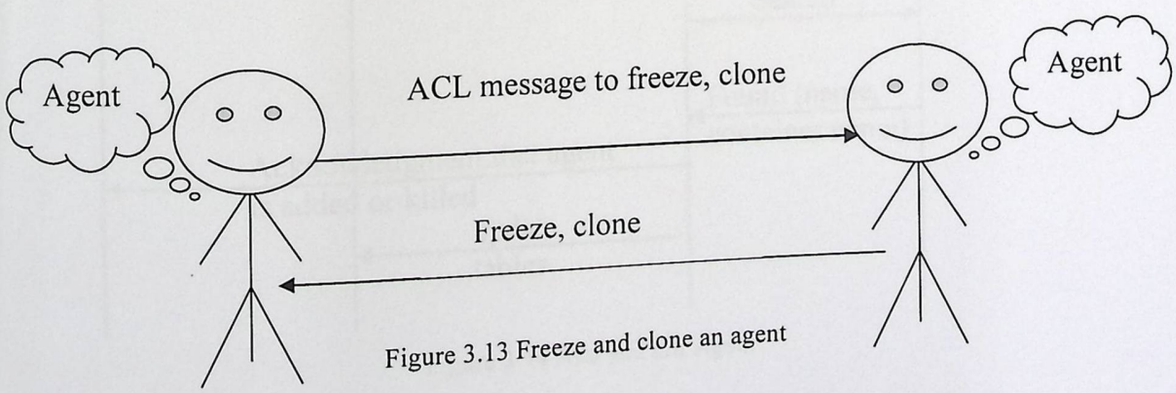


Figure 3.13 Freeze and clone an agent

3.6.2 Sequence Diagram

Here, we will show the sequence diagram for our system, this will appear in Figure 3.14:-

Agent services may be kill, resume, suspend, send message, clone agent .....etc

The AMS (Agent Management System) that provides the naming service (i.e. ensures that each agent in the platform has a unique name) and represents the authority in the platform (for instance it is possible to create/kill agents on remote containers by requesting that to the AMS).

The DF (Directory Facilitator) that provides a Yellow Pages service by means of which an agent can find other agents providing the services he requires in order to achieve his goals.

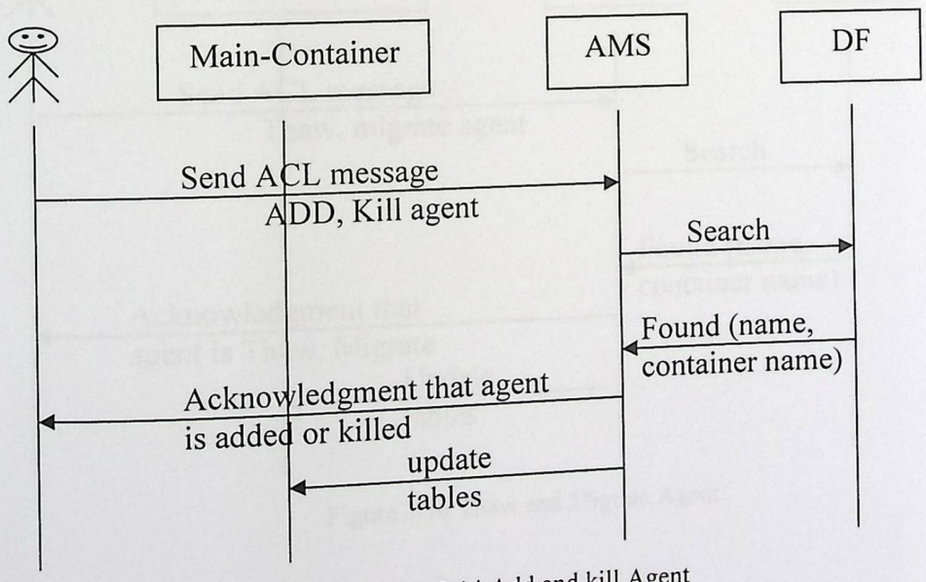


Figure 3.14 Add and kill Agent

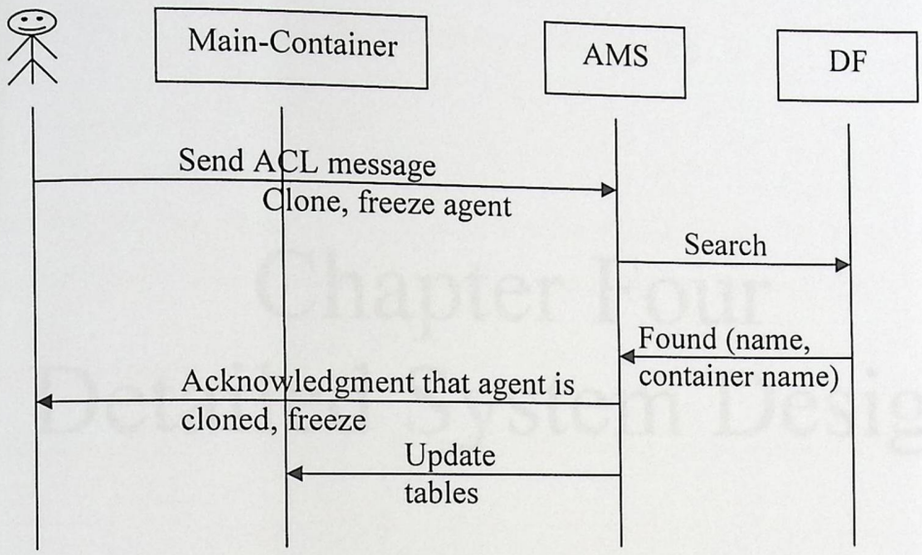


Figure 3.15 Clone and Freeze Agent

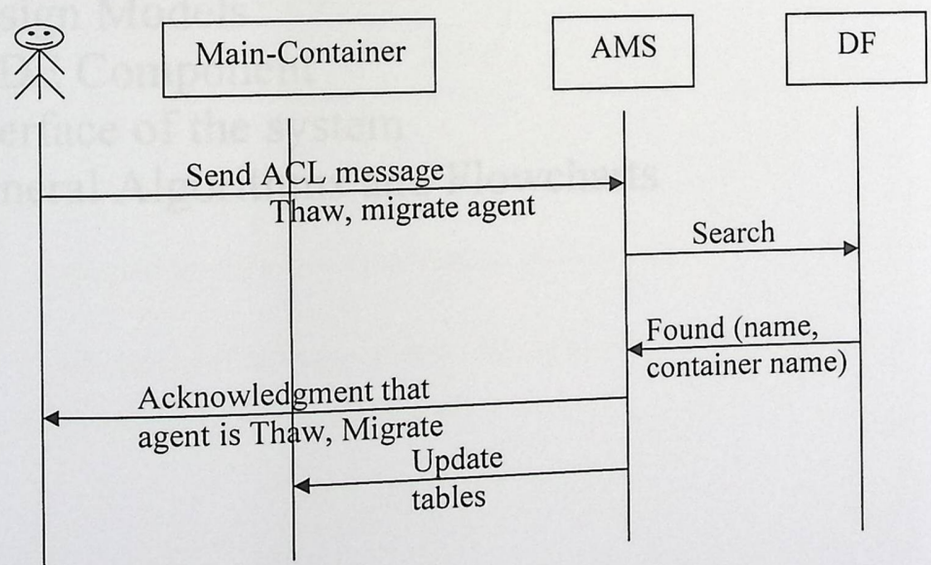


Figure 3.16 Thaw and Migrate Agent

Chapter Four
Detailed System Design

Overview
Design Models
JADE Component
Interface of the system
General Algorithms and Flowcharts

Chapter Four

Detailed System Design

4.1 Overview

In this chapter we are going to describe the whole design for the system, describe the project components in detail, the interface, and the flow chart for the system.

4.2 Design Models

In this section we will describe the general block diagram in detail.

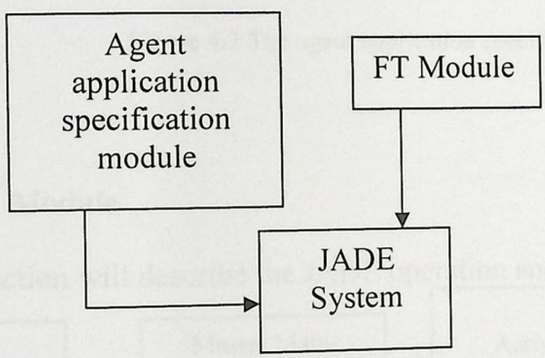


Figure 4.1 The general block diagram

4.2.1 The agent application specification module

This module will concern with the setting up for the system such as giving agent their services and task to be performed.

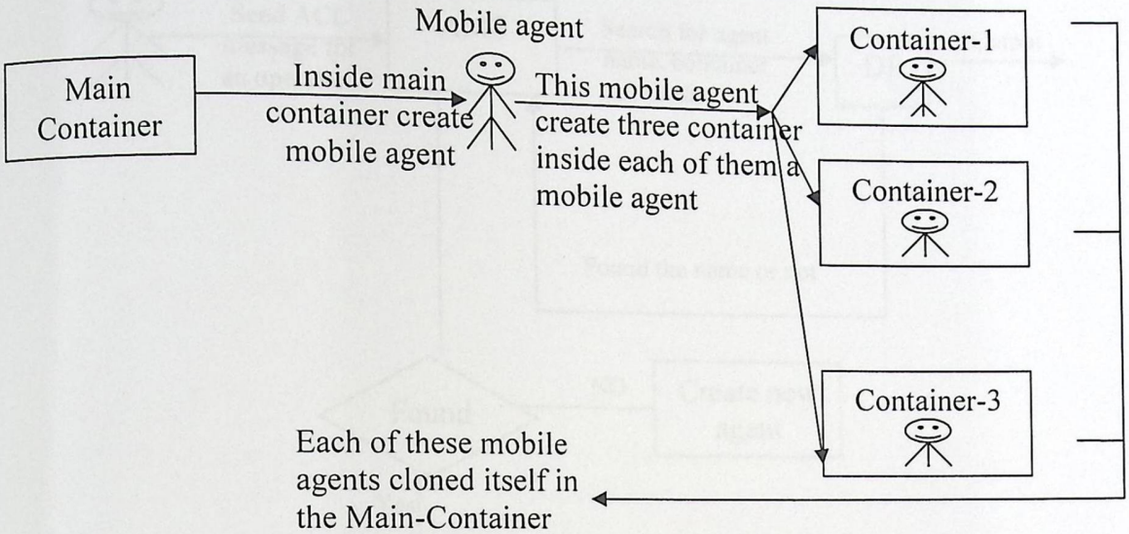


Figure 4.2 The agent application specification diagram

4.2.2 JADE Module

This section will describe the JADE operation and tasks

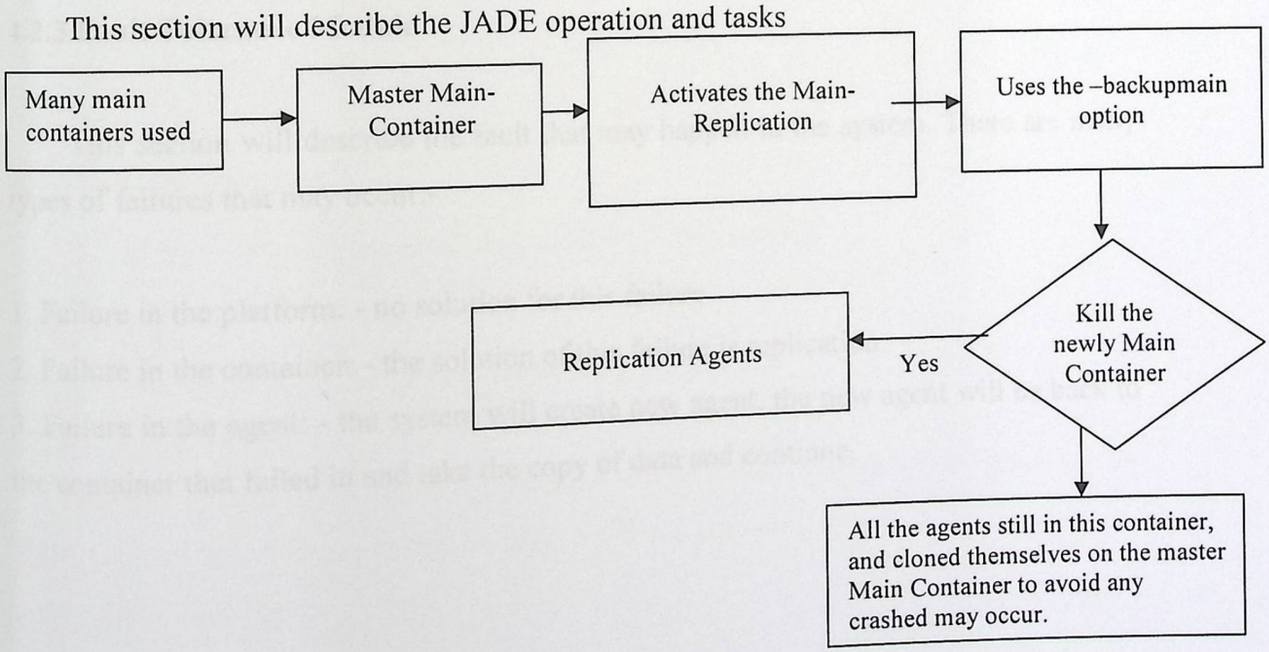


Figure 4.3 JADE fault tolerance

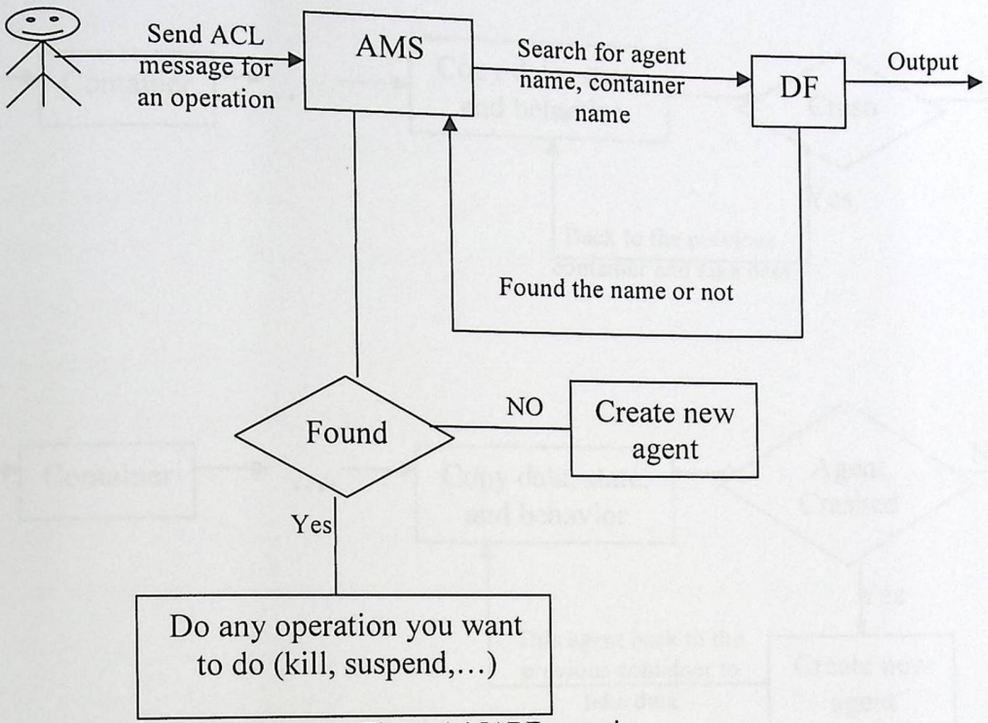


Figure 4.4 JADE operation

4.2.3 Fault Tolerance Module

This section will describe the fault that may happen in the system. There are many types of failures that may occur:-

1. Failure in the platform: - no solution for this failure
2. Failure in the container: - the solution of this failure is replication.
3. Failure in the agent: - the system will create new agent, the new agent will be back to the container that failed in and take the copy of data and continue.

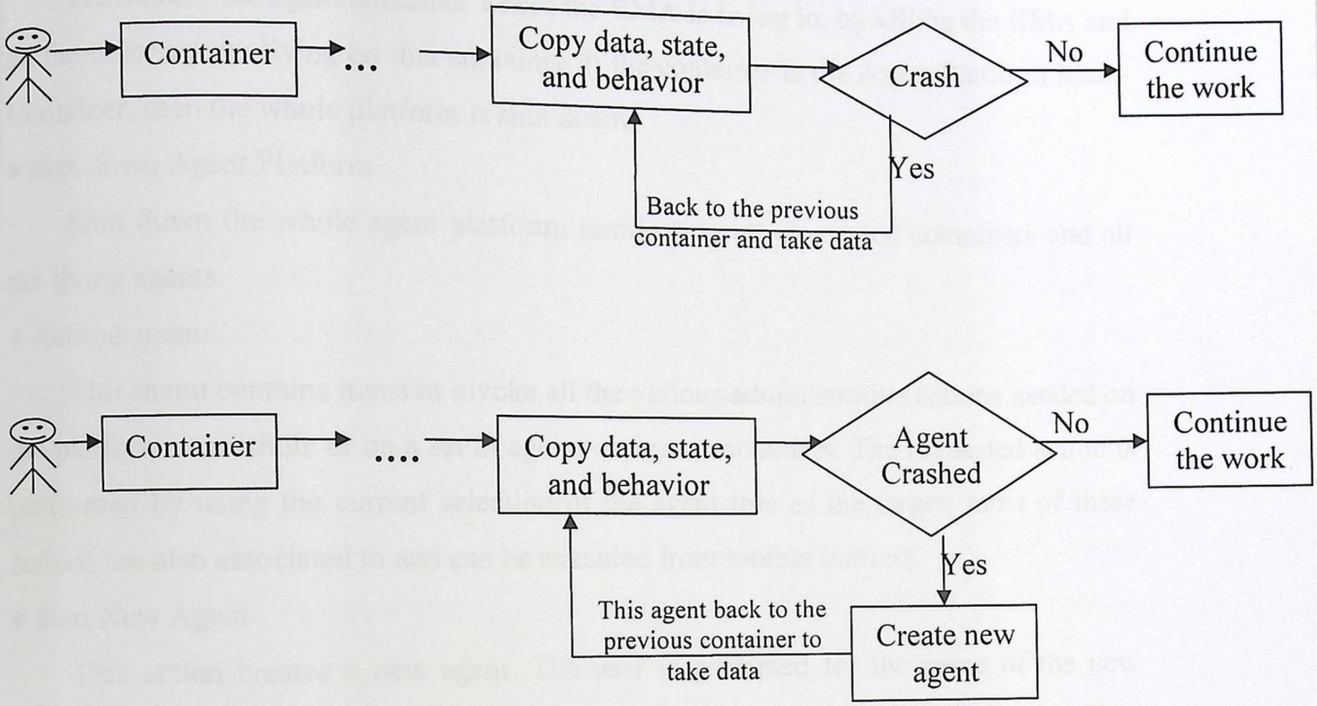


Figure 4.5 Fault Tolerance

4.3 JADE Components

There are many component that be used in JADE such as :

♦ File menu:

This menu contains the general commands to the RMA.

♦ Close RMA Agent

Terminates the RMA agent by invoking its doDelete() method. The closure of the RMA window has the same effect as invoking this command.

♦ Exit this Container

Terminates the agent container where the RMA is living in, by killing the RMA and all the other agents living on that container. If the container is the Agent Platform Main-Container, then the whole platform is shut down.

♦ shut down Agent Platform

Shut down the whole agent platform, terminating all connected containers and all the living agents.

♦ Actions menu:

This menu contains items to invoke all the various administrative actions needed on the platform as a whole or on a set of agents or agent containers. The requested action is performed by using the current selection of the agent tree as the target; most of these actions are also associated to and can be executed from toolbar buttons.

♦ Start New Agent

This action creates a new agent. The user is prompted for the name of the new agent and the name of the Java class the new agent is an instance of. Moreover, if an agent container is currently selected, the agent is created and started on that container; otherwise, the user can write the name of the container he wants the agent to start on. If no container is specified, the agent is launched on the Agent Platform Main- Container.

♦ Kill Selected Items

This action kills all the agents and agent containers currently selected. Killing an agent is equivalent to calling its doDelete() method, whereas killing an agent container kills all the agents living on the container and then de-registers that container from the platform. Of course, if the Agent Platform Main-Container is currently selected, then the whole platform is shut down.

♦ Suspend Selected Agents

This action suspends the selected agents and is equivalent to calling the doSuspend() method. Beware that suspending a system agent, particularly the AMS, deadlocks the entire platform.

♦ Resume Selected Agents

This action puts the selected agents back into the AP_ACTIVE state, provided they were suspended, and works just the same as calling their doActivate() method.

♦ Migrate Agent

This action allows to migrate an agent. When the user selects this menu item, a special dialog is displayed in which the user must specify the container of the platform where the selected agent must migrate. Not all the agents can migrate because of lack of serialization support in their implementation. In this case the user can press the cancel button of this dialog.

♦ Clone Agent

This action allows to clone a selected agent. When the user selects this menu item a dialog is displayed in which the user must write the new name of the agent and the container where the new agent will start.

♦ RemotePlatforms menu:

This menu allows controlling some remote platforms that comply with the FIPA specifications. Notice that these remote platforms can even be non-JADE platforms.

♦ Add Remote Platform via AMS AID

This action allows getting the description (called APDescription in FIPA terminology) of a remote Agent Platform via the remote AMS. The user is requested to insert the AID of the remote AMS and the remote platform is then added to the tree showed in the RMA GUI.

♦ Add Remote Platform via URL

This action allows getting the description (called APDescription in FIPA terminology) of a remote Agent Platform via a URL. The content of the URL must be the stringified APDescription, as specified by FIPA. The user is requested to insert the URL that contains the remote APDescription and the remote platform is then added to the tree showed in the RMA GUI.

♦ Remove Remote Platform

This action permits to remove from the GUI the selected remote platform.

♦ Refresh Agent List

This action performs a search with the AMS of the Remote Platform and the full list of agents belonging to the remote platform are then displayed in the tree.

4.4 Graphical User Interface of the system

As we are using the JADE simulator, the main GUI screens are obtained and used the same as those used in JADE itself.

The mostly used and most important ones are the following:

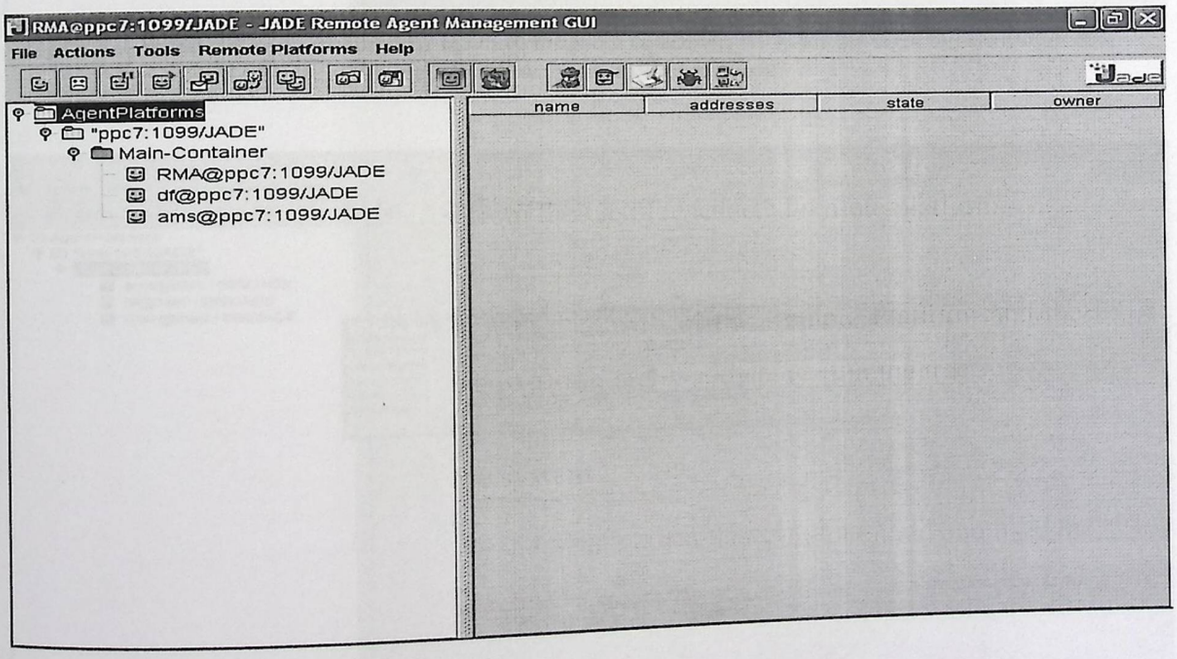


Figure 4.6 This is the first page that appear when we compile the JADE simulation, it shows the default platform "Agent Platform", with the default container "Main-Container" which include the three default agents, the RMA, df and ams.

♦ Remove Remote Platform

This action permits to remove from the GUI the selected remote platform.

♦ Refresh Agent List

This action performs a search with the AMS of the Remote Platform and the full list of agents belonging to the remote platform are then displayed in the tree.

4.4 Graphical User Interface of the system

As we are using the JADE simulator, the main GUI screens are obtained and used the same as those used in JADE itself.

The mostly used and most important ones are the following:

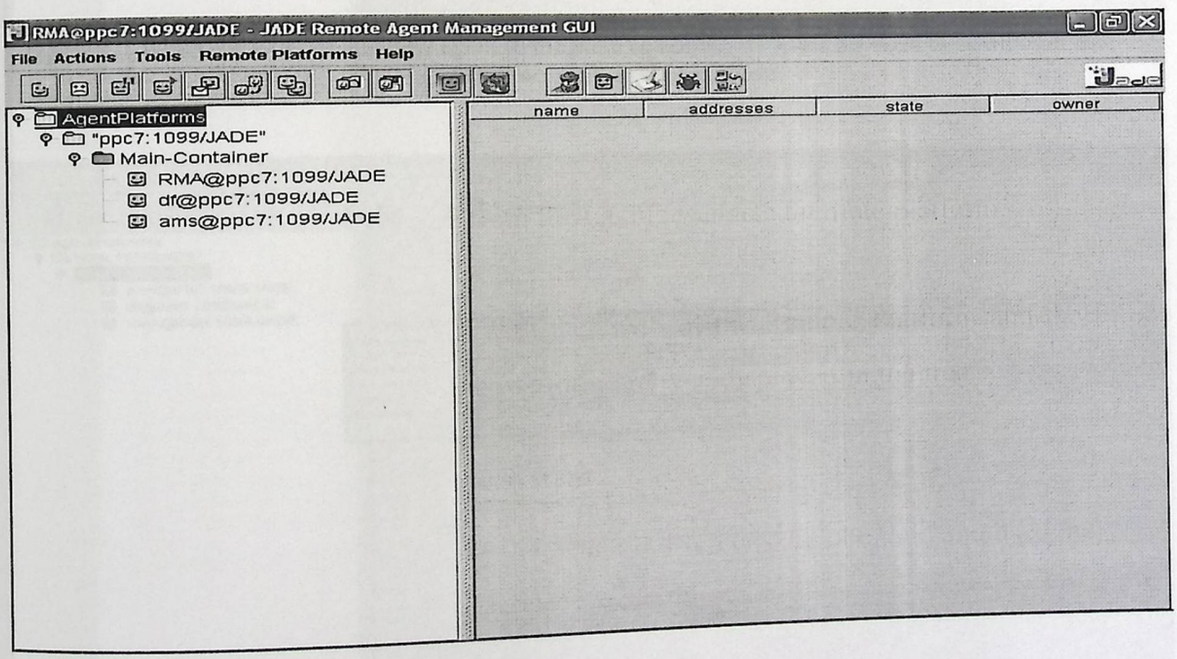


Figure 4.6 This is the first page that appear when we compile the JADE simulation, it shows the default platform "Agent Platform", with the default container "Main-Container" which include the three default agents, the RMA, df and ams.

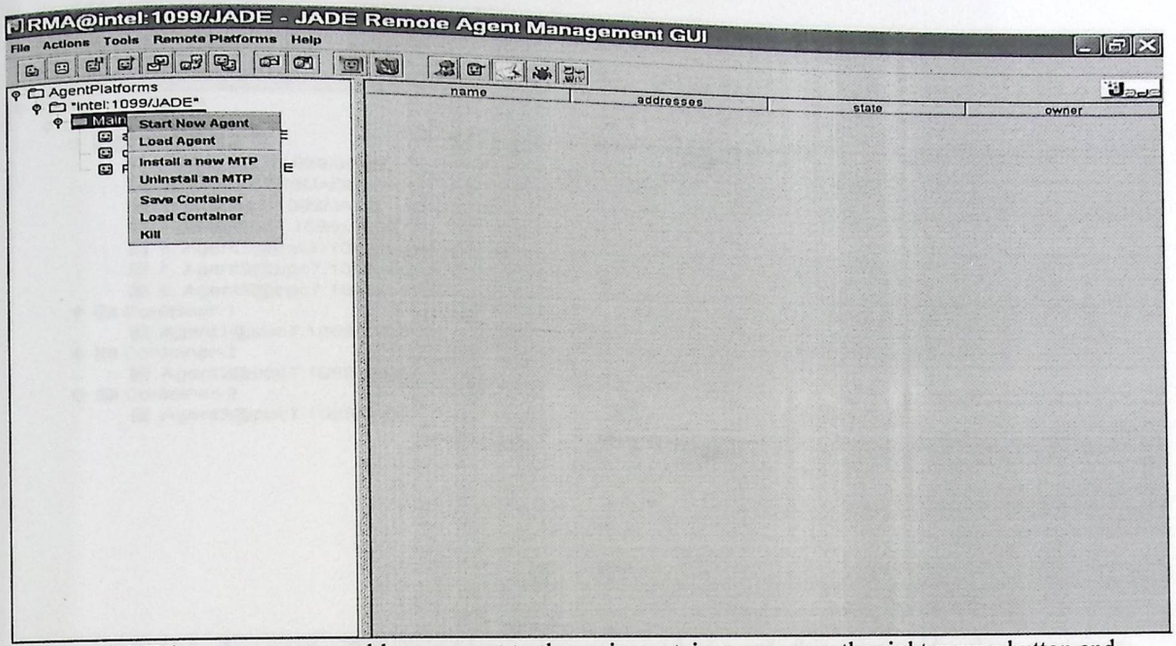


Figure 4.7 When we want to add new agent to the main container, we press the right mouse button and select start new agent, as shown above.

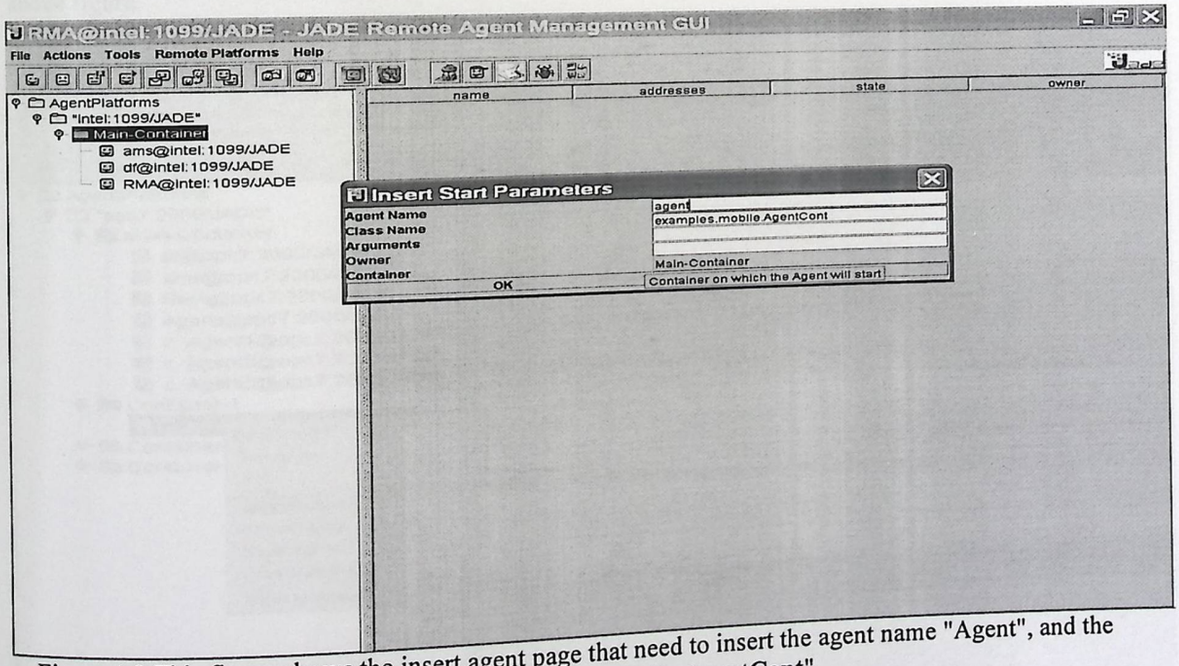


Figure 4.8 This figure shows the insert agent page that need to insert the agent name "Agent", and the class name **"examples.mobile.AgentCont"**.

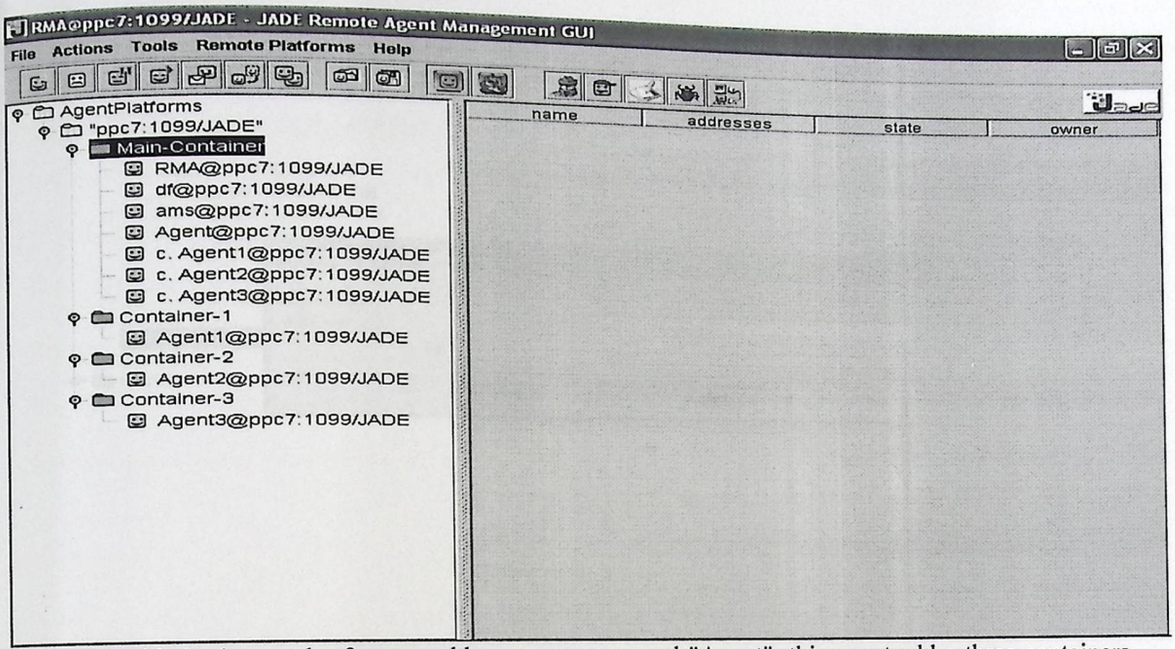


Figure 4.9 This is the graph after we add anew agent named "Agent", this agent add a three containers, Container-1, Container-2, and Container-3 each with mobile agent Agent1, Agent2, and Agent3, theses agent cloned them self to the Main-Container with c.Agent1, c.Agent2, and c.Agent3 form as shown in the above figure.

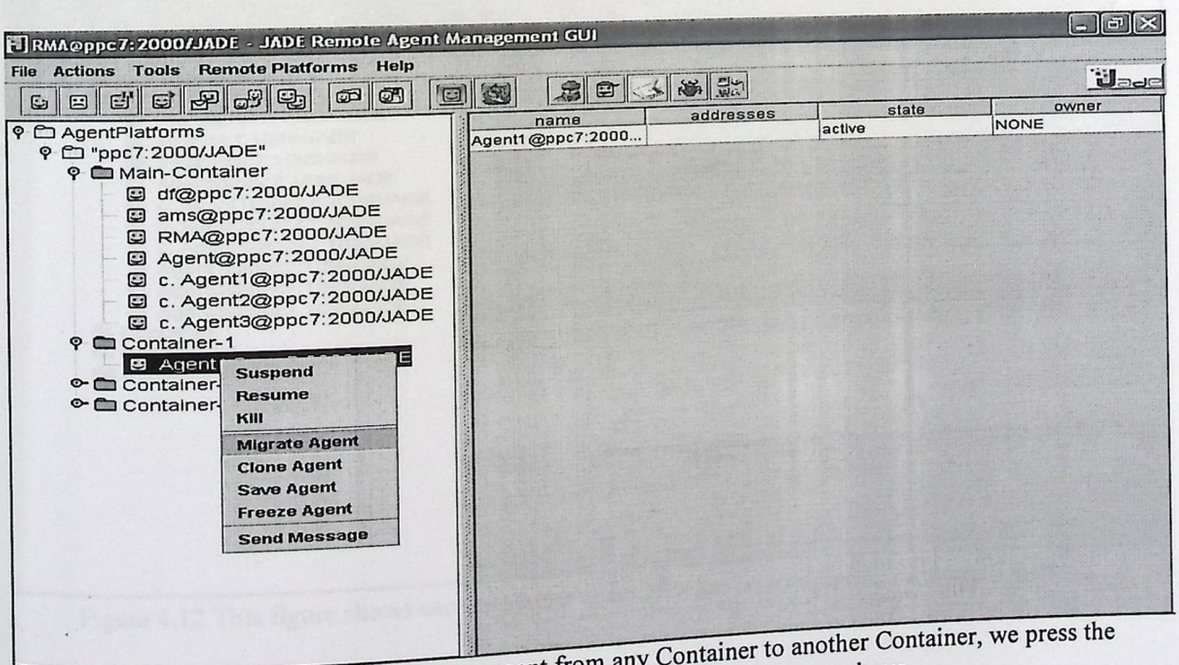


Figure 4.10 When we want to migrate an agent from any Container to another Container, we press the right mouse button and select Migrate Agent, as shown above.

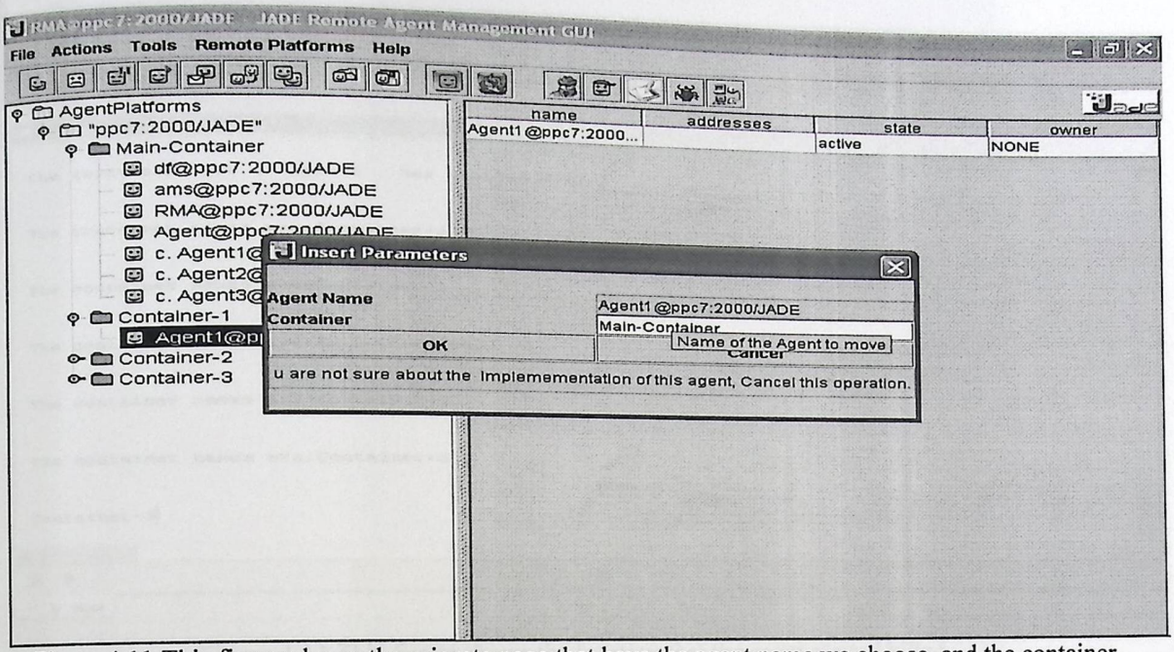


Figure 4.11 This figure shows the migrate page that have the agent name we choose, and the container name we want to move to "Main-Container ".

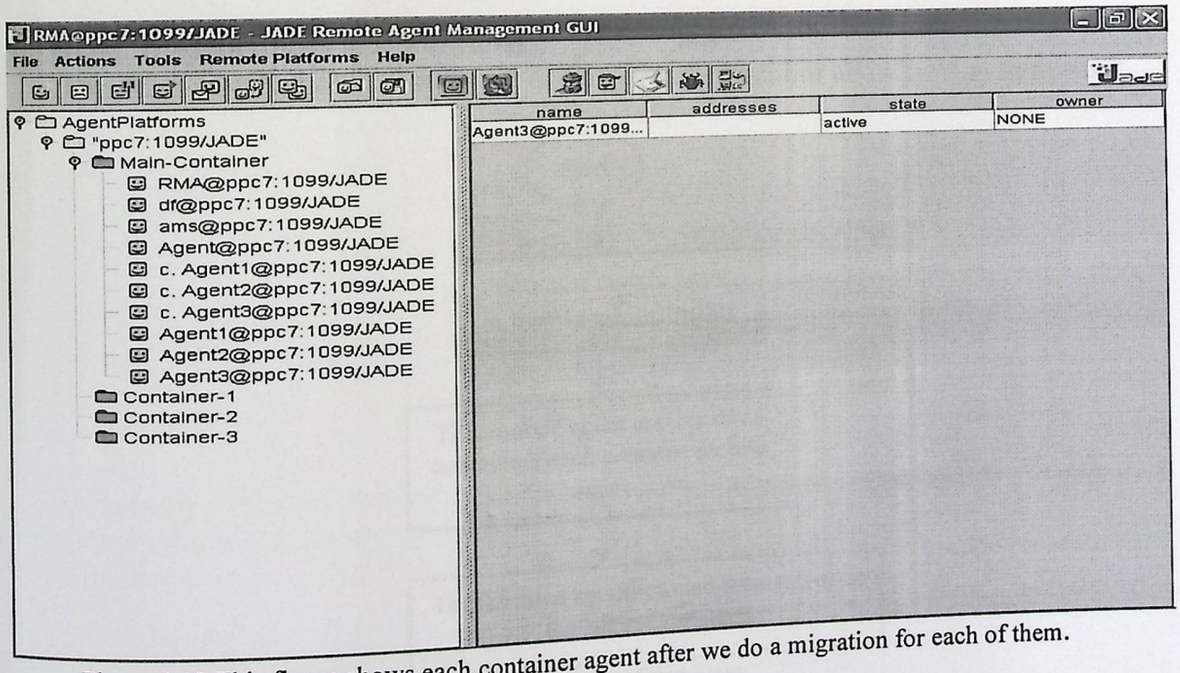


Figure 4.12 This figure shows each container agent after we do a migration for each of them.

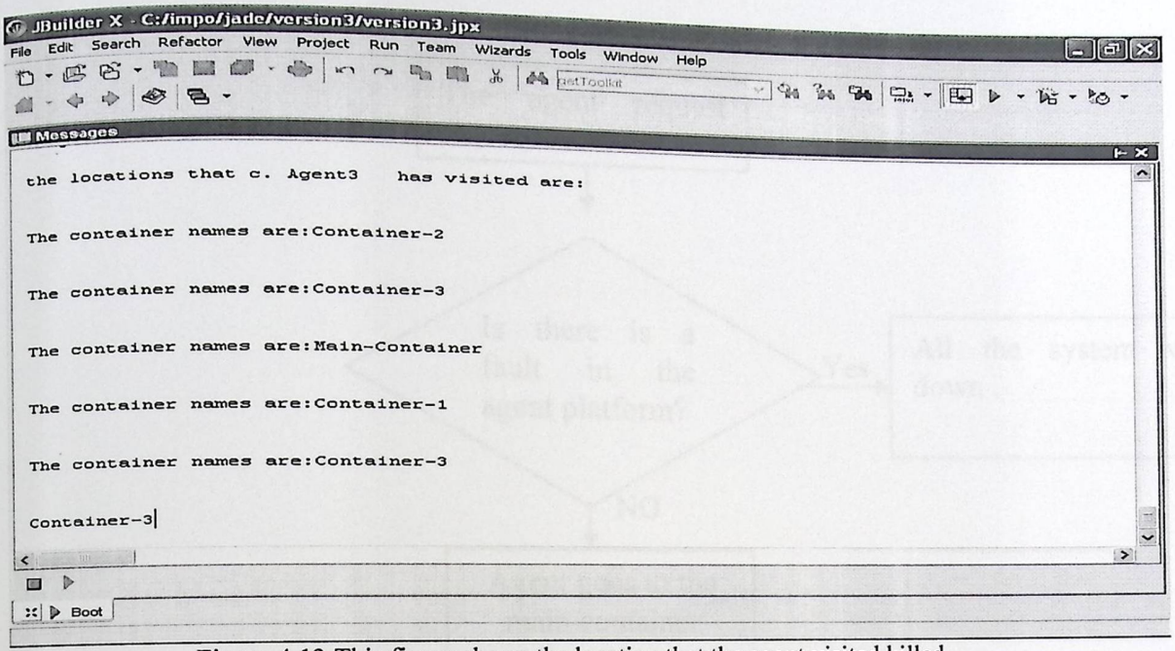


Figure 4.13 This figure shows the location that the agent visited killed.

4.5 General Algorithms and Flowcharts

This section will show a flow chart and a general algorithm about how system work.

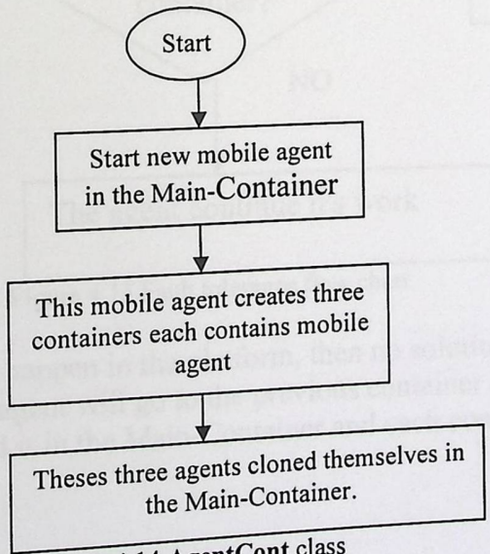


Figure 4.14 **AgentCont** class

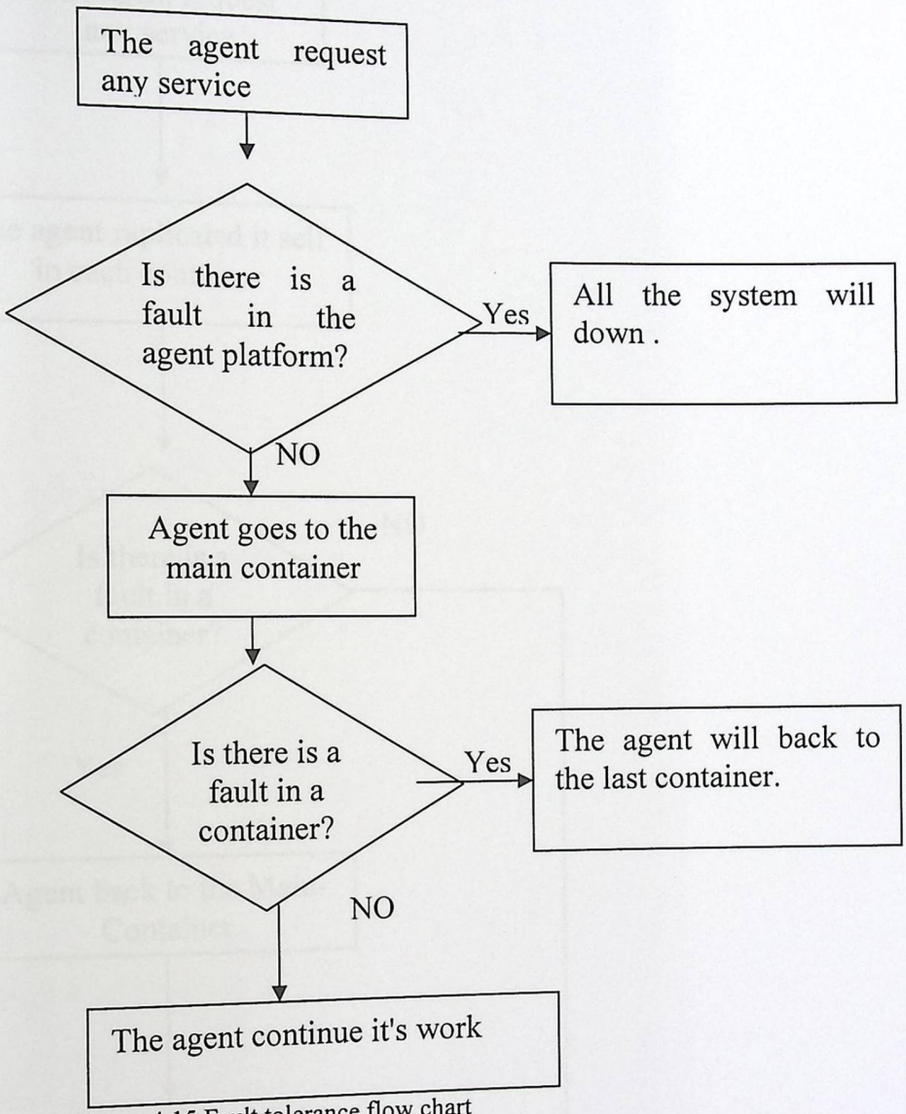


Figure 4.15 Fault tolerance flow chart

Fault tolerance happen may happen in the platform, then no solution can use, when it happen in the container the agent will go to the previous container and take all information and it replicated it in the Main-Container and each container that agent visit it .

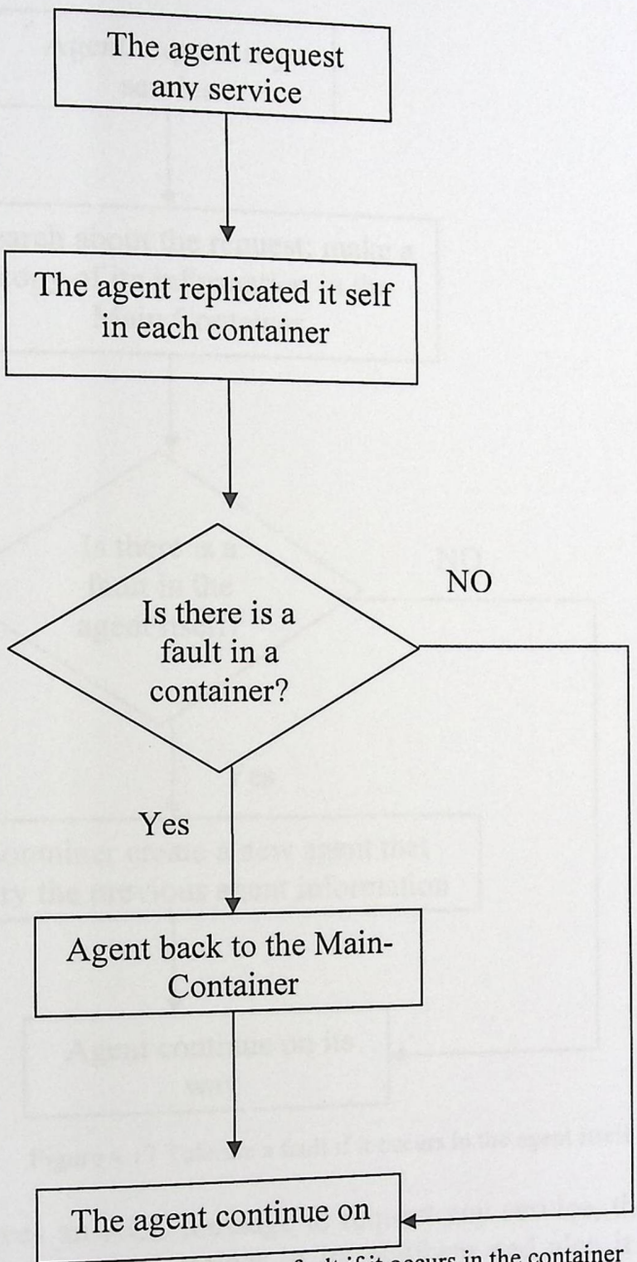


Figure 4.16 Tolerate a fault if it occurs in the container

The agent gives an ACL message to request any service, it save a copy of its information in the Main Container, if there is a fault in the container, it will back to the last container he visit it and take the information and continue.

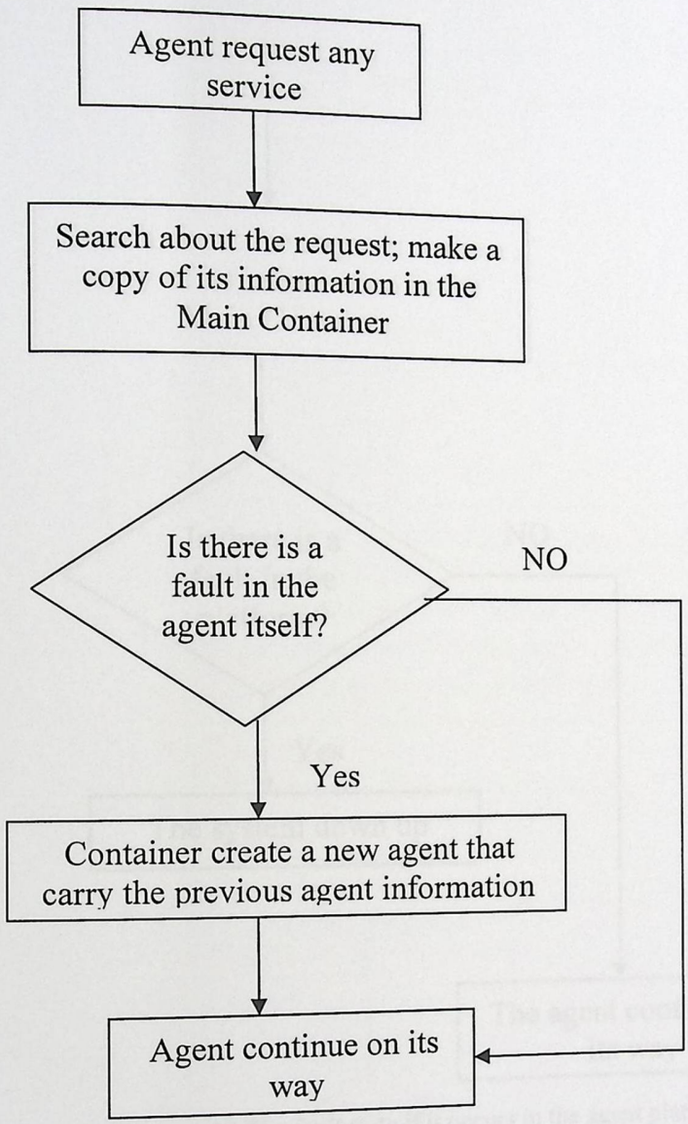


Figure 4.17 Tolerate a fault if it occurs in the agent itself

The agent gives an ACL message to request any service, then the agent search about this request in each container in its platform and also it save a copy of its information in the Main Container, if there is a fault in the agent itself the container that the agent fault in should create a new agent that carry the previous agent information, else if there is no fault in the agent this agent will continue on its way (search) to find its goal

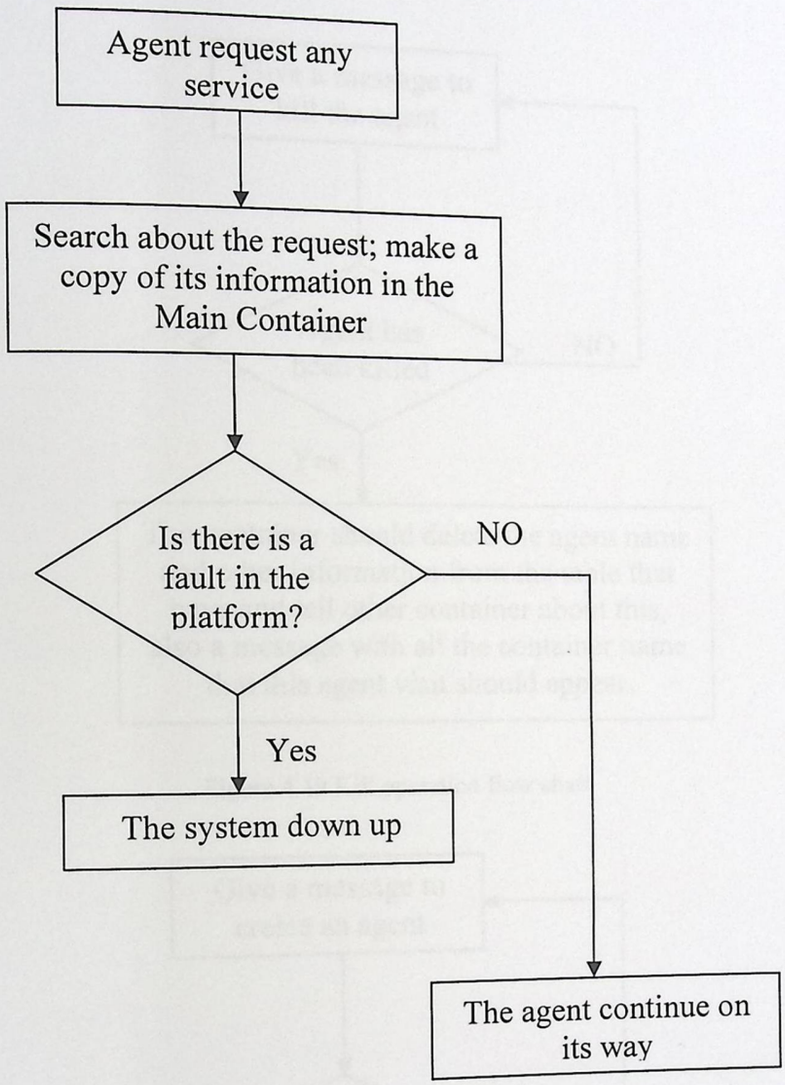


Figure 4.18 The fault state if it occurs in the agent platform

The agent gives an ACL message to request any service, then the agent search about this request in each container in its platform and also it save a copy of its information in the Main Container, if there is a fault in the platform the system will down up, (this mean that there is no fault tolerance for this state), else if there is no fault in the platform the agent continue on its way (search) to find its goal

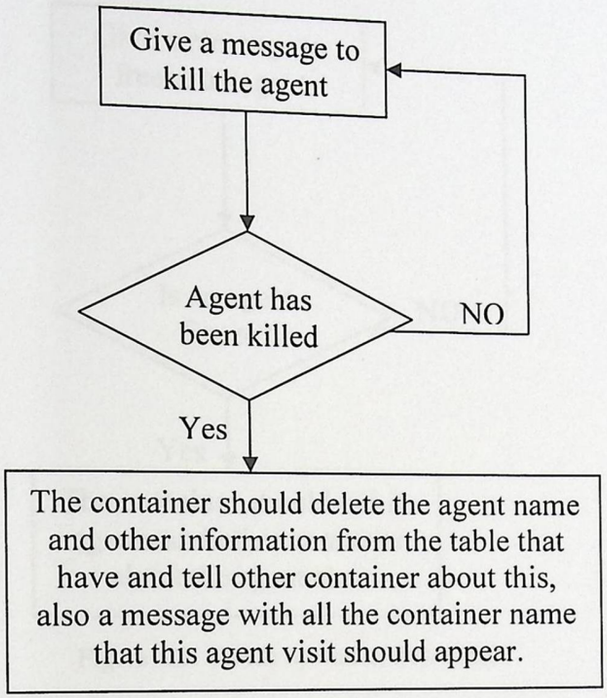


Figure 4.19 Kill operation flow chart

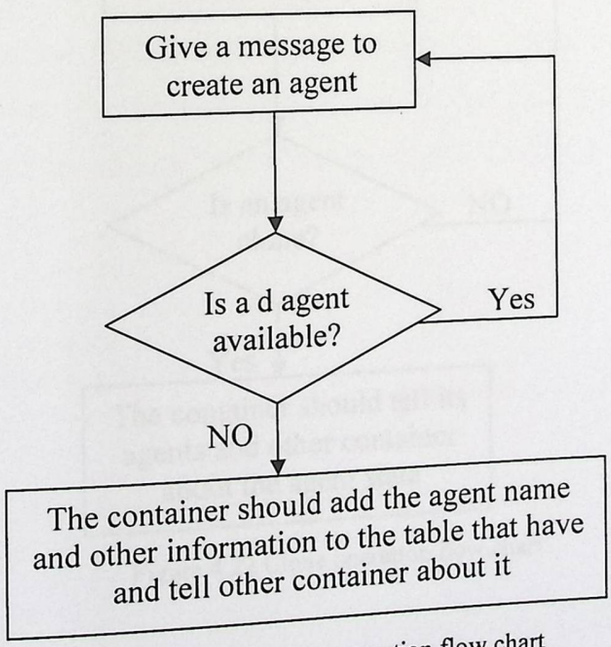


Figure 4.20 Create operation flow chart

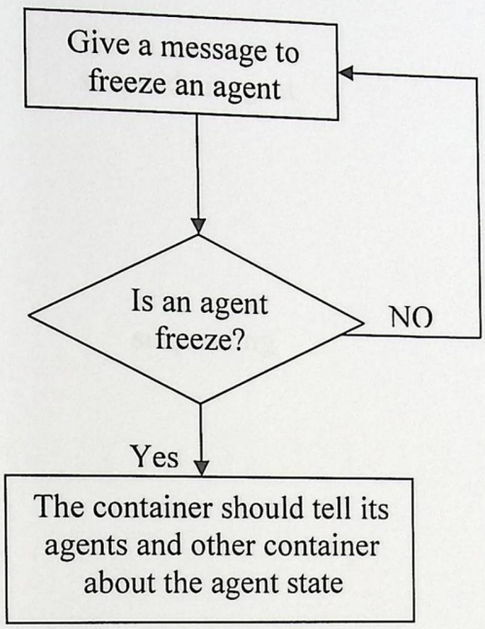


Figure 4.21 Freeze operation flow chart

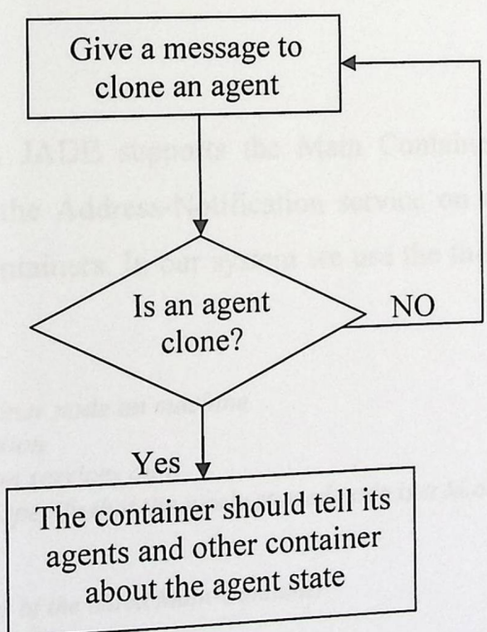


Figure 4.22 Clone operation flow chart

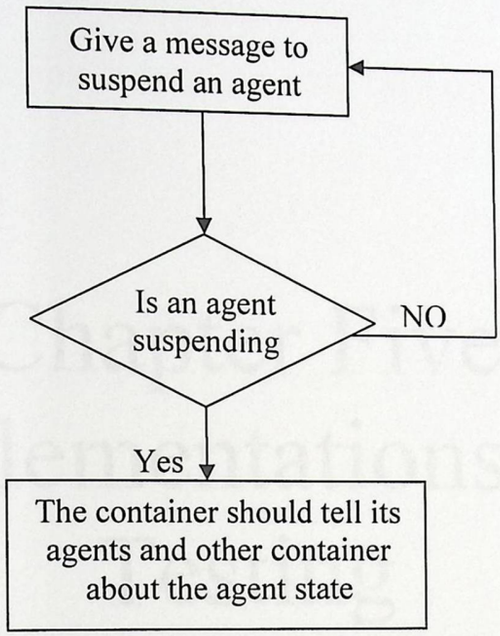


Figure 4.23 Suspend operation flow chart

4.5.1 Algorithm

For fault-tolerance, JADE supports the Main Container node list to peripheral containers. By activate the Address-Notification service on all Main Container nodes and on the peripheral containers. In our system we use the this option and the algorithm is :-

{
starts a master Main Container node on machine
activates the Main-Replication
activate Address-Notification services on it.
activating `backupmain` *to specify that the newly started node is a Main-Container*
if Main-Container killed
{
Backupmain is take places of the killed Main-Container
}
}

Chapter Five
Implementations and Testing

Overview
Implementations
Testing

Chapter Five
Implementations and Testing

5.1 Overview

In this chapter we are going to talk about the project implementation, jade tools, the command lines that we used, and the interfaces that appear for each of theses commands.

5.2 Implementations

In this section we will describe the project implementations

5.2.1 The code was built

We create a class named *AgentCont* that used to create three containers in each there is a mobile agent, also it creates a cloned version for each of the three created mobile agents in the Main-Container after we start a new mobile agent from this class.

This class can be compiled as shown in the following figures:

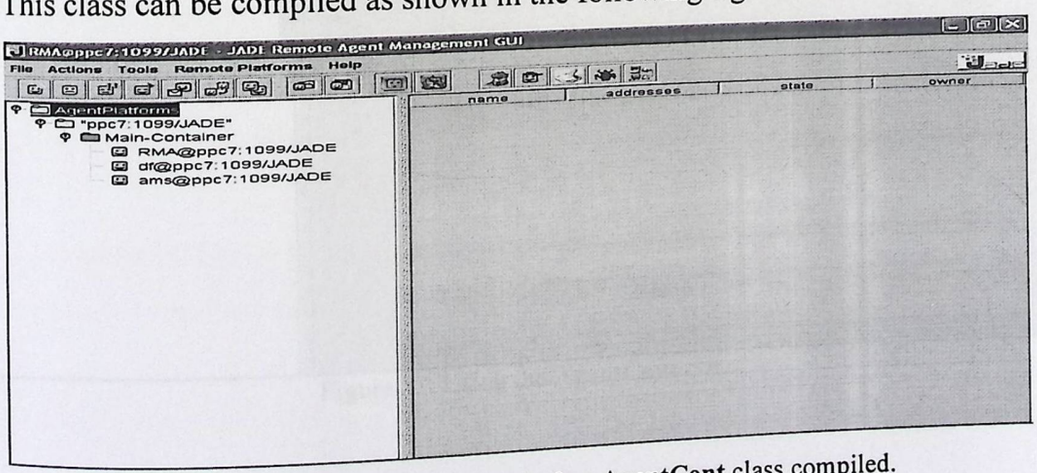


Figure 5.1 The first page appears before **AgentCont** class compiled.

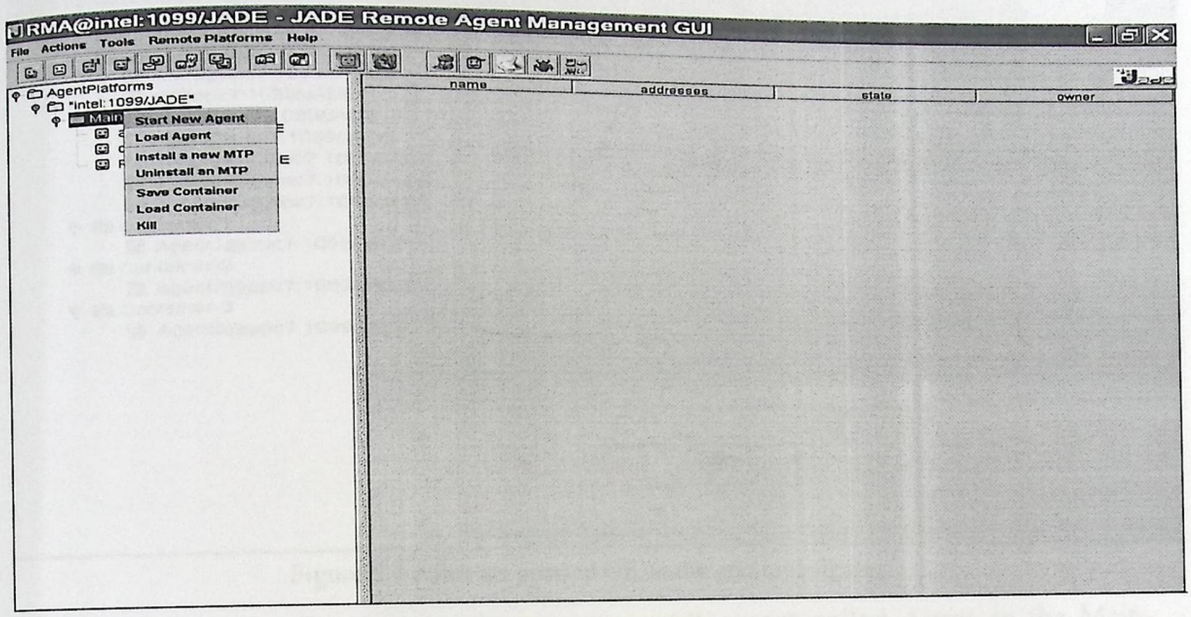


Figure 5.2 To start a new mobile agent from **AgentCont** class.

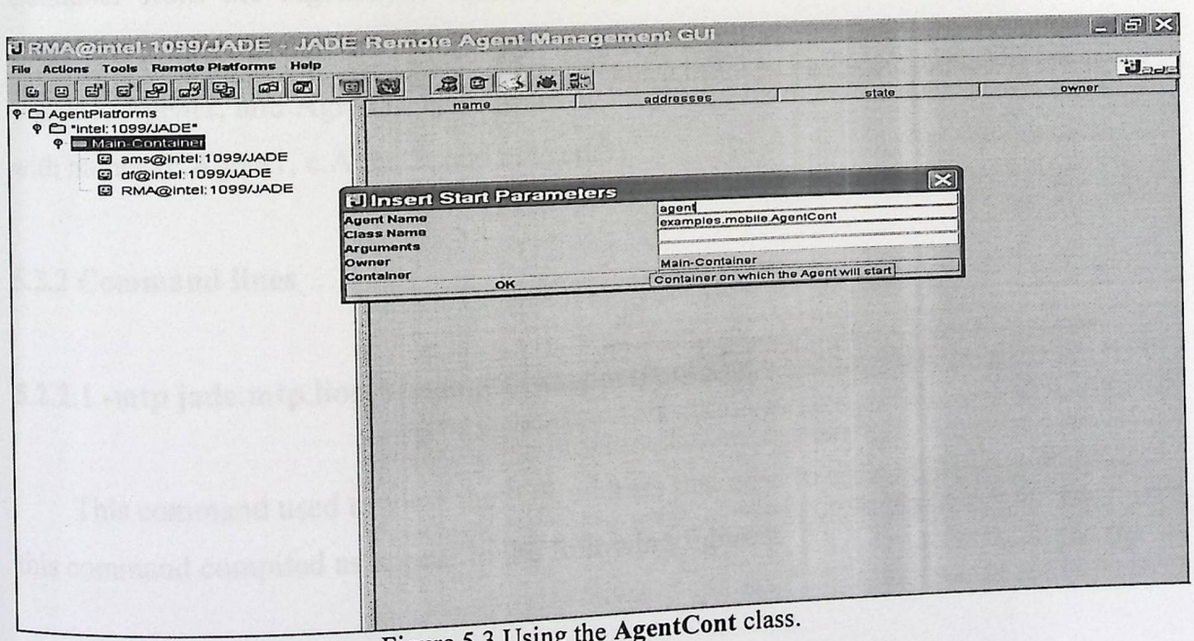


Figure 5.3 Using the **AgentCont** class.

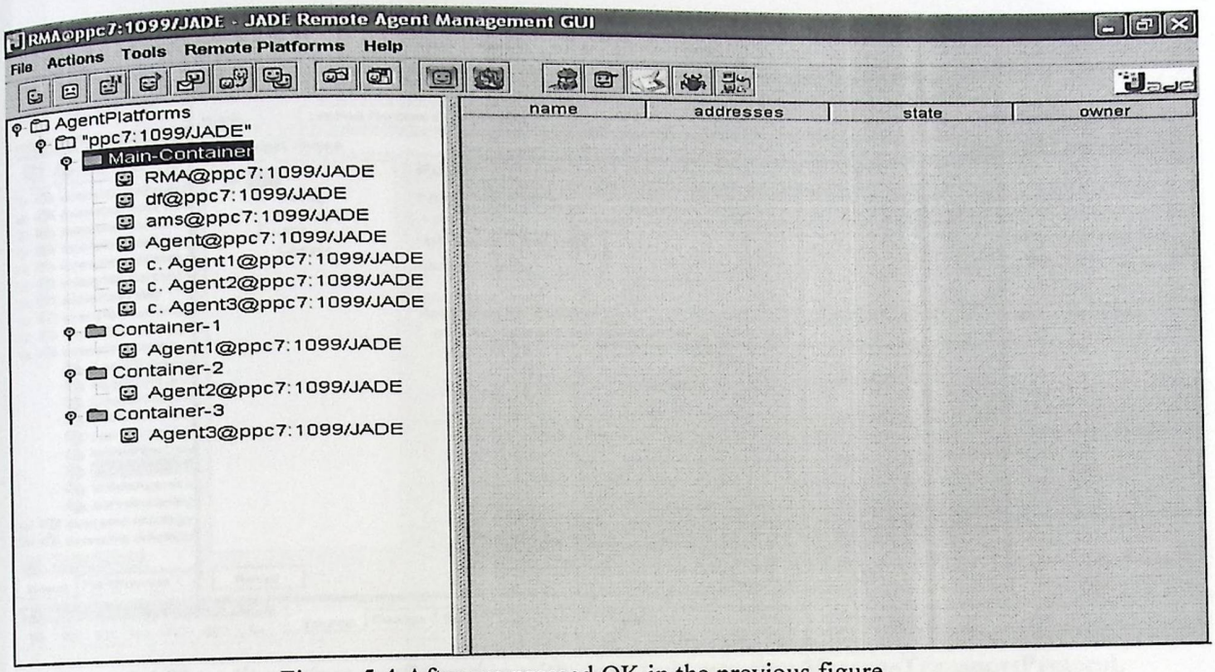


Figure 5.4 After we pressed OK in the previous figure.

In the previous figure we create a new mobile agent called Agent in the Main-Container from the AgentCont class, this agent built three containers (Container-1, Container-2, and Container-3) in each of theses containers there is a mobile agent (Agent1, Agent2, and Agent3), also theses agent cloned them self in the Main-Container with names (c.Agent1, c.Agent2, and c.Agent3).

5.2.2 Command lines

5.2.2.1 -mtp jade.mtp.iiop.MessageTransportProtocol

This command used to view the IOR address that used to create a Remote Platform, this command compiled as appear in the following figures:

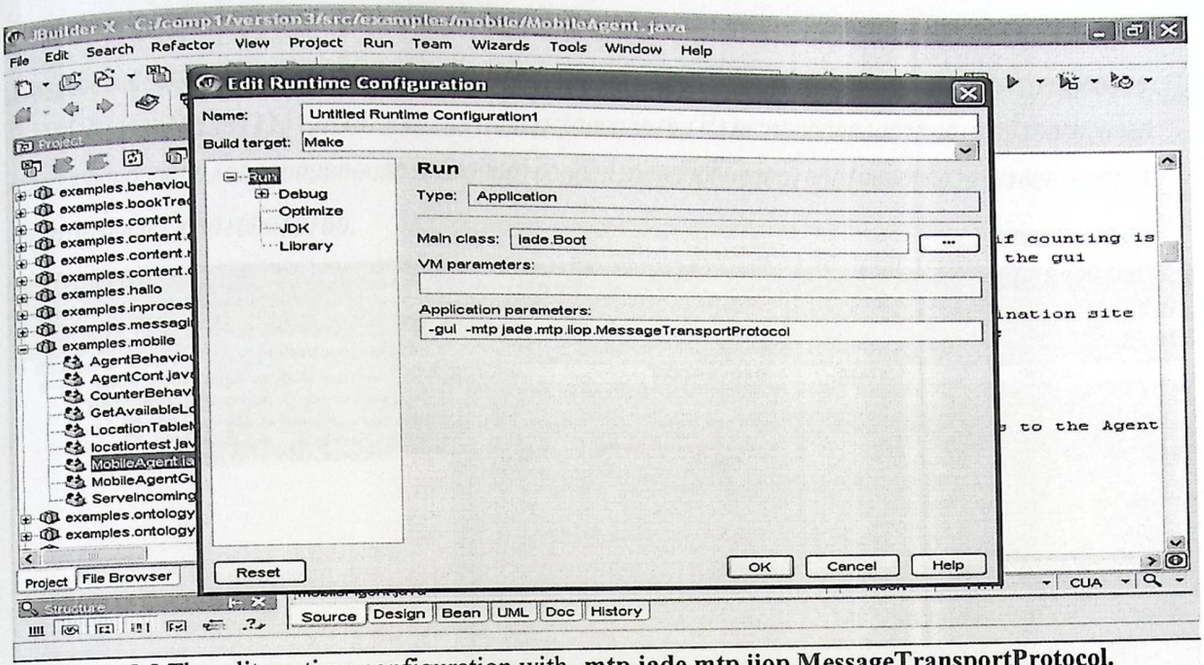


Figure 5.5 The edit runtime configuration with **-mtp jade.mtp.iiop.MessageTransportProtocol.**

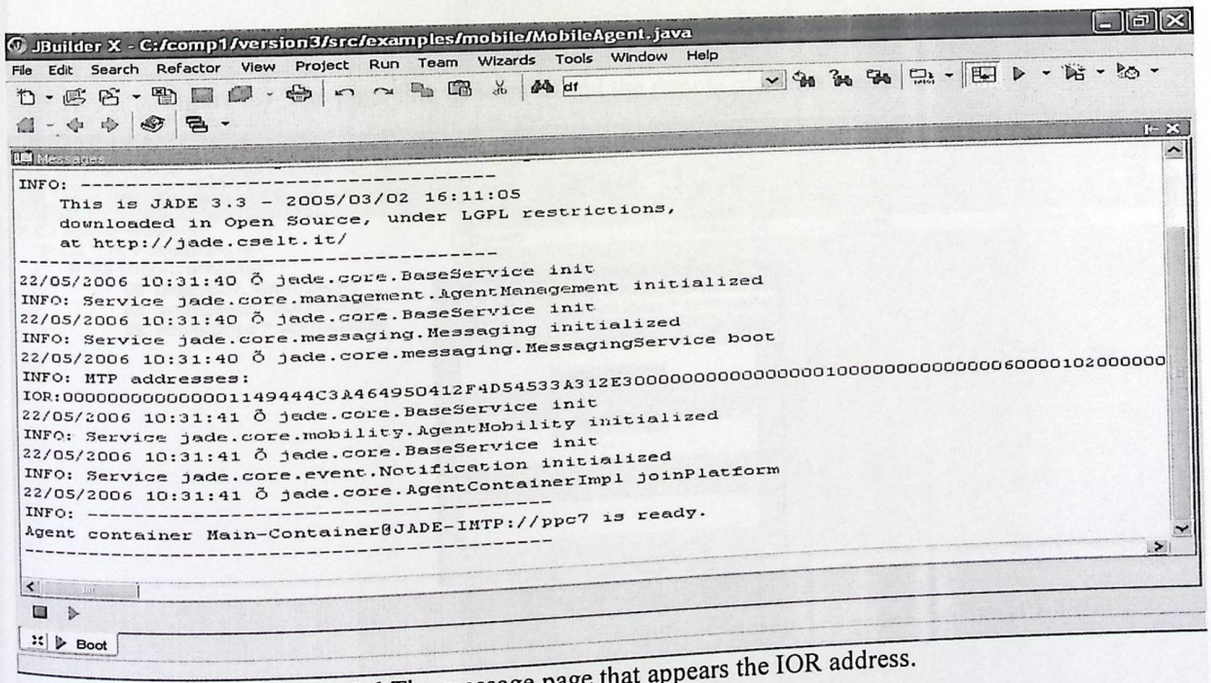


Figure 5.6 The message page that appears the IOR address.

The IOR that appear on the machine that used is (differ from machine to machine):

IOR:*00000000000001149444C3A464950412F4D54533A312E3000000000000000010000000000000006*
0000102000000000A3132372E302E302E310004F300000019AFABCB00000000025B019DB1000000080
000000000000000A000000000000001000000010000002000000000000100010000000205010001000100
20000101090000000100010100.

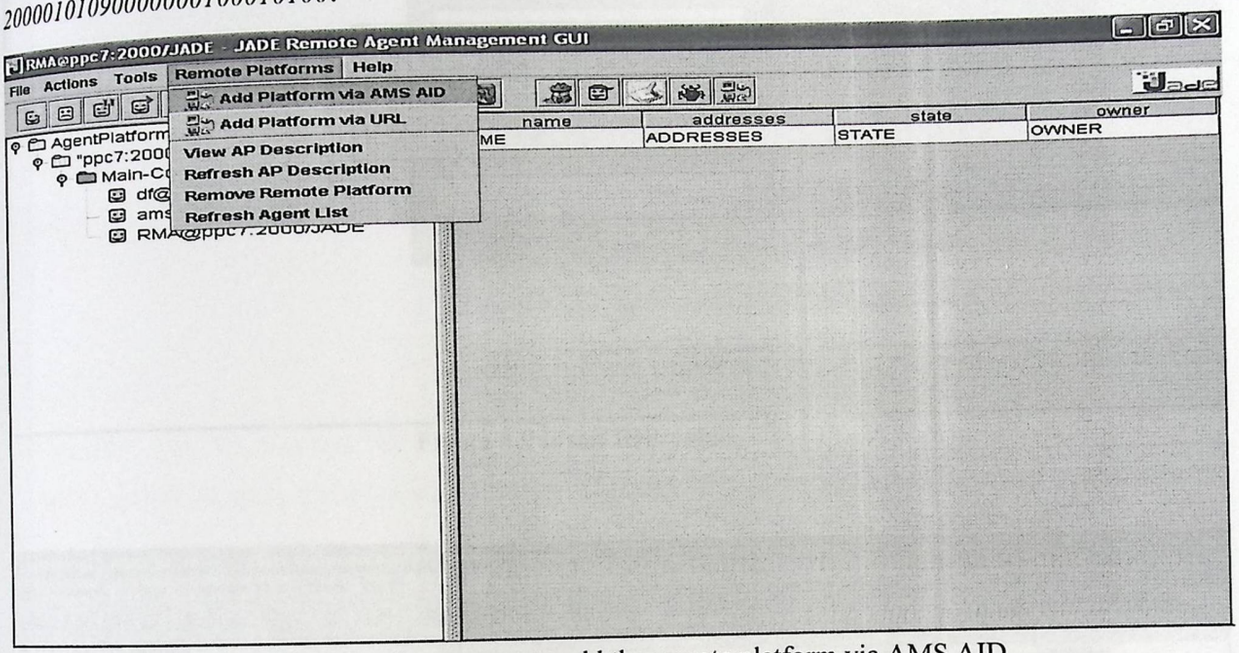


Figure 5.7 The page that appears to add the remote platform via AMS AID.

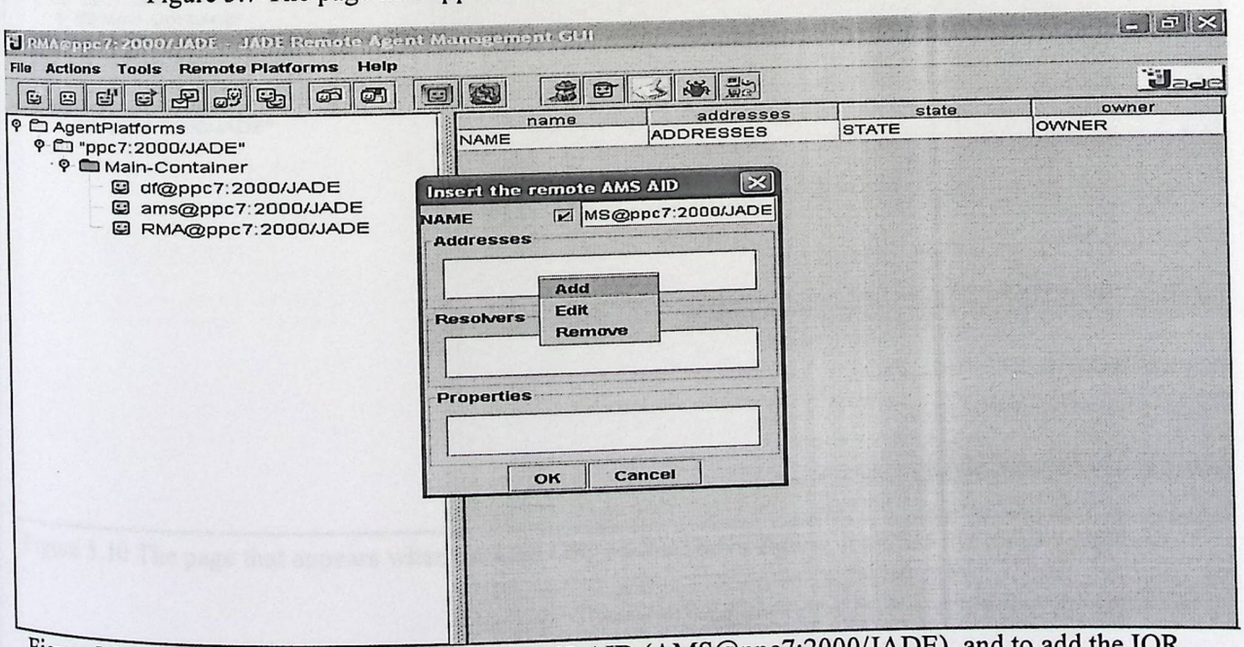


Figure 5.8 The page that appears to write the AMS AID (AMS@ppc7:2000/JADE), and to add the IOR

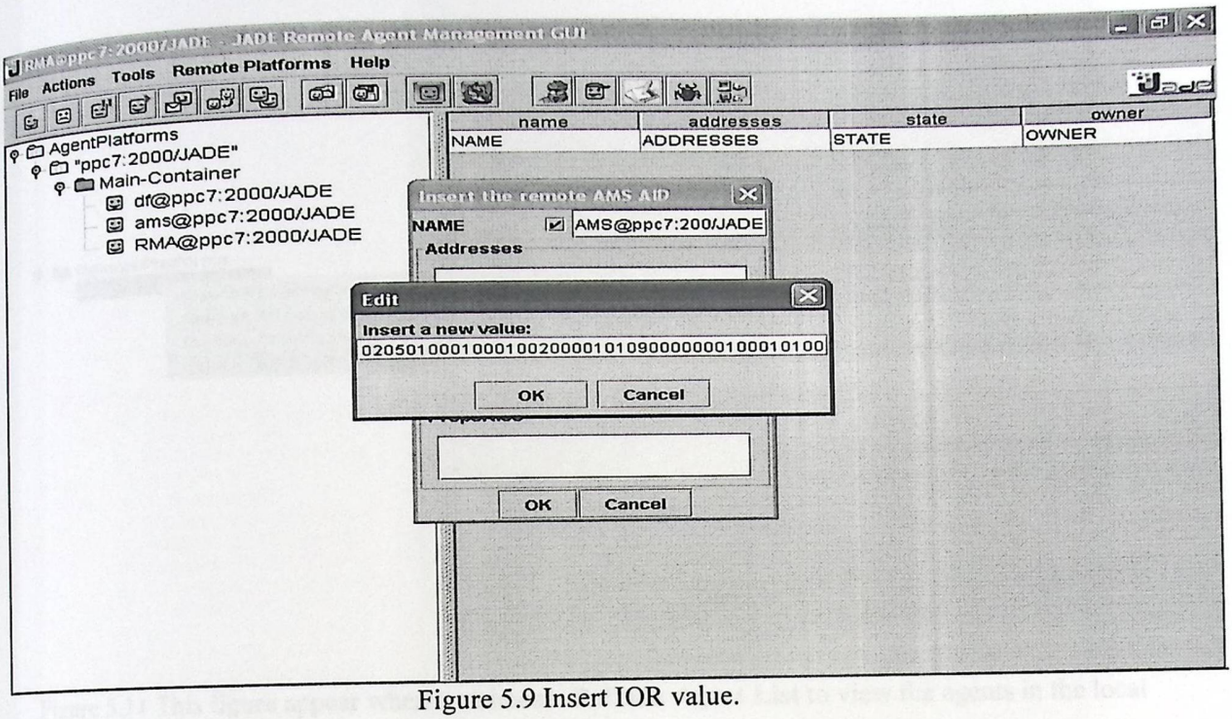


Figure 5.9 Insert IOR value.

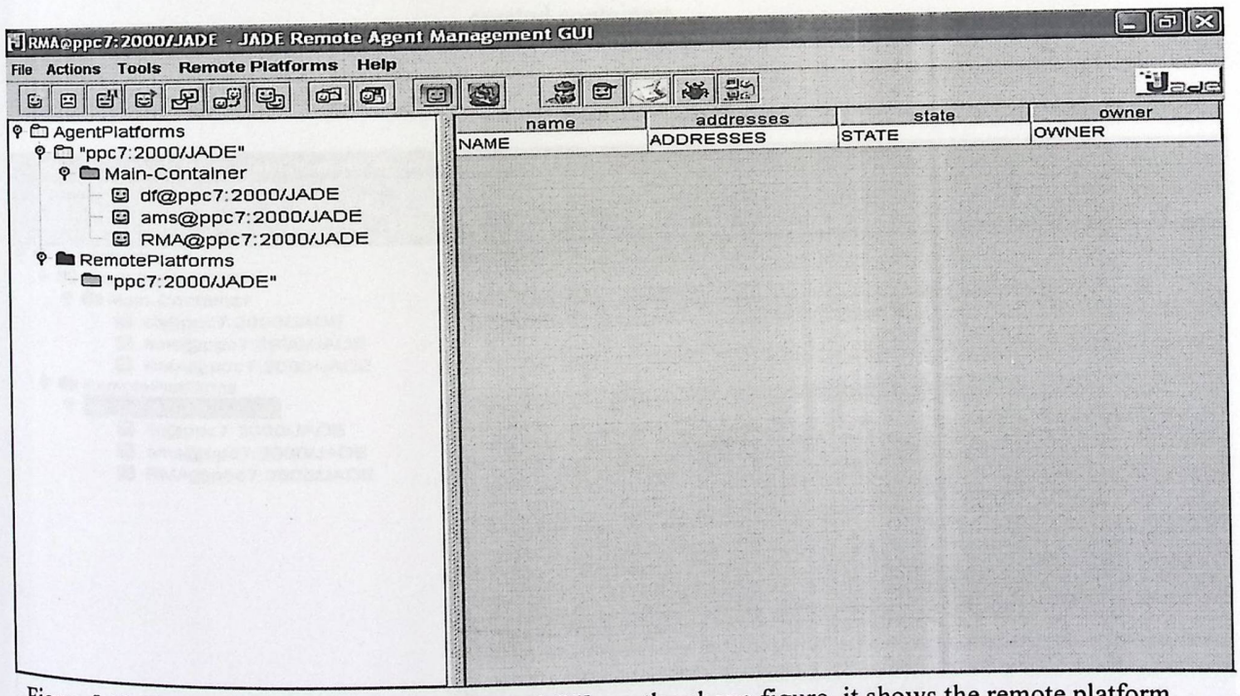


Figure 5.10 The page that appears when pressed OK on the above figure, it shows the remote platform.

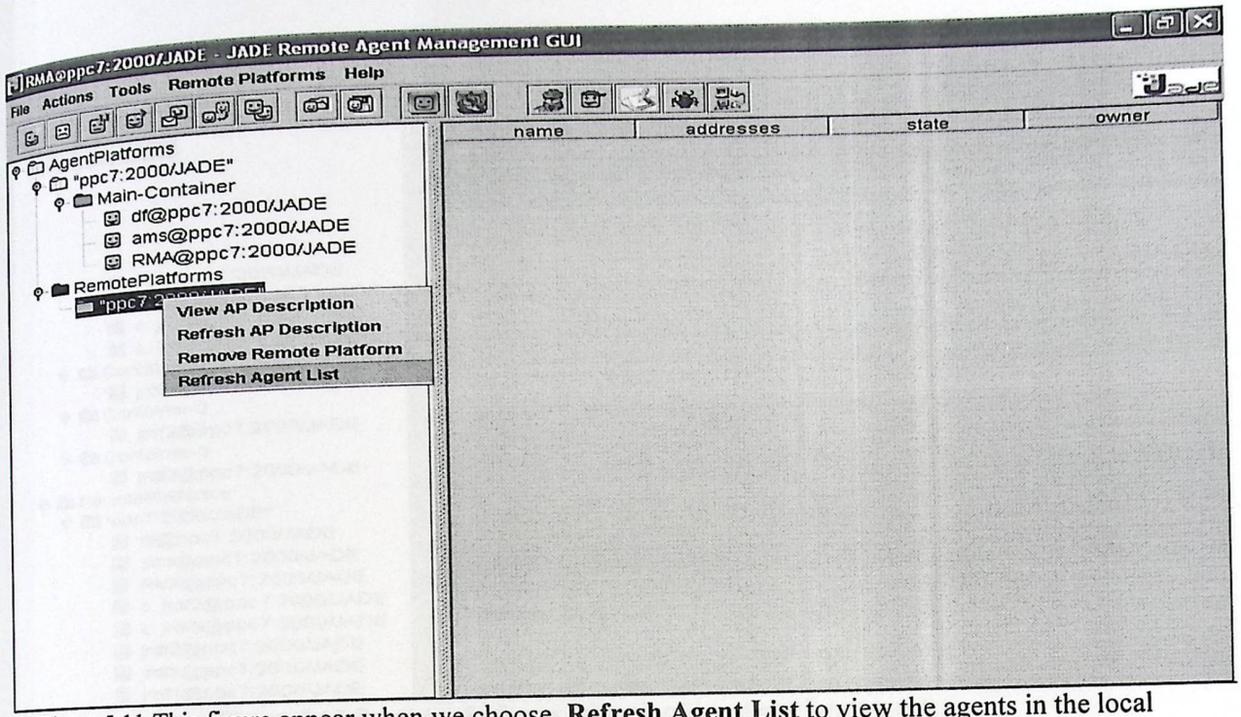


Figure 5.11 This figure appear when we choose **Refresh Agent List** to view the agents in the local platform.(AMS, df, rma), and all other created and cloned agents in the Main-Container and the other created containers.

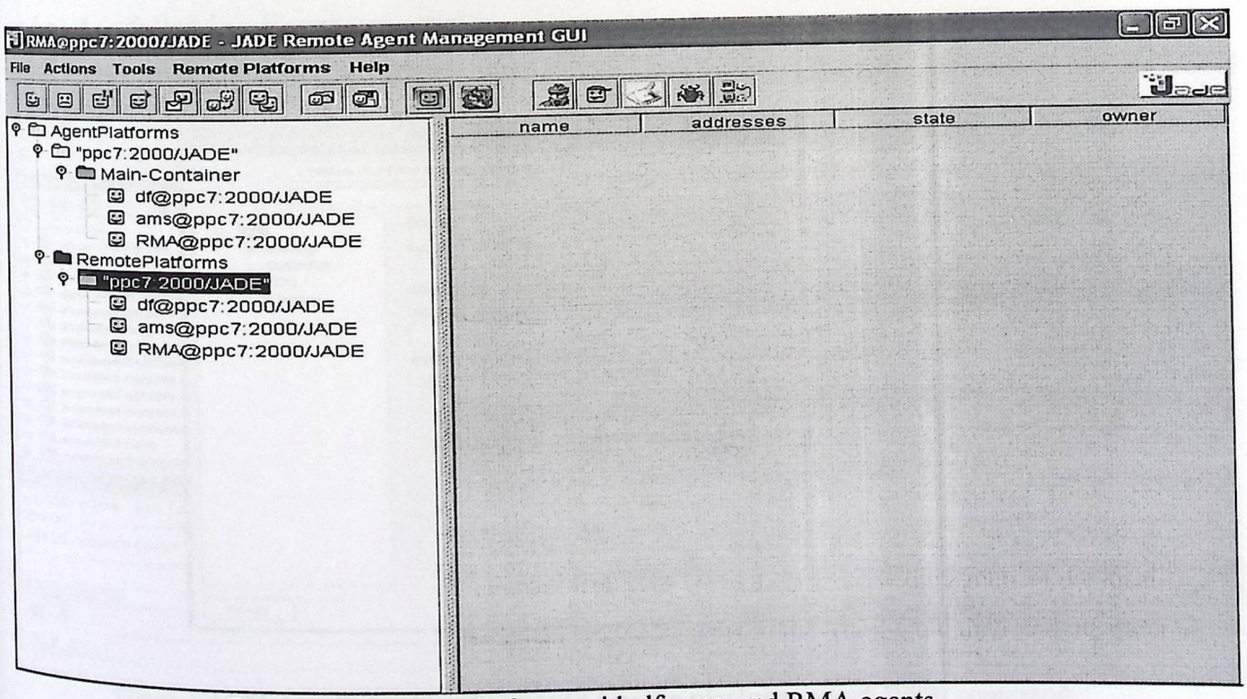


Figure 5.12 Remote platforms with df, ams, and RMA agents.

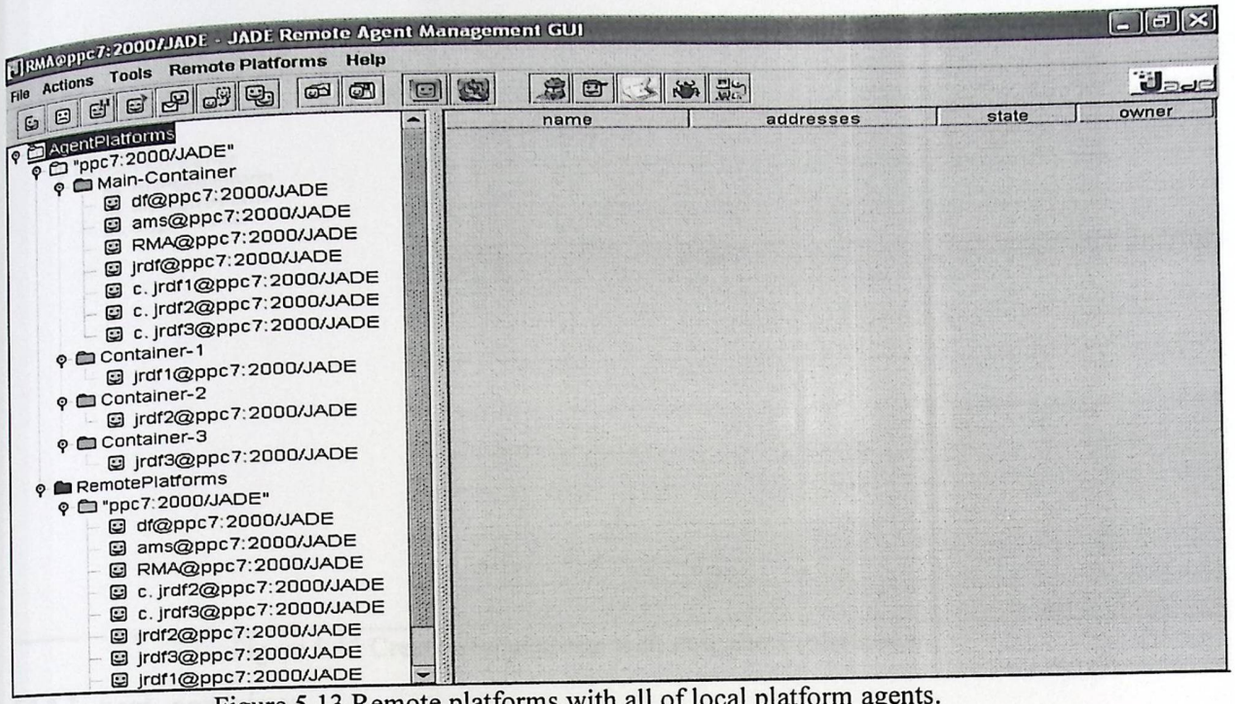


Figure 5.13 Remote platforms with all of local platform agents.

5.2.2.2 –name –port

This command is used to create new platform in new GUI page on different name, and port as following figures appear:

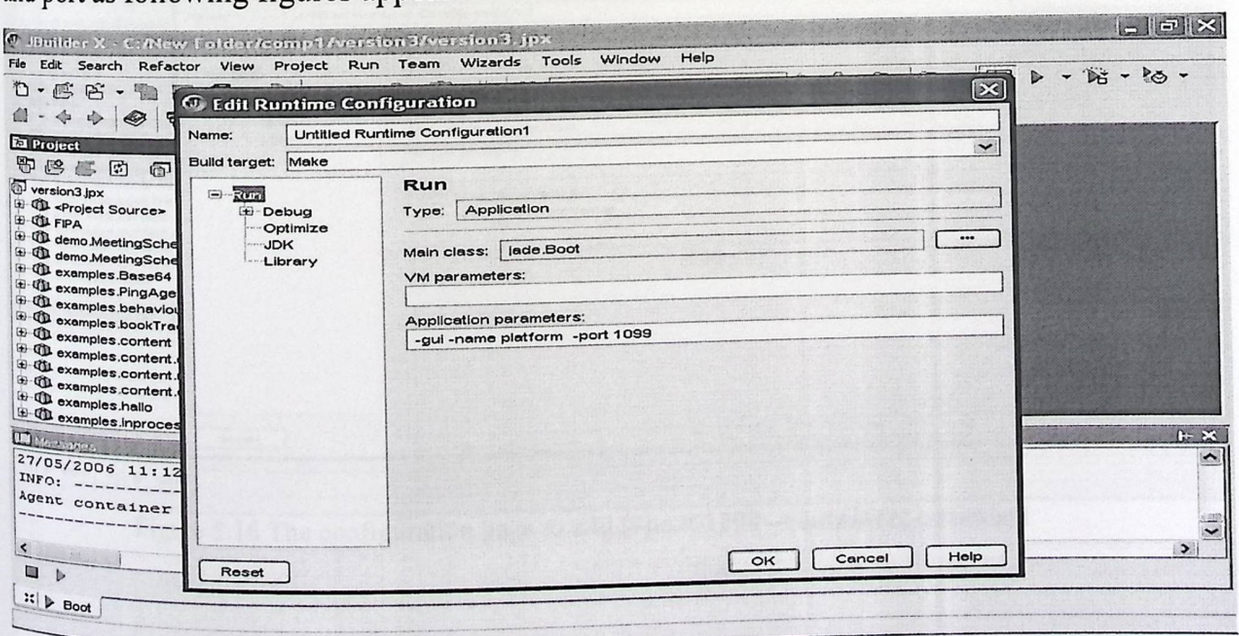


Figure 5.14 The configuration page to add **(-name platform –port 1099)** command.

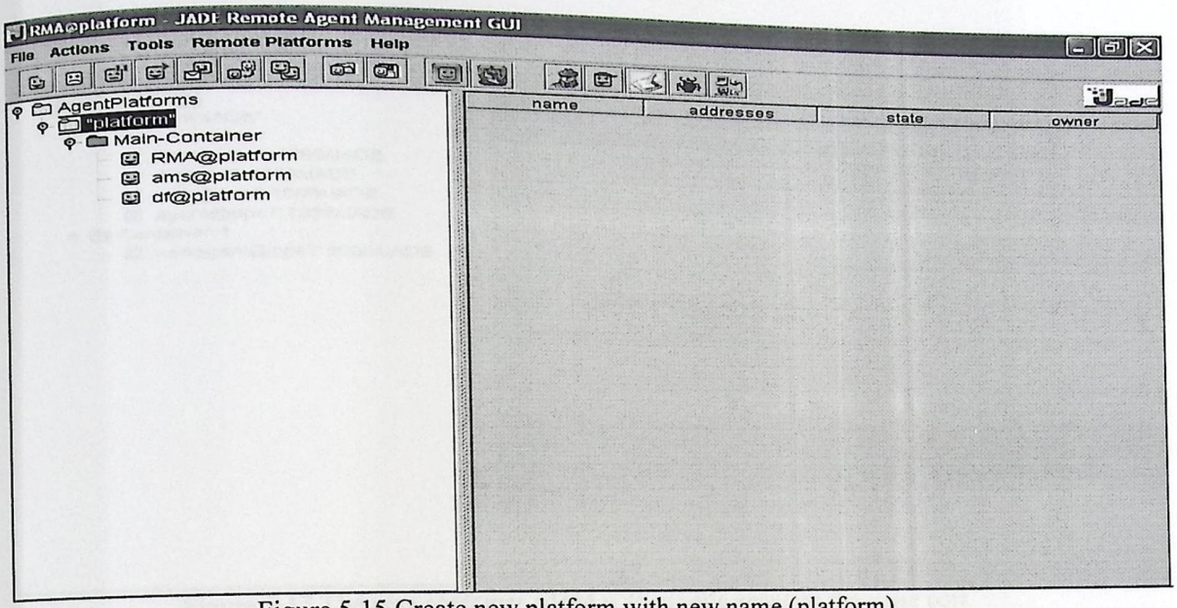


Figure 5.15 Create new platform with new name (platform).

5.2.2.3 –port -container

This command adds a container in new platform that appear on new GUI page with new port number.

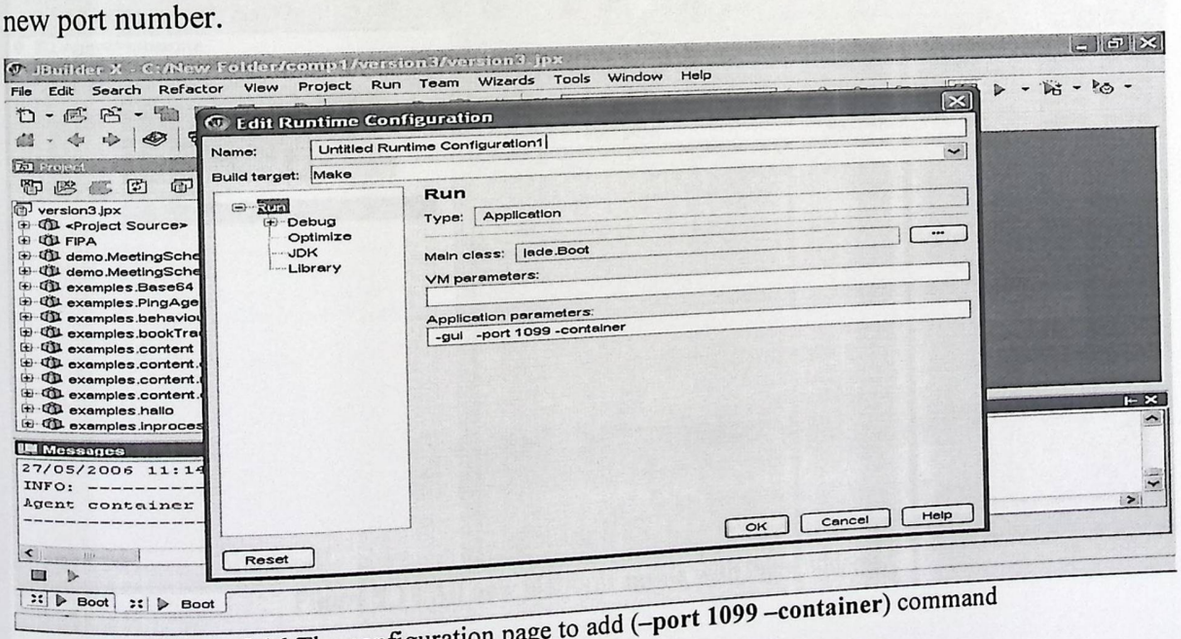


Figure 5.16 The configuration page to add (**–port 1099 –container**) command

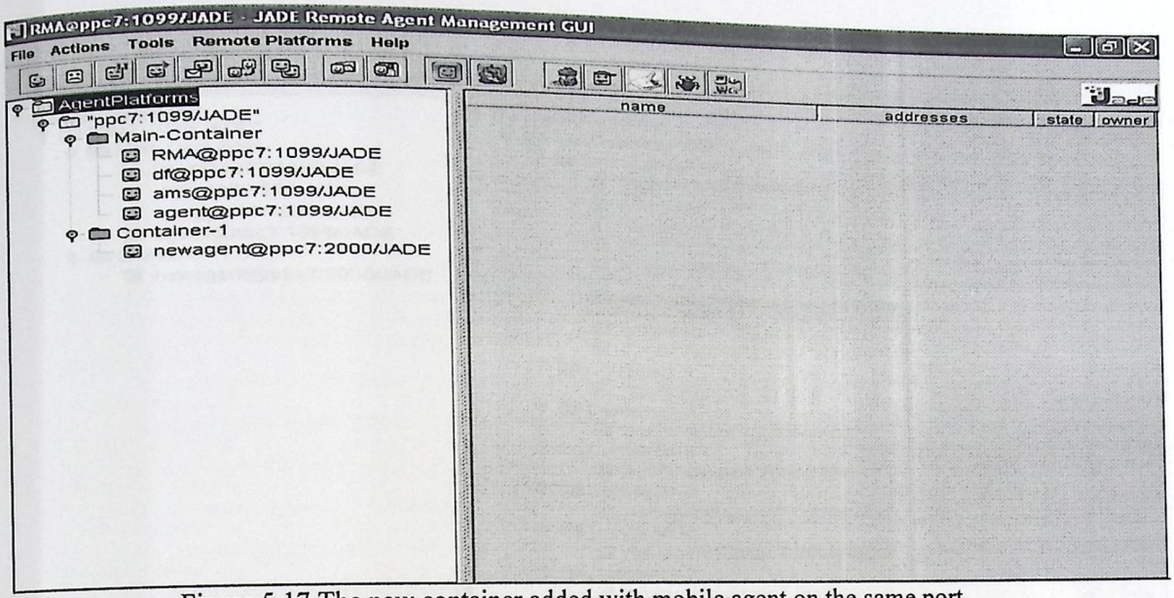


Figure 5.17 The new container added with mobile agent on the same port.

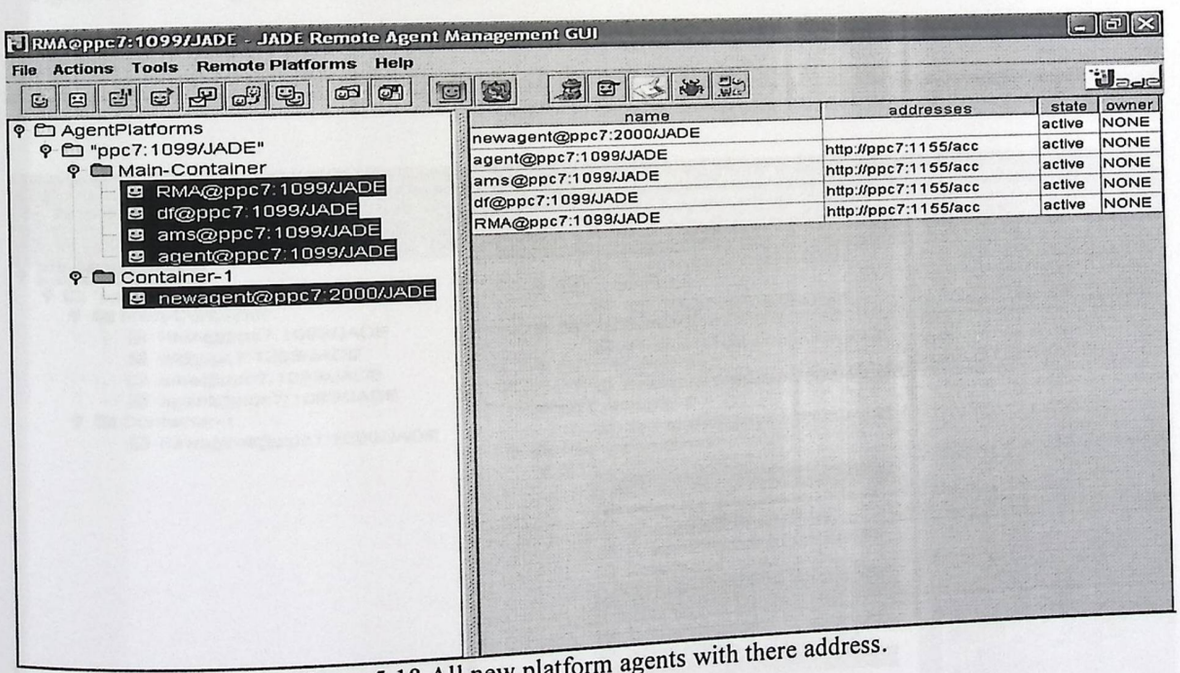


Figure 5.18 All new platform agents with there address.

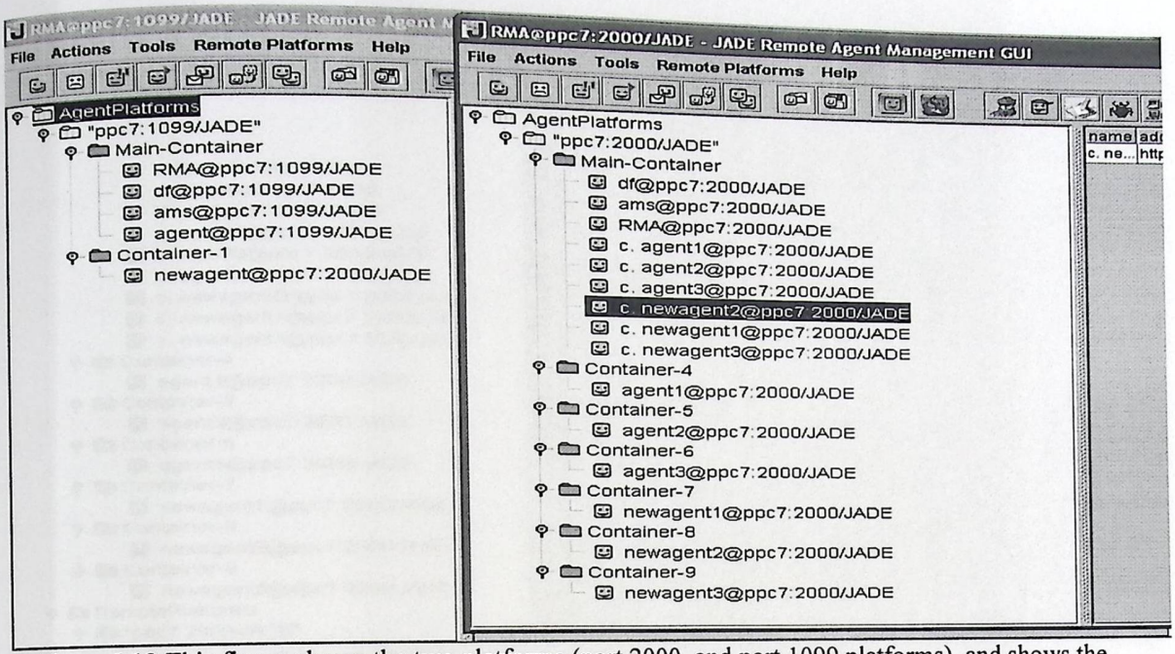


Figure 5.19 This figure shows the two platforms (port 2000, and port 1099 platforms), and shows the cloned agent and the containers that created after compiled **AgentCont** class on the port 1099 platform and theses agents appear on the port 2000 platform as shown.

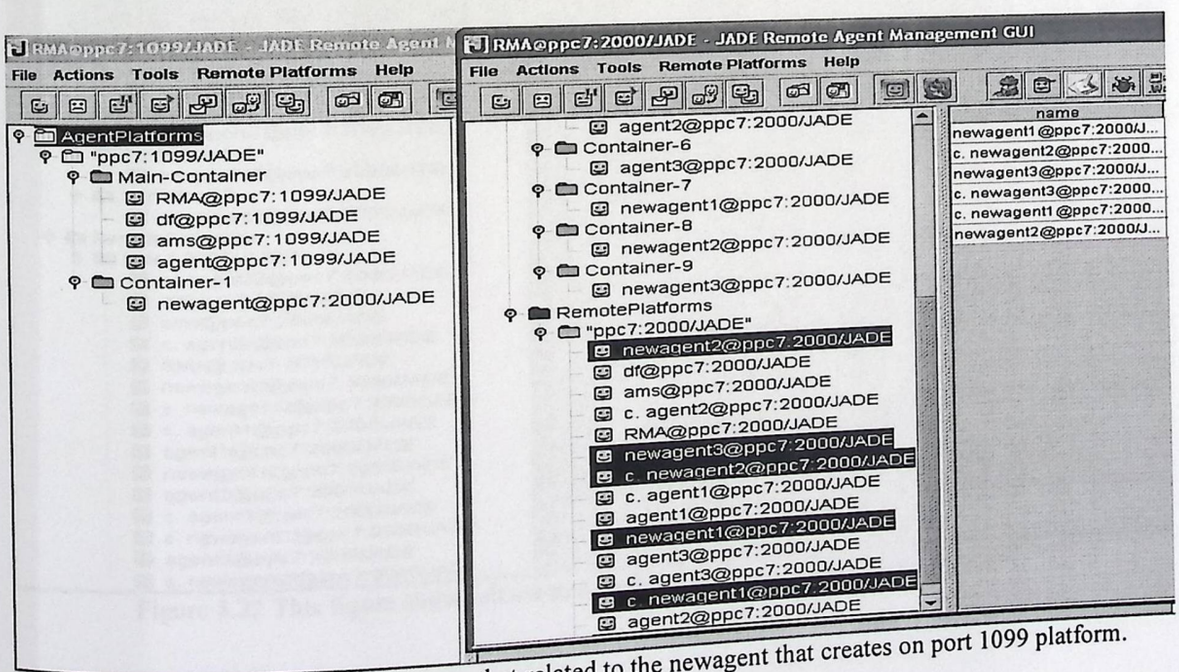


Figure 5.20 This figure shows the agents that related to the newagent that creates on port 1099 platform.

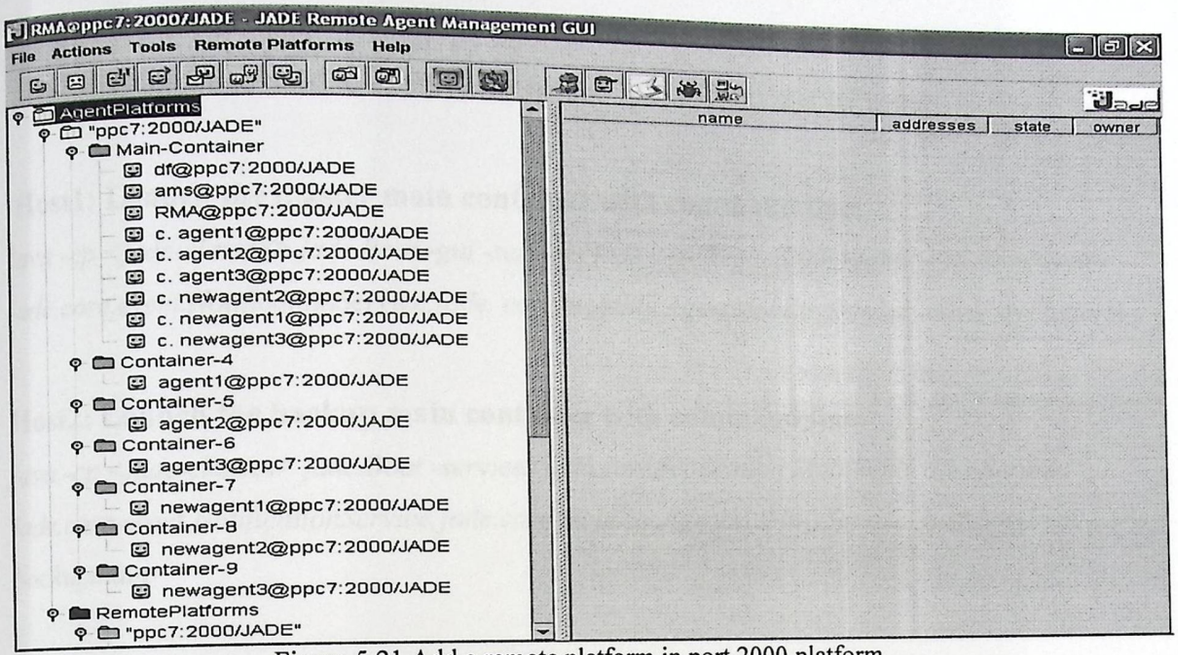


Figure 5.21 Add a remote platform in port 2000 platform.

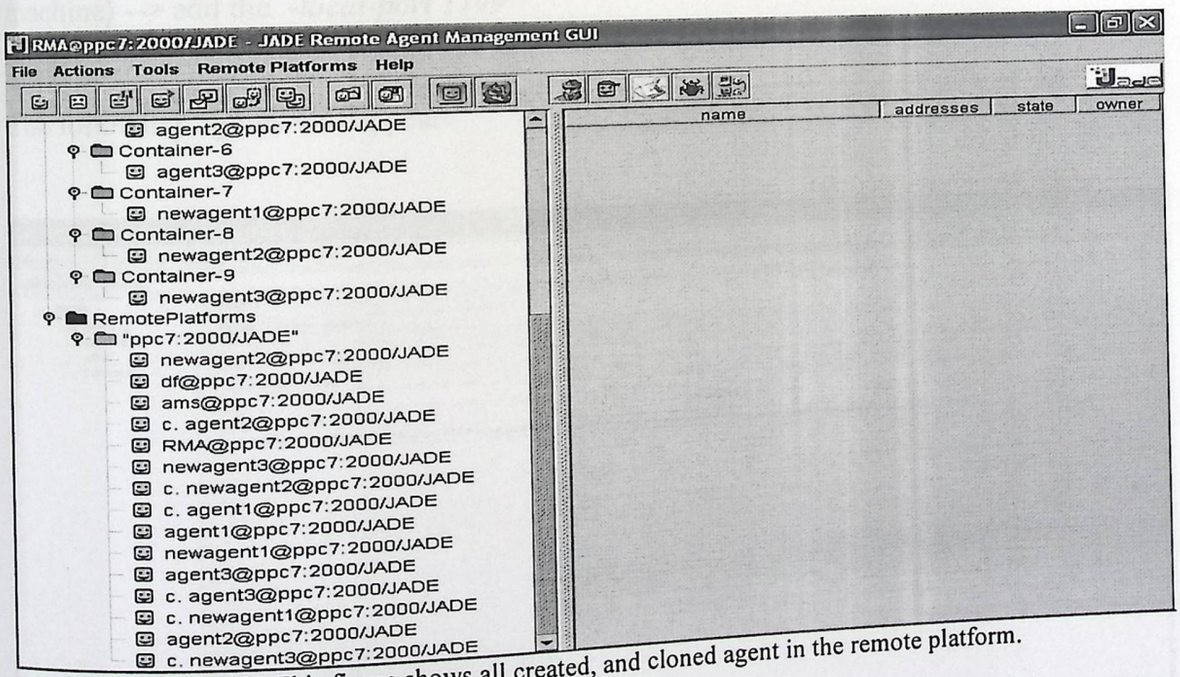


Figure 5.22 This figure shows all created, and cloned agent in the remote platform.

5.2.2.4 –backupmain

This command used to identify the master Main-Container by using

Host1: Launch the master main container with command line:

java -cp <jade-classes> jade.Boot -gui -services jade.core.replication.MainReplicationService ; jade.core.event.NotificationService;jade. core.mobility.AgentMobilityService -name myPlatform

Host2: Launch the backup main container with command line:

java -cp <jade-classes> jade.Boot -services jade.core.replication.MainReplicationService; jade.core.event.NotificationService;jade.core.mobility.AgentMobilityService -host host1 -backupmain

If Host2 == Host1 (i.e. if you are launching the master and backup main on the same machine) --> add the *-local-port 1199*

The following figures will appear:

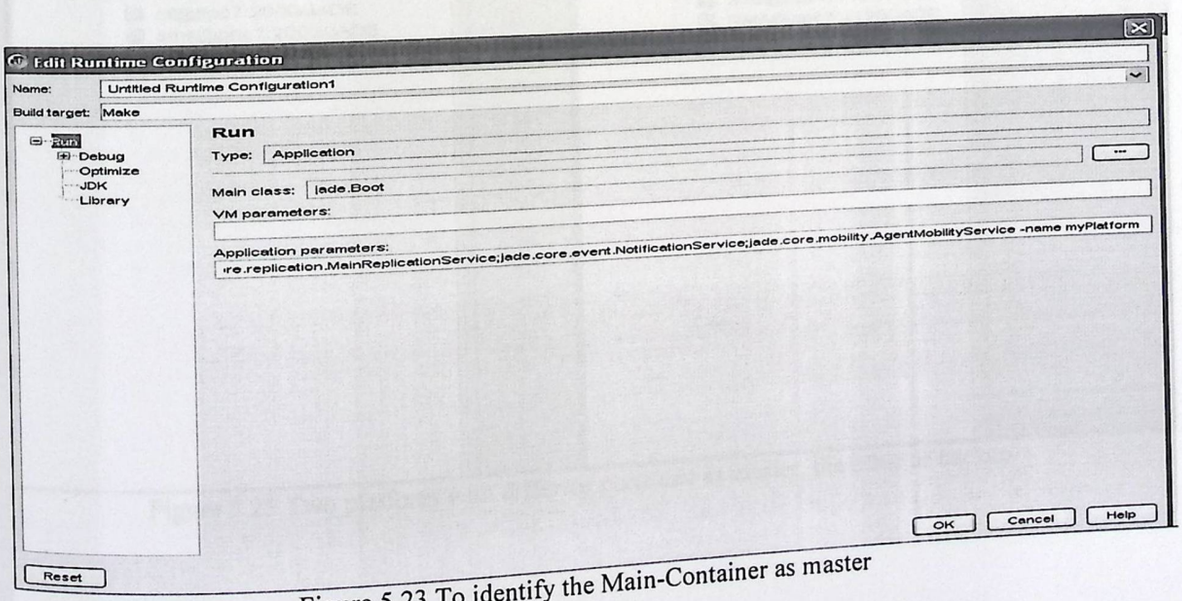


Figure 5.23 To identify the Main-Container as master

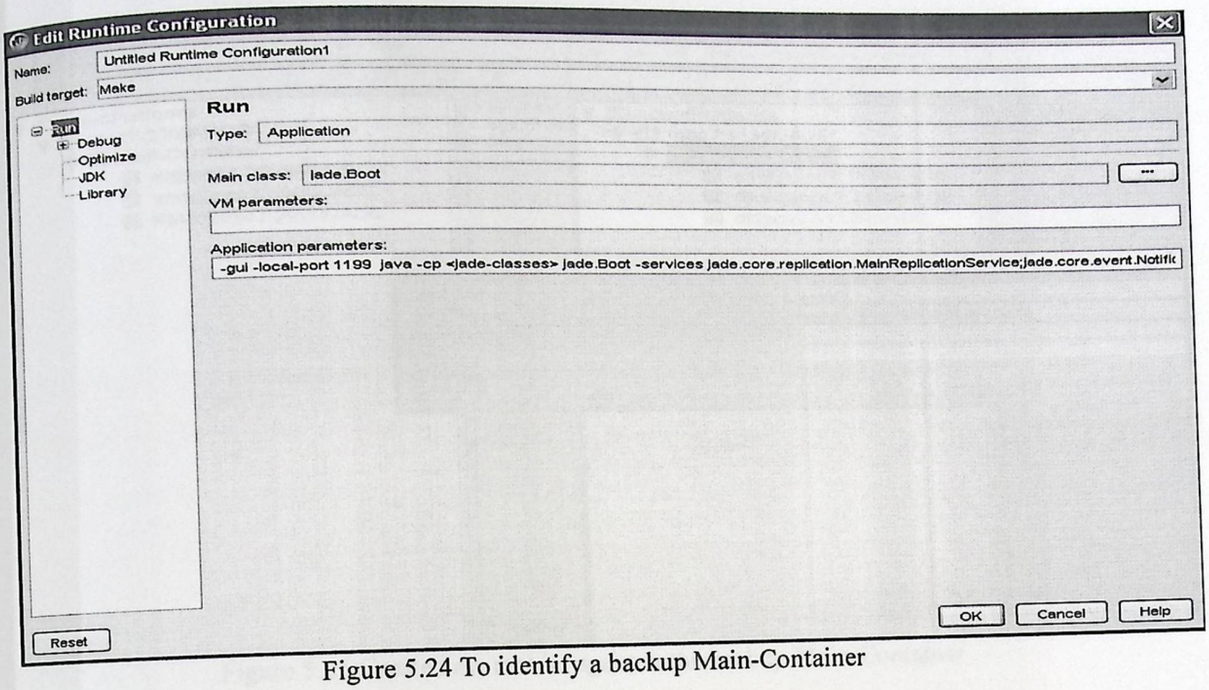


Figure 5.24 To identify a backup Main-Container

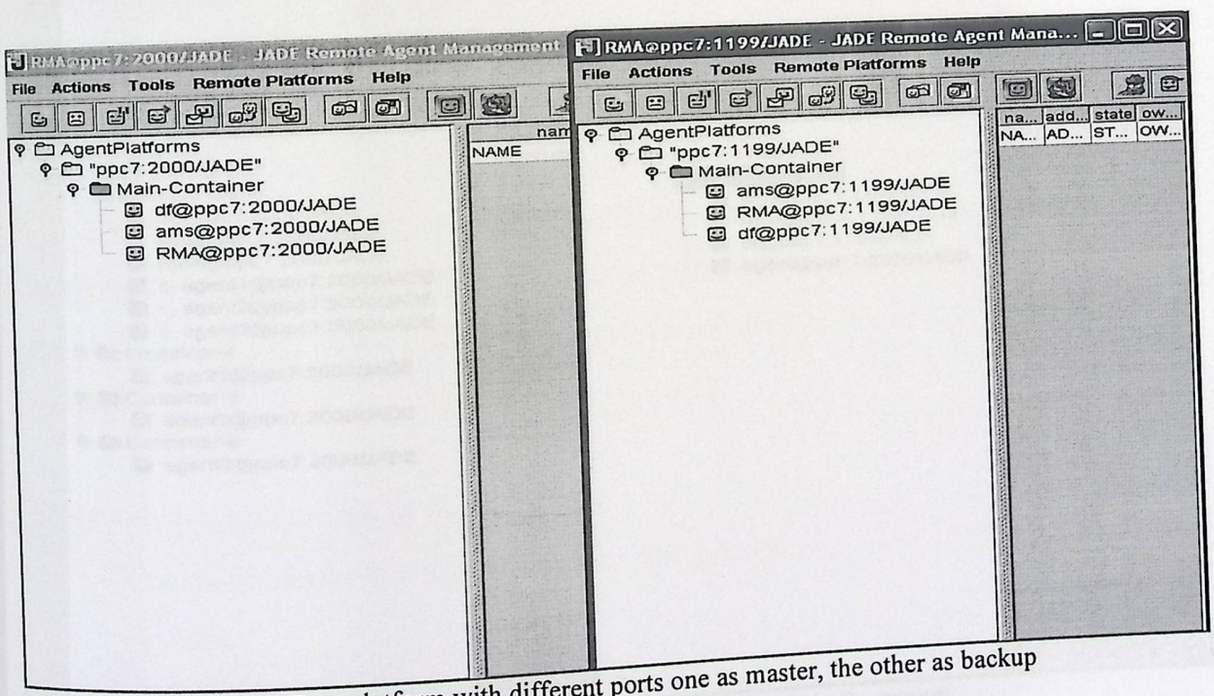


Figure 5.25 Two platform with different ports one as master, the other as backup

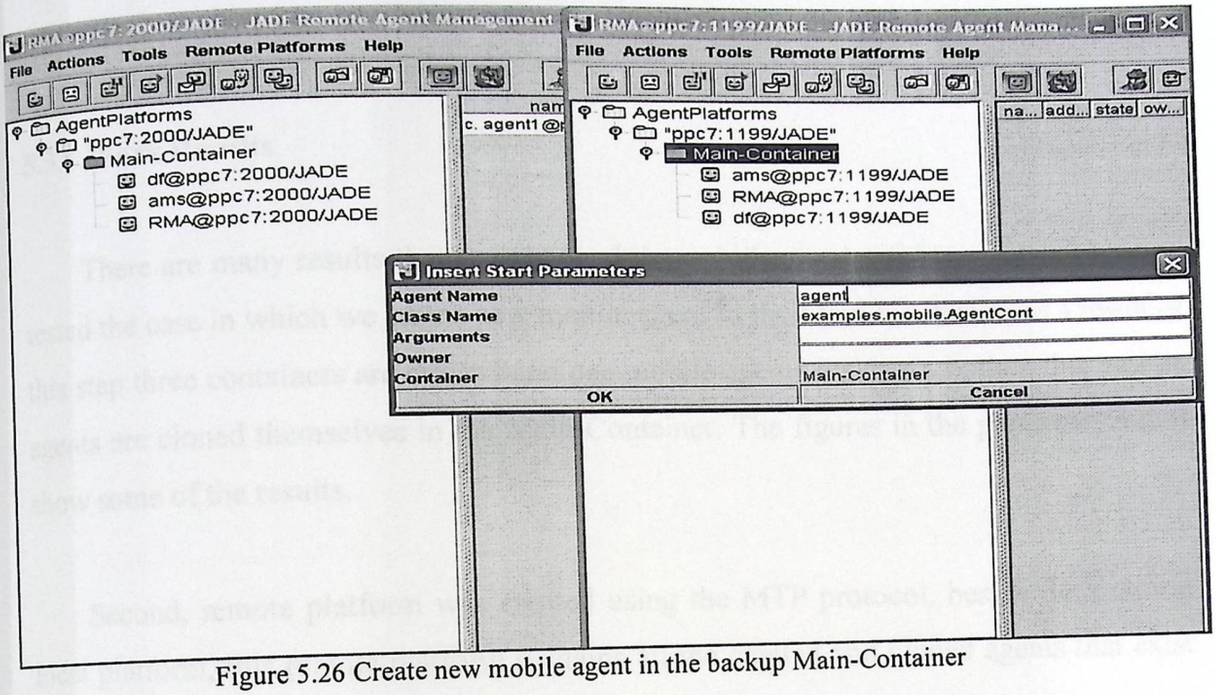


Figure 5.26 Create new mobile agent in the backup Main-Container

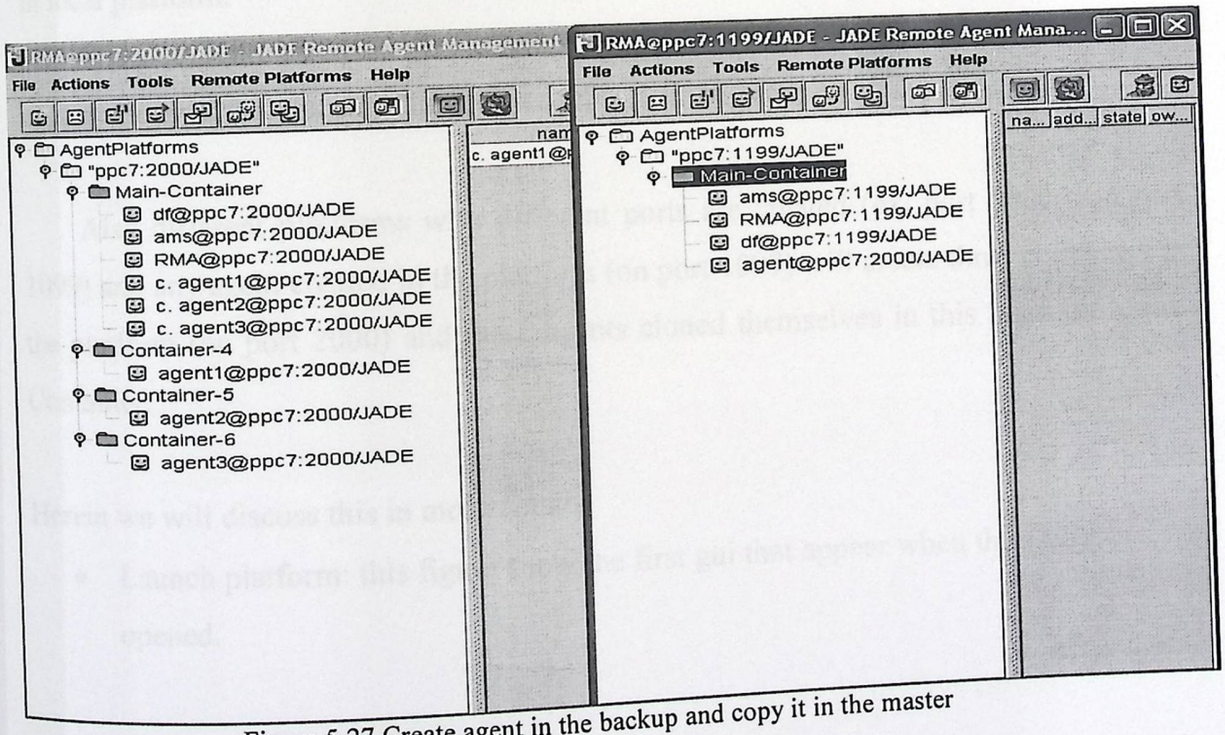


Figure 5.27 Create agent in the backup and copy it in the master

5.3 Testing

5.3.1 Work Results

There are many results that are obtained through the work of this project. First, we tested the case in which we create new mobile agent in the Main-Container, as a result of this step three containers are created and one mobile agents inside each, then this mobile agents are cloned themselves in the Main-Container. The figures in the previous section show some of the results.

Second, remote platform was created using the MTP protocol, beside the existing local platform, this remote platform contains all the created and cloned agents that exist in local platform.

And as another result the created agent can migrate between the different containers, and if any agent killed a list of the visited containers name is appeared.

Also different platforms with different ports are created (ex. port 2000, and port 1099) and any agent created in the platform (on port 1099) will create three containers in the platform (on port 2000) and these agents cloned themselves in this platform Main-Container.

Herein we will discuss this in more details.

- Launch platform: this figure show the first gui that appear when the JADE is opened.

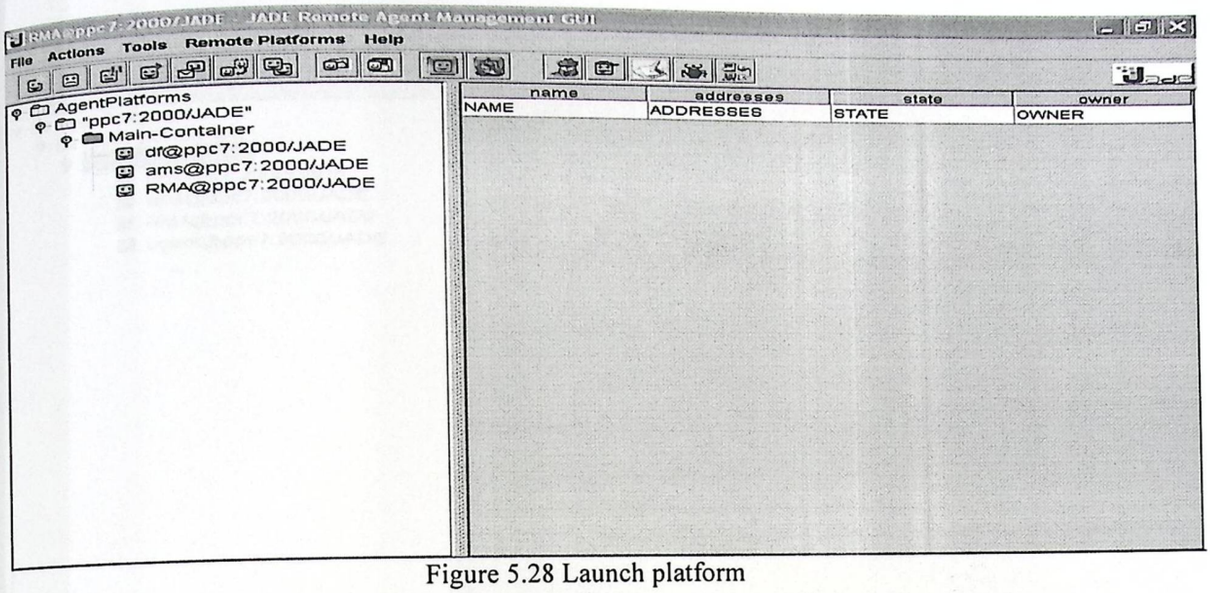


Figure 5.28 Launch platform

- Create stationary agent: this figure show how to create stationary agent.

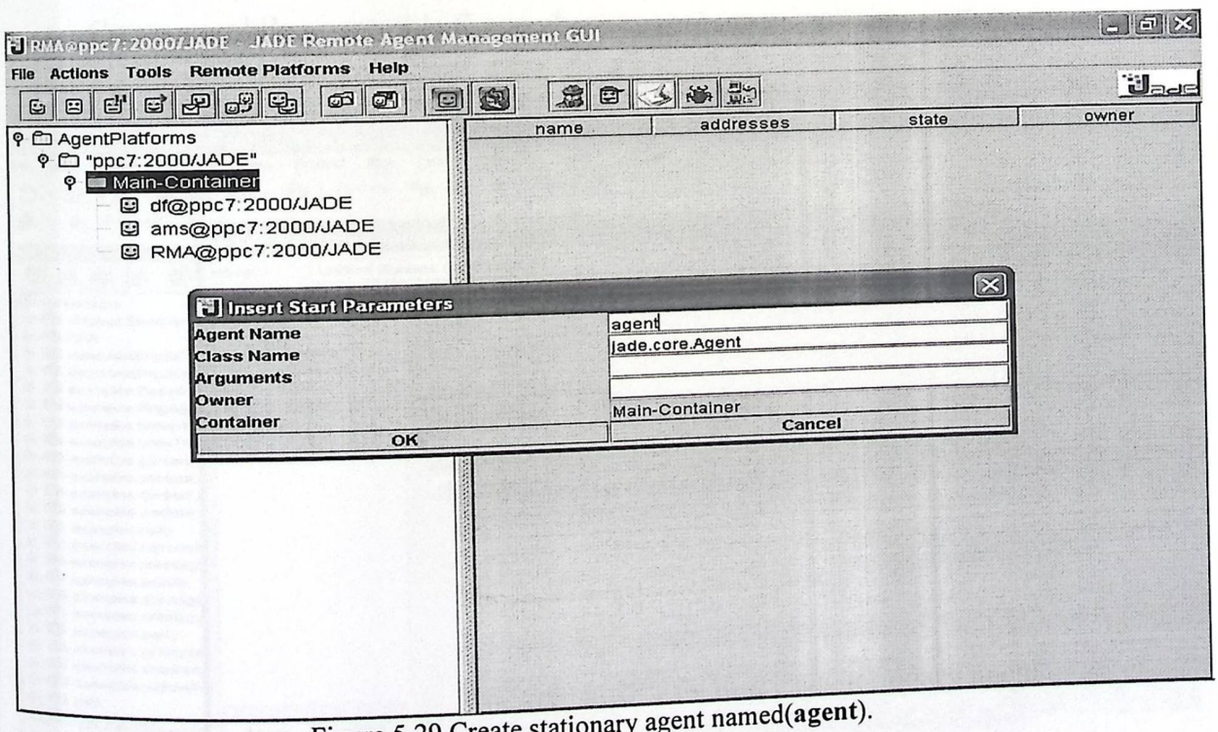


Figure 5.29 Create stationary agent named(**agent**).

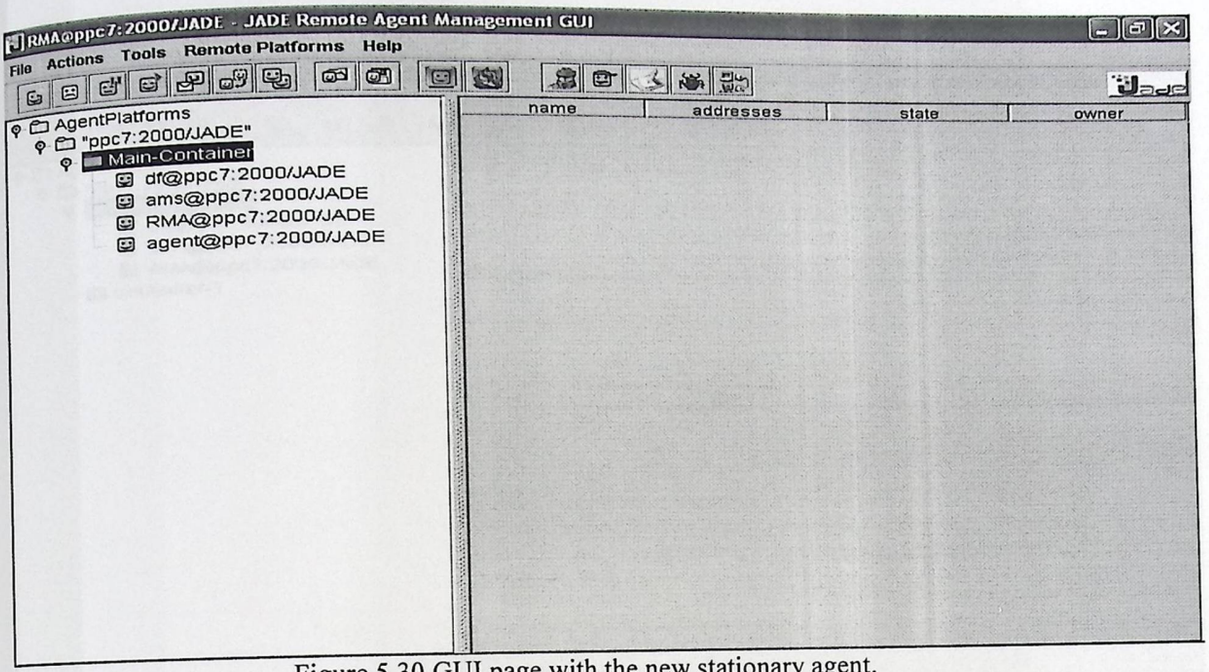


Figure 5.30 GUI page with the new stationary agent.

- Create a mobile agent: this figure show how to create mobile agent.

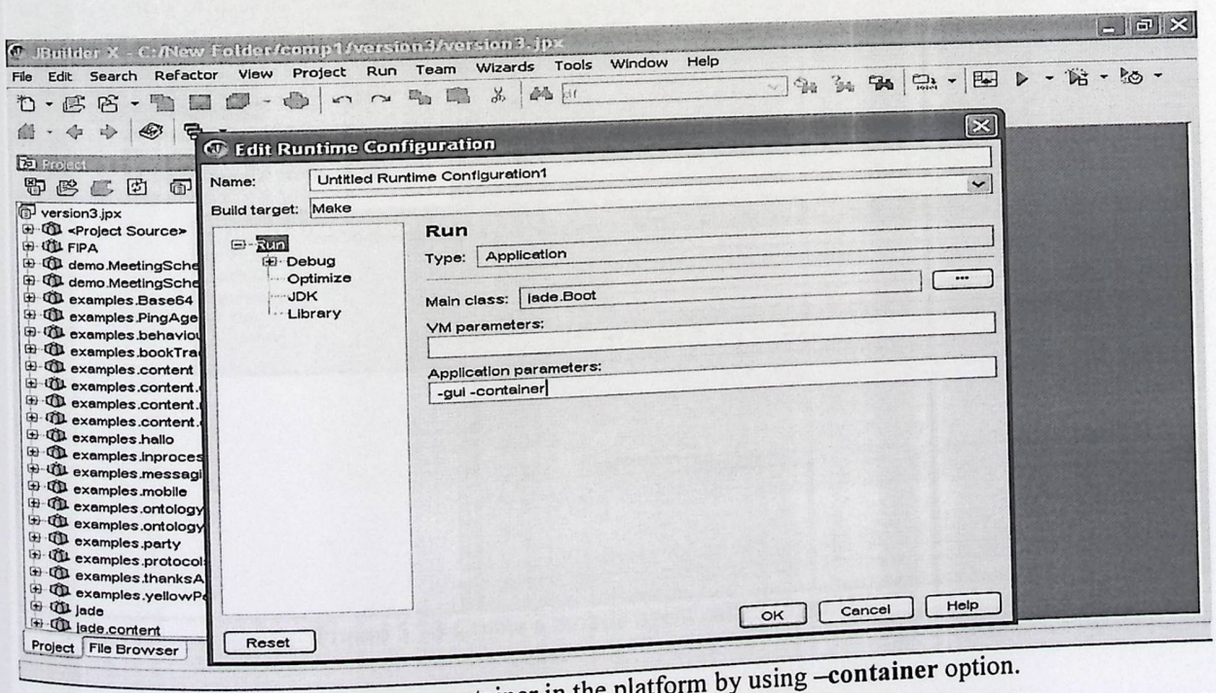


Figure 5.31 Create new container in the platform by using **–container** option.

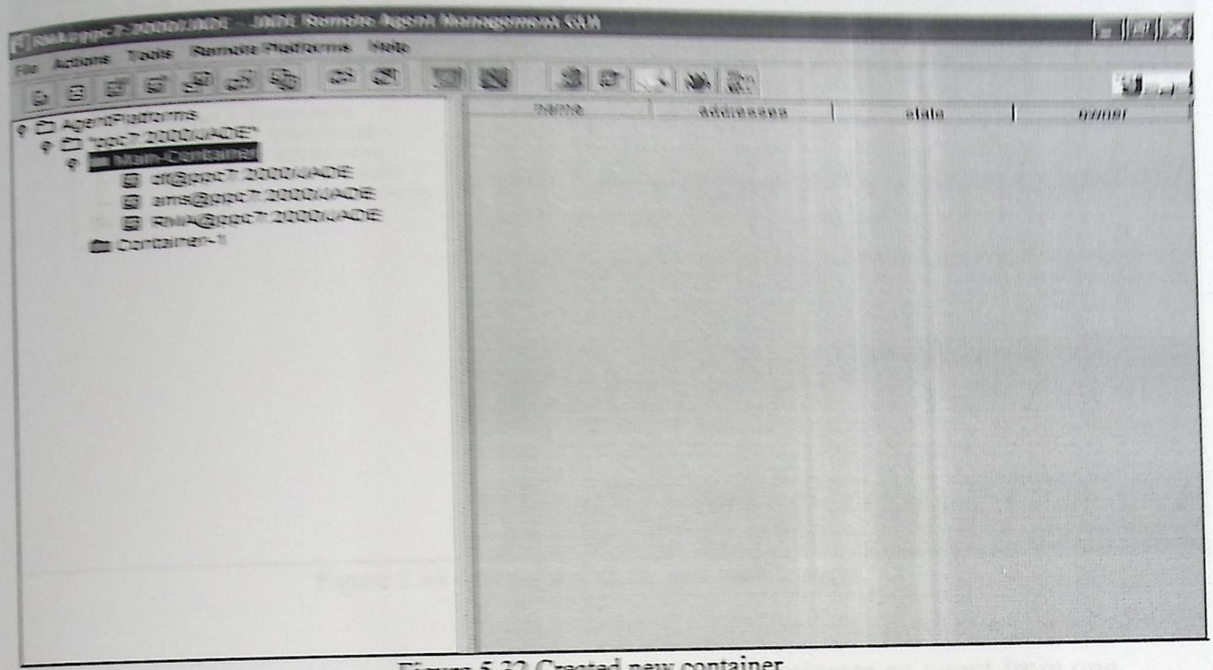


Figure 5.32 Created new container.

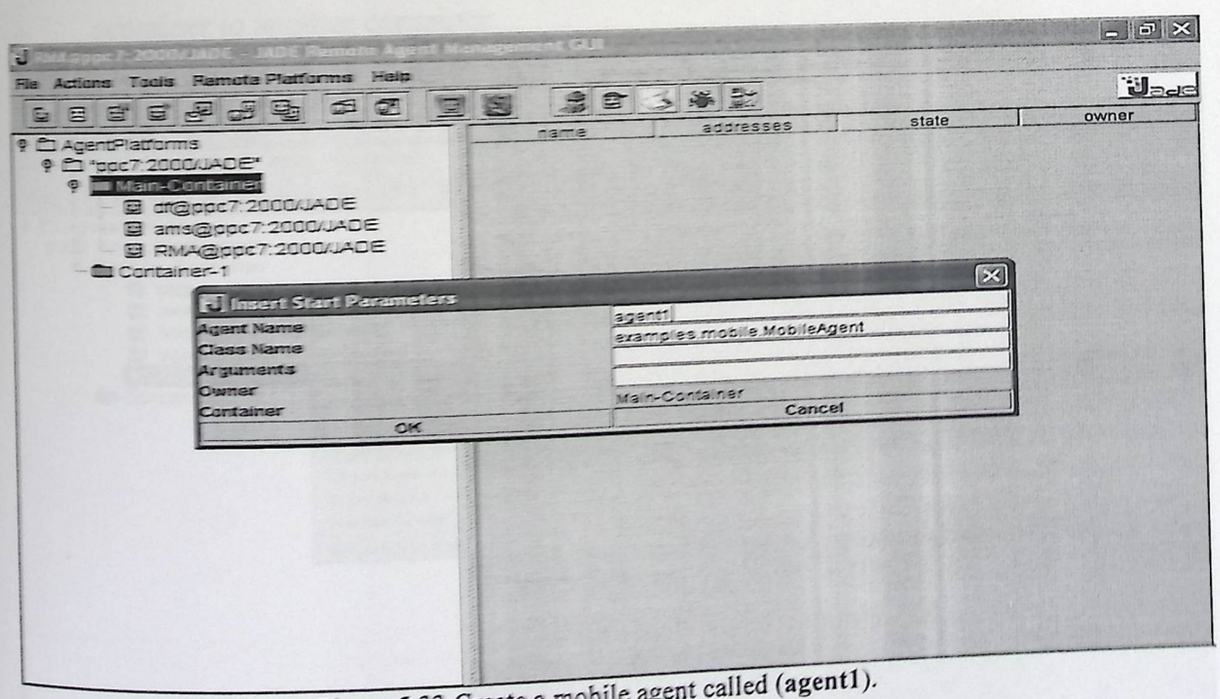


Figure 5.33 Create a mobile agent called (**agent1**).

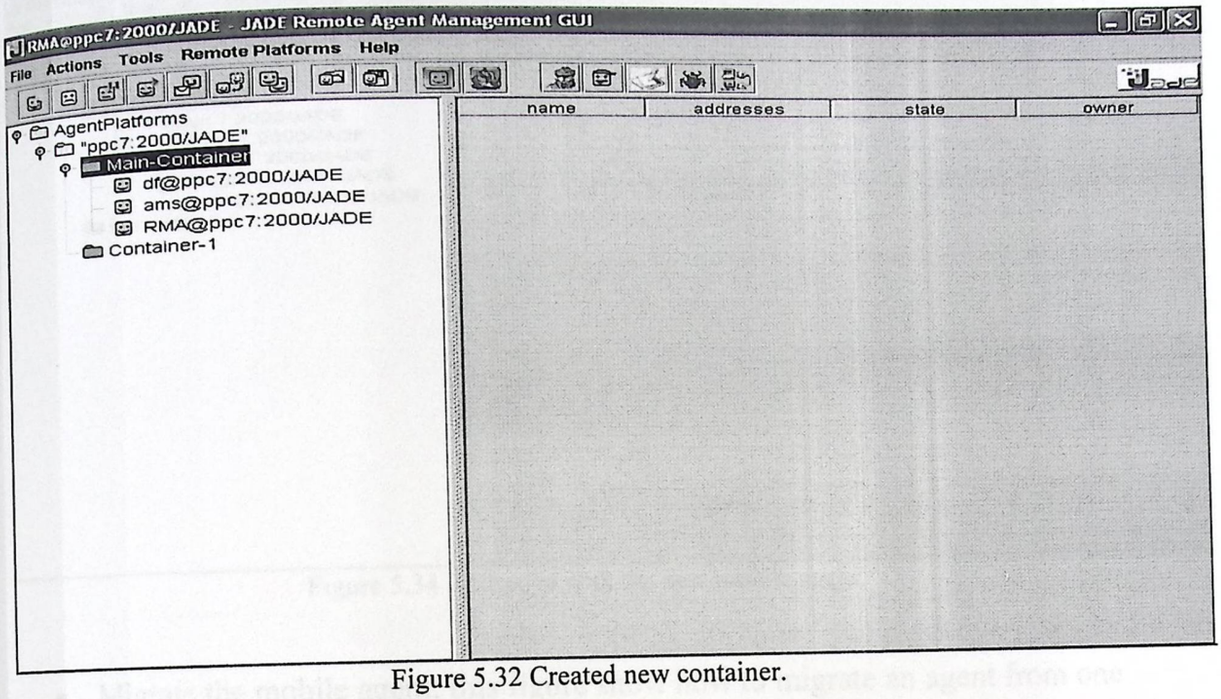


Figure 5.32 Created new container.

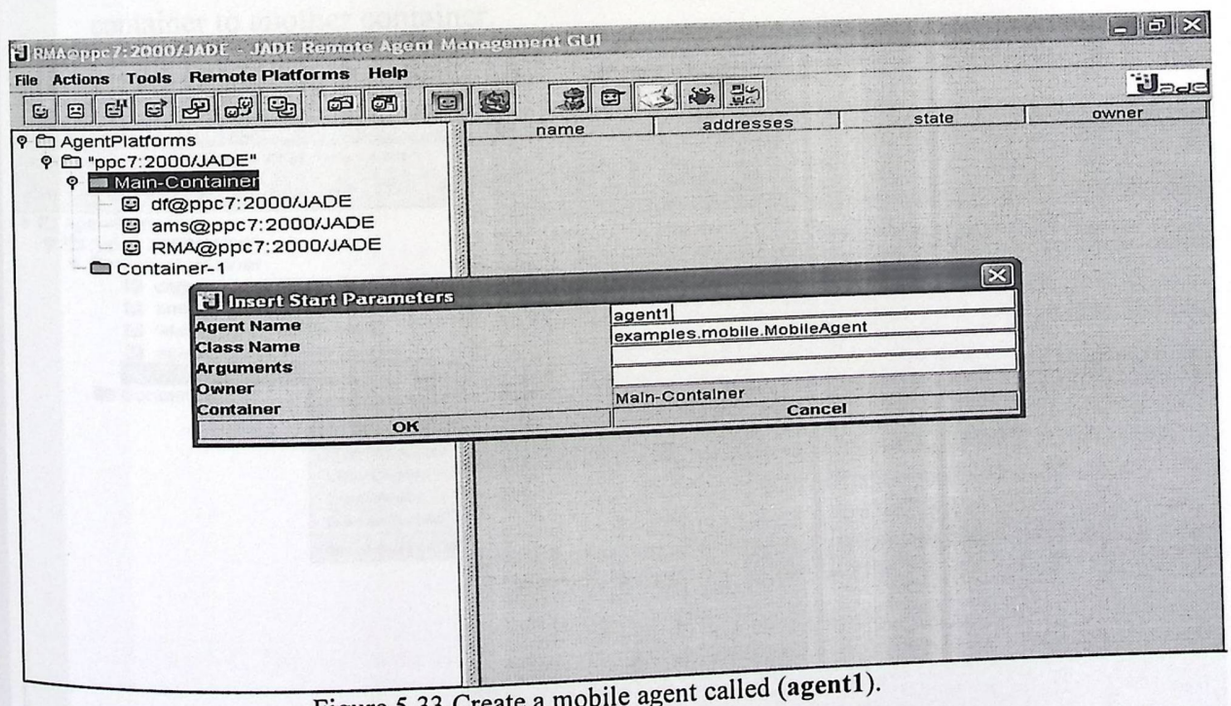


Figure 5.33 Create a mobile agent called (**agent1**).

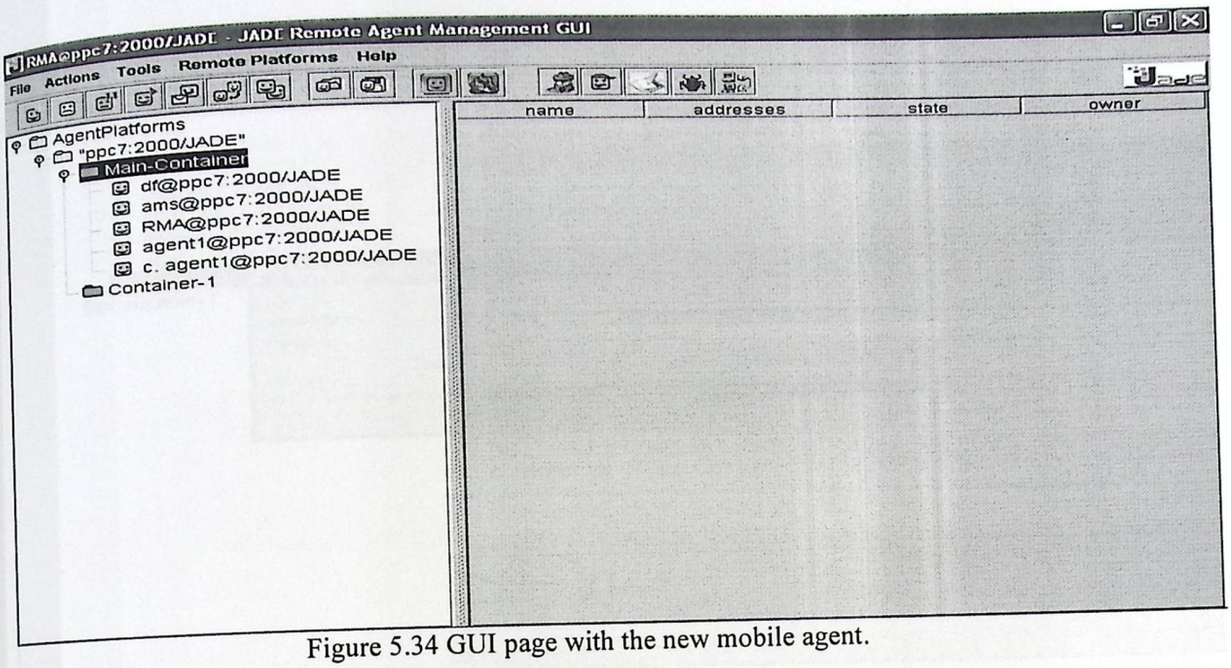


Figure 5.34 GUI page with the new mobile agent.

- Migrate the mobile agent: this figure show how to migrate an agent from one container to another container.

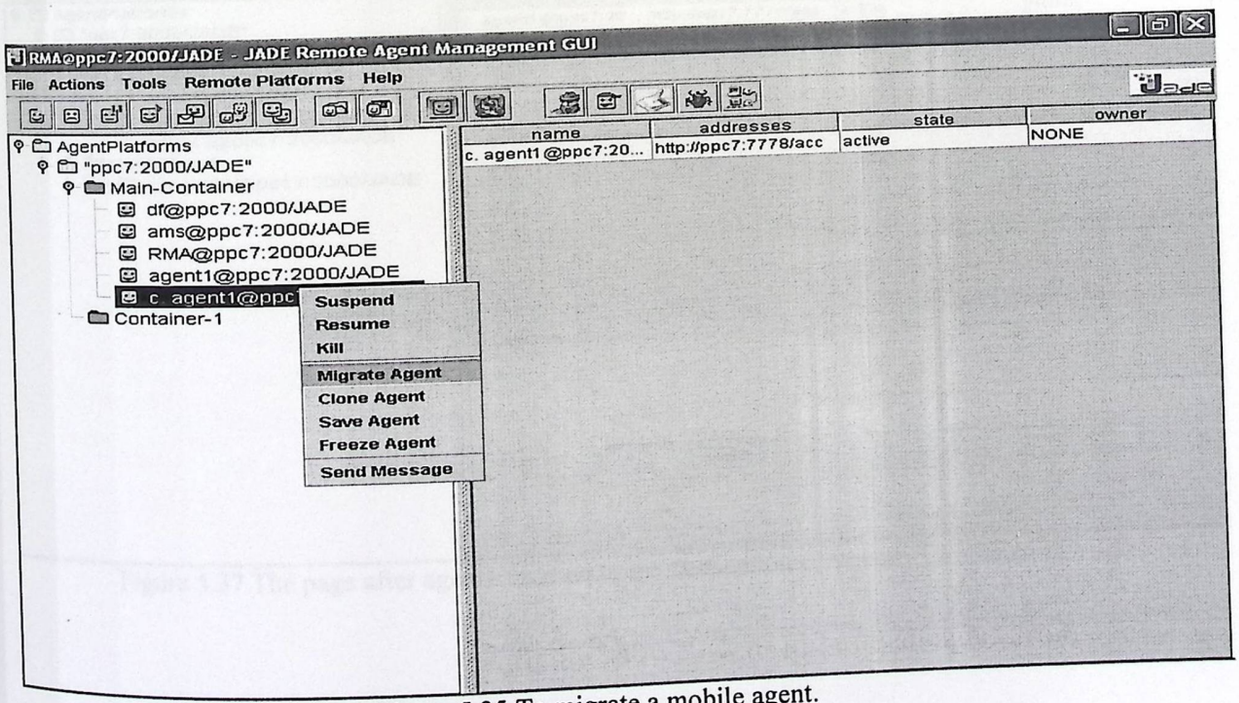


Figure 5.35 To migrate a mobile agent.

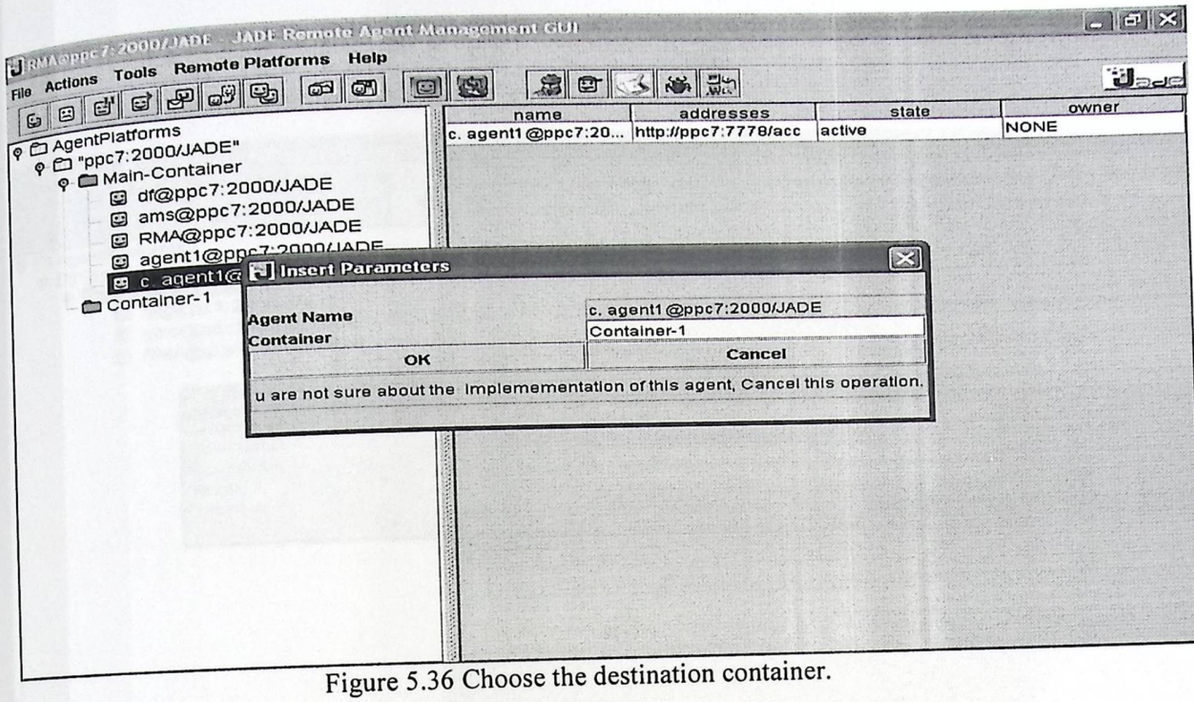


Figure 5.36 Choose the destination container.

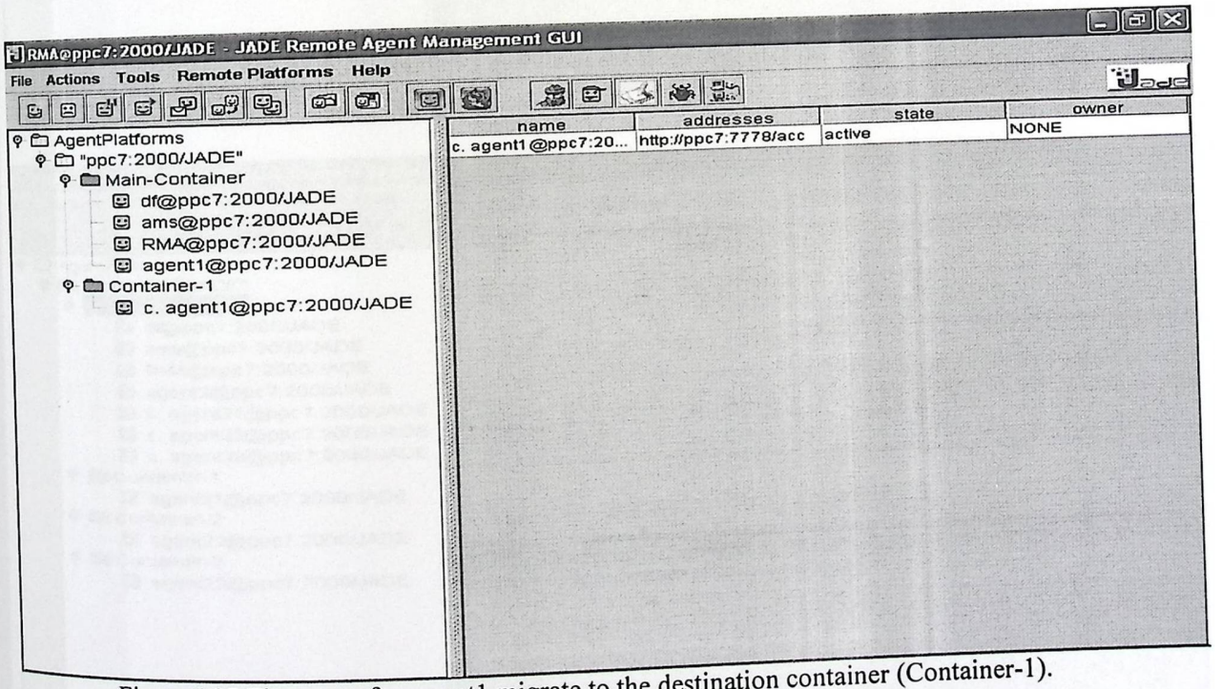


Figure 5.37 The page after agent1 migrate to the destination container (Container-1).

- Create a mobile agent that create three containers each with a mobile agent that cloned themselves to the Main-Container.

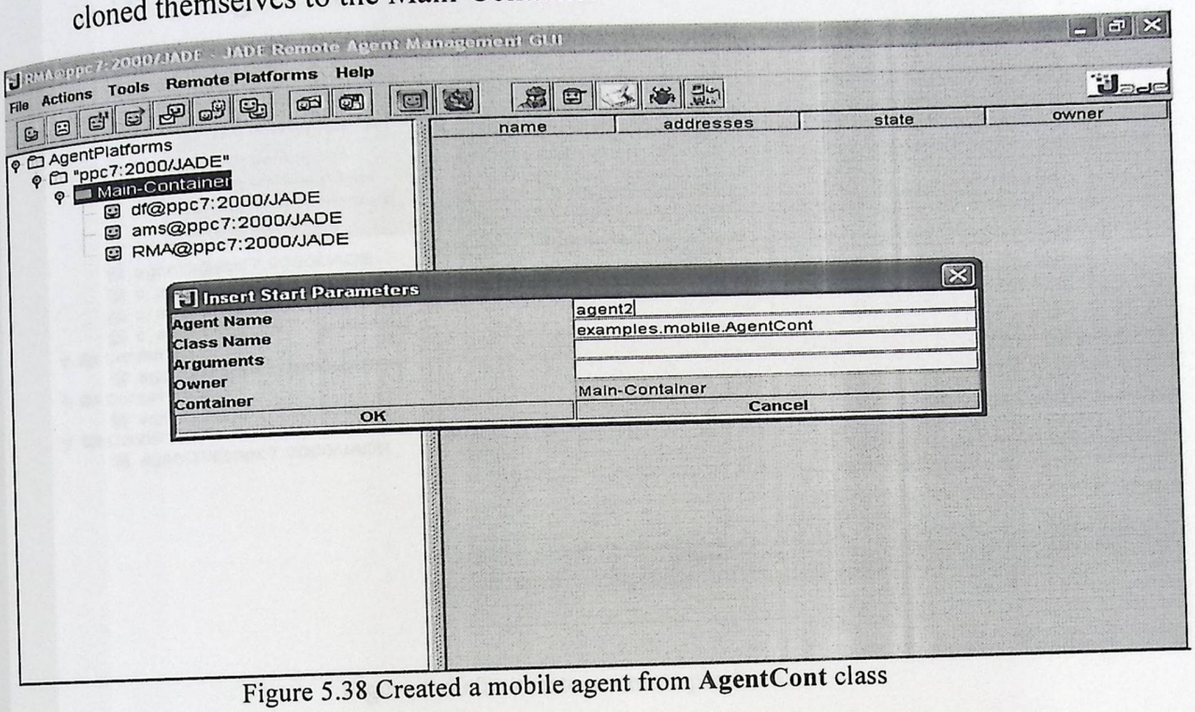


Figure 5.38 Created a mobile agent from **AgentCont** class

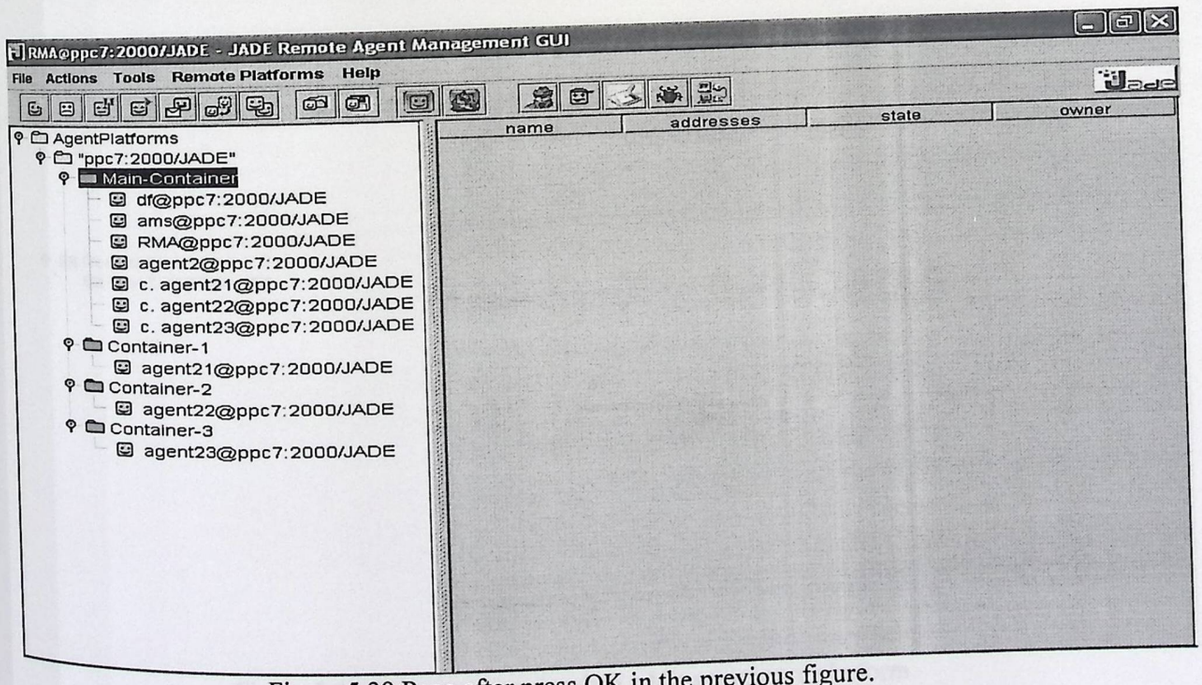


Figure 5.39 Page after press OK in the previous figure.

- Create a remote platform: this figure show how to create a remote platform beside the original one.

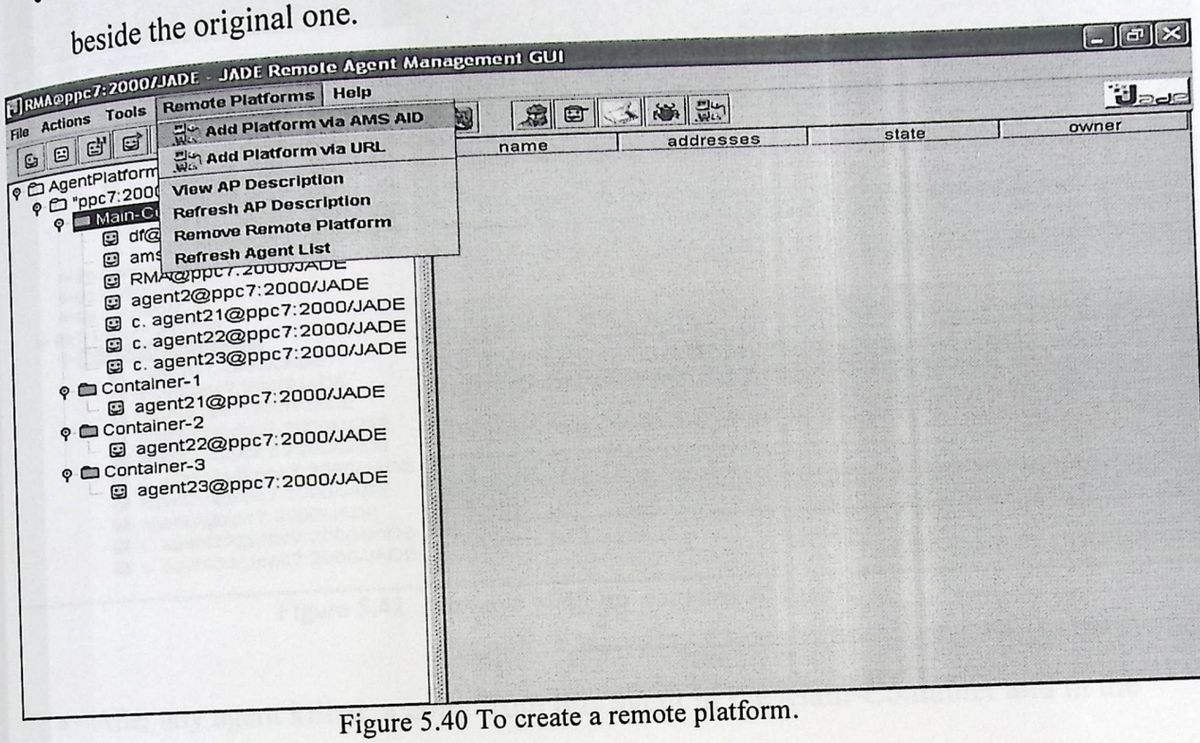


Figure 5.40 To create a remote platform.

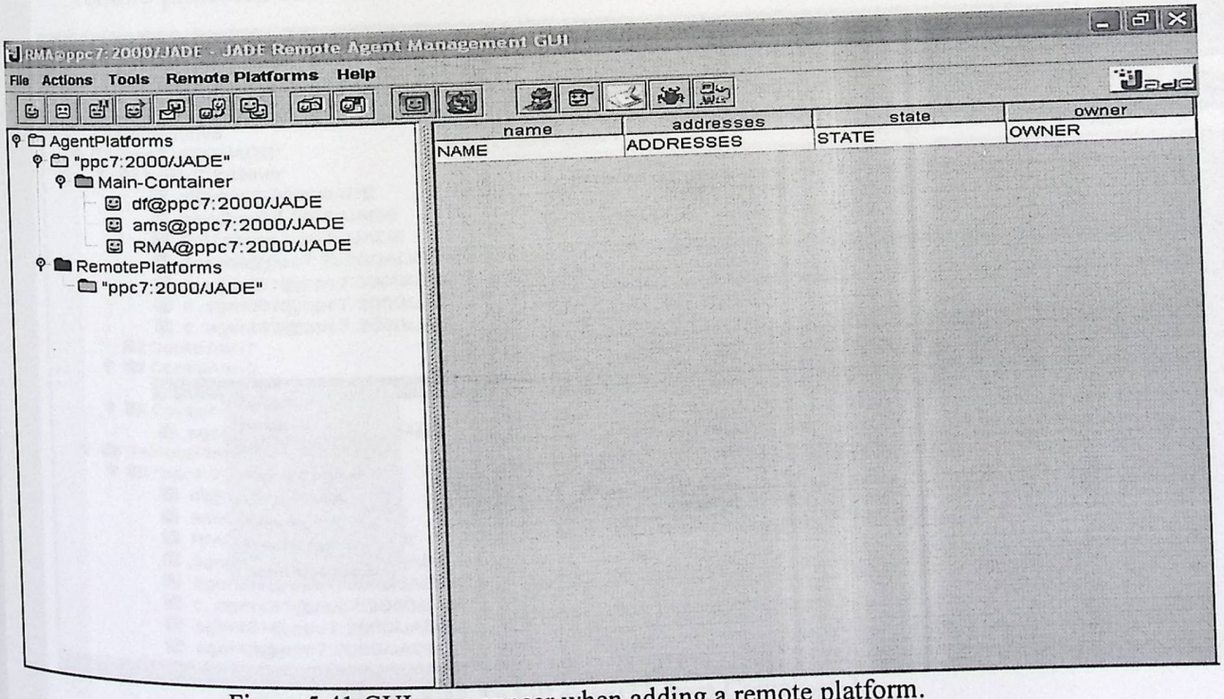


Figure 5.41 GUI page appear when adding a remote platform.

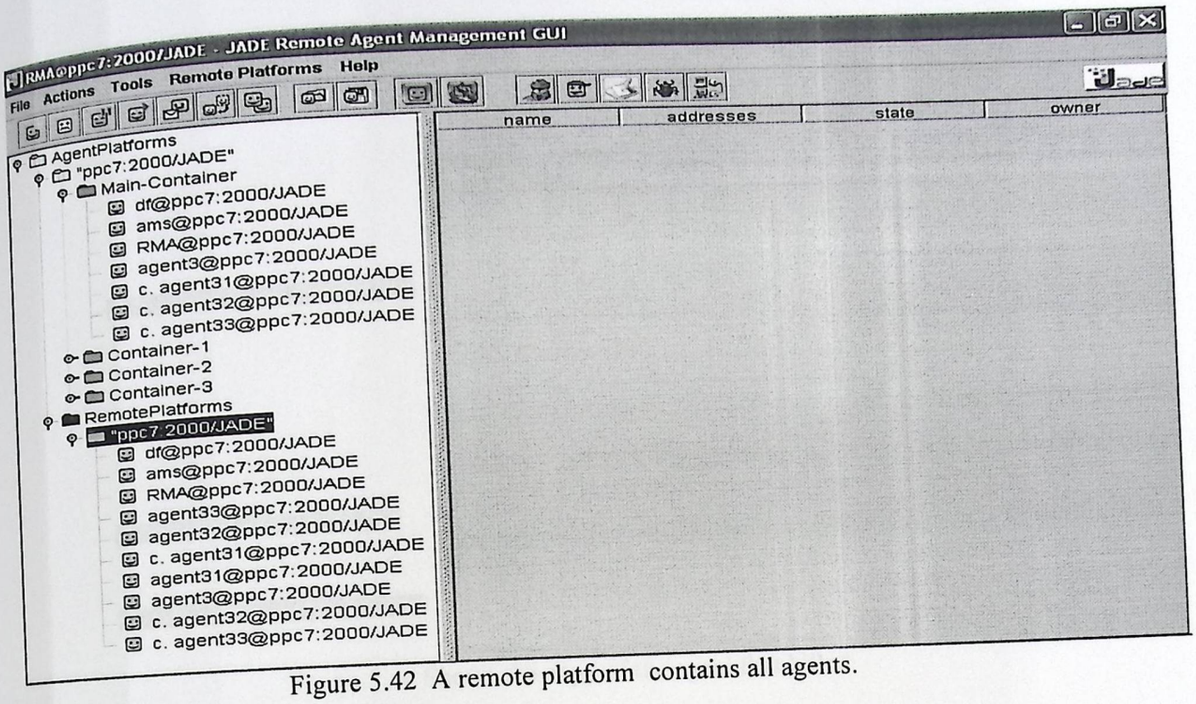


Figure 5.42 A remote platform contains all agents.

- After any agent killed, a copy from this agent in the Main-Container and in the remote platform still exist.

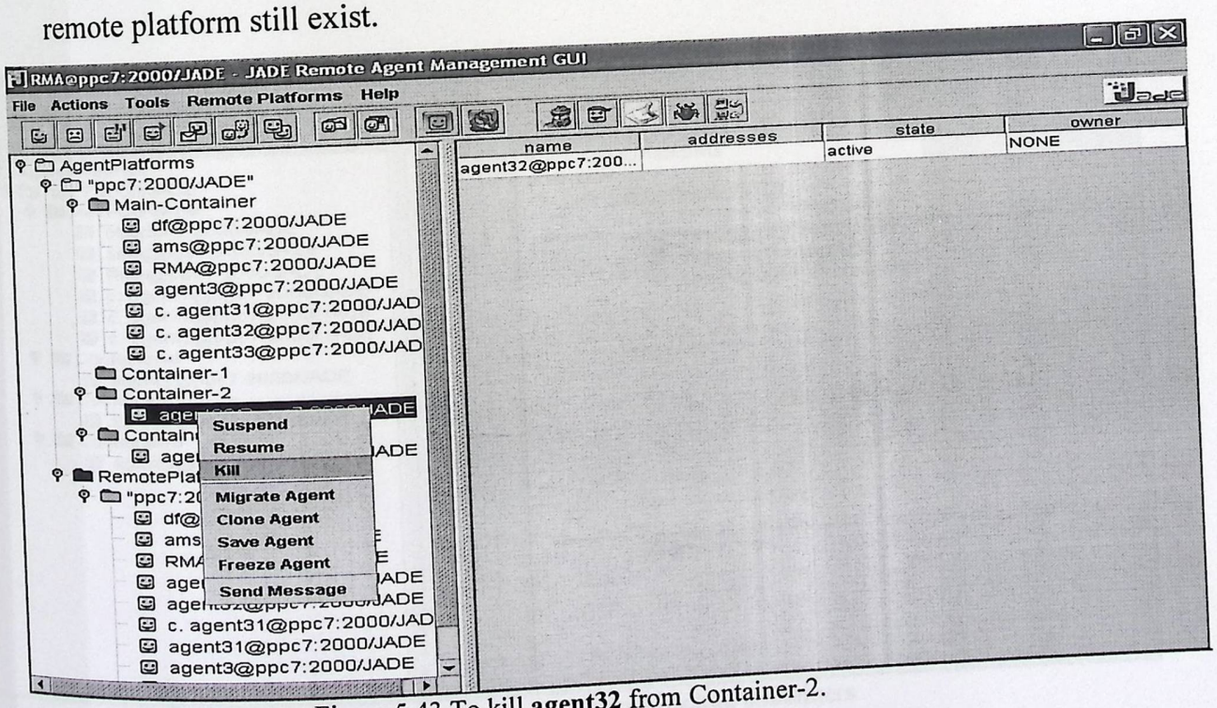


Figure 5.43 To kill **agent32** from Container-2.

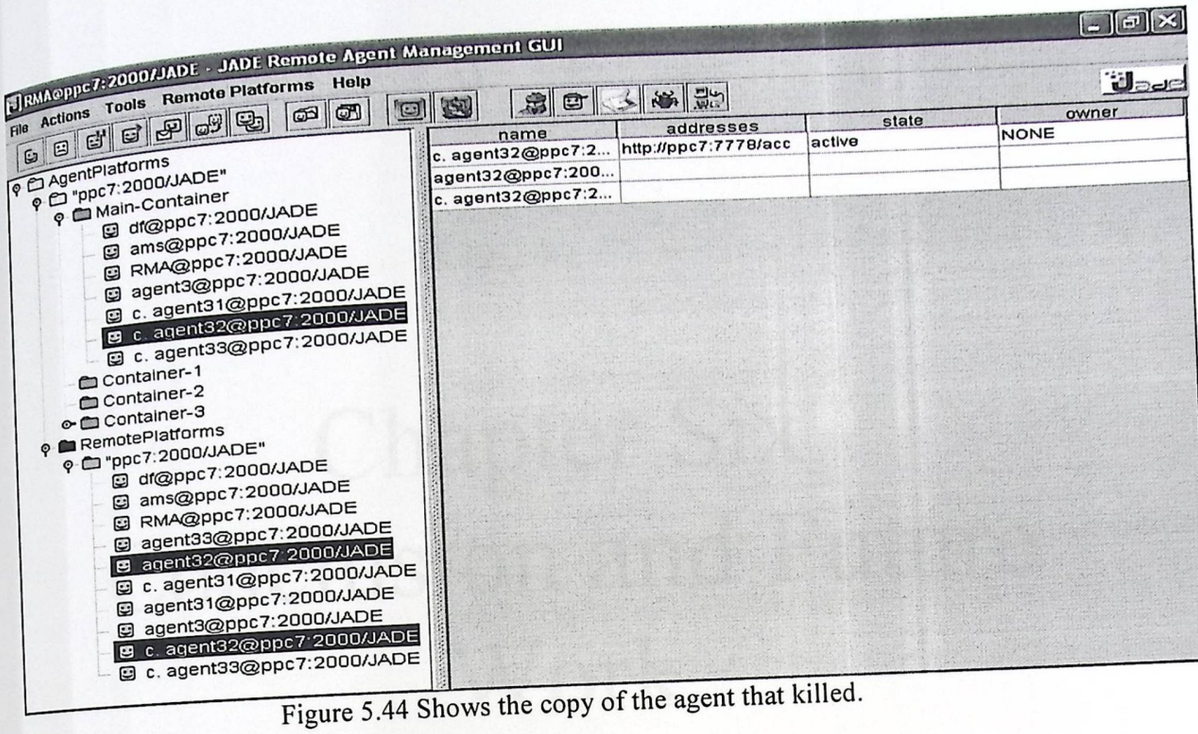


Figure 5.44 Shows the copy of the agent that killed.

- Create master and backup Main Containers:

The result after apply this step shown in the following figure:

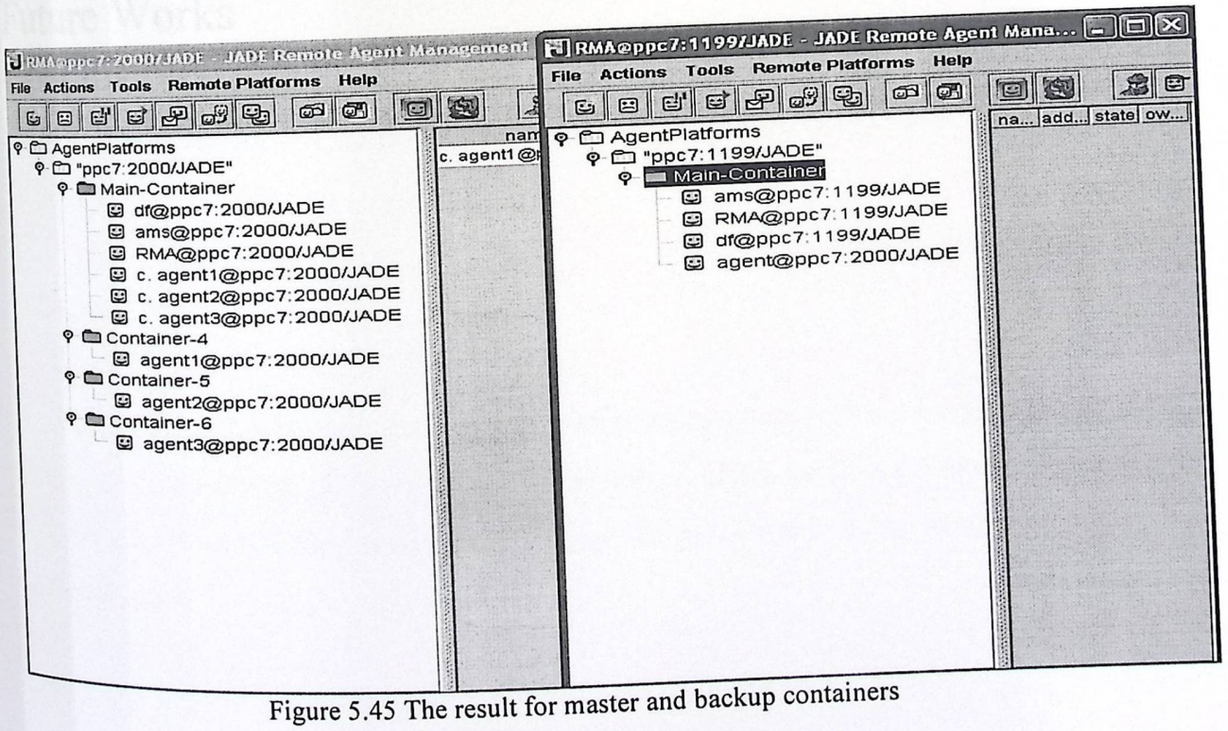


Figure 5.45 The result for master and backup containers

Chapter Six
Conclusion and Future Work

Future Works
Conclusions

Chapter Six

Conclusion and Future Work

This chapter is to talk about future works and conclusions related to the JADE simulation and Agent System.

6.1 Conclusions

In our system, many results and conclusions were appearing while dealing with the JADE simulation System. First, create new stationary or mobile agent, also create new containers and agents inside each, and clone the created agents in the Main-Container. And migrate the created agent between the different containers, when kill the created agent, and show the visited locations that was visit.

Then create a remote platform, beside the local platform, apply fault-tolerance using the JADE replication technique, and create different platforms with different ports.

Many problems appear when working in this project, the most important one that the JADE 3.3 that we use do not support the migrate option for mobile agent from host to host.

6.2 Future Works

In our system, we made part of the replication technique on the same host, with the same port or even with different ports. In the future, there will be a connection between different hosts and make replication among these containers by replicate agents in different hosts.

Appendix

```
            ...Agent {

            ...tainer=null;
            ... ac = new AgentContainer [5];
            ...Name = null, a2AgentName = null,
            ... null, c3 = null,c1 = null, c2 = null;
            ... a1 = null,a2=null,a1copy=null,
            ...,a3=null,a3copy=null;
            ... null;



    ...ln(getLocalName ()+" STARTED");
    ... getArguments ();

    ...null && args.length > 0) {
    ...new AID((String) args[0], AID.ISLOCALNAME);
        }
    ... = getLocalName ()+"1";
    ... = getLocalName ()+"2";
    ... = getLocalName ()+"3";
    ...+a1AgentName ;
    ...+a2AgentName ;
    ...+a3AgentName ;


...tainer = (AgentContainer )getContainerController ();

...time rt = Runtime.instance ();
...ofileImpl p = new ProfileImpl (false);

ac[0] = rt.createAgentContainer (p);
a1 = ac[0].createNewAgent (a1AgentName ,
    "examples.mobile.MobileAgent" , null);
a1.start ();
...tem.out.println (getLocalName ()+" CREATED AND STARTED NEW agent:"
...a1AgentName + " ON CONTAINER " +Mcontainer.getContainerName ());
  if (! here().getName ().equals ("Main-Container" ))
  {System.out.println ("\n this is not the main container \n" ); }
}
catch (Exception any) {
any.printStackTrace ();
}

Runtime rt = Runtime.instance ();
ProfileImpl p = new ProfileImpl (false);

try {
   ac[1] = rt.createAgentContainer (p);
   AgentController a2 = ac[1].createNewAgent (a2AgentName ,
   "examples.mobile.MobileAgent" ,new Object[0]);
   a2.start ();
}
catch (Exception e2) {
e2.printStackTrace ();
}
rt = Runtime.instance ();
p = new ProfileImpl (false);

try {
```

```
package examples.mobile;

import jade.core.AID;
import jade.core.Runtime;
import jade.core.ProfileImpl;

import jade.wrapper.*;
import jade.core.Location;

    public class AgentCont extends MobileAgent {

    public static AgentContainer Mcontainer=null;
    public  static AgentContainer[] ac = new AgentContainer[5];
    public static String a1AgentName = null, a2AgentName = null,
                a3AgentName = null, c3 = null,c1 = null, c2 = null;
    public static AgentController a1 = null,a2=null,a1copy=null,
                a2copy=null,a3=null,a3copy=null;
    private AID initiator = null;

    public void setup () {

      System.out.println(getLocalName()+" STARTED");
      Object[] args = getArguments();

      if (args != null && args.length > 0) {
      initiator = new AID((String) args[0], AID.ISLOCALNAME);
                        }
      a1AgentName = getLocalName()+"1";
      a2AgentName = getLocalName()+"2";
      a3AgentName = getLocalName()+"3";
      c1="c. "+a1AgentName;
      c2="c. "+a2AgentName;
      c3="c. "+a3AgentName;

     try {

        Mcontainer = (AgentContainer)getContainerController();

        Runtime rt = Runtime.instance();
        ProfileImpl p = new ProfileImpl(false);

        ac[0] = rt.createAgentContainer(p);
        a1 = ac[0].createNewAgent(a1AgentName,
             "examples.mobile.MobileAgent" , null);
        a1.start();
     System.out.println(getLocalName()+" CREATED AND STARTED NEW agent:"
     +a1AgentName + " ON CONTAINER "+Mcontainer.getContainerName());
        if (! here().getName().equals("Main-Container"))
        {System.out.println("\n this is not the main container \n" ); }
     }
     catch (Exception any) {
     any.printStackTrace();
     }

     Runtime rt = Runtime.instance();
     ProfileImpl p = new ProfileImpl(false);

     try {
         ac[1] = rt.createAgentContainer(p);
         AgentController a2 = ac[1].createNewAgent(a2AgentName,
         "examples.mobile.MobileAgent" ,new Object[0]);
         a2.start();
     }
     catch (Exception e2) {
     e2.printStackTrace();
     }
     rt = Runtime.instance();
     p = new ProfileImpl(false);

      try {
```

```
    "examples.mobile.MobileAgent" ,new Object[1]);
    a3.start();

  }
  catch (Exception e2) {
  e2.printStackTrace ();
  }

}
```

```
package examples.mobile;
import jade.core.*;
import jade.core.Location;
import jade.core.Agent.*;
import jade.core.behaviours.*;
import examples.mobile.MobileAgent;
import jade.core.Agent;
import jade.core.*;

public class AgentBehaviour extends SimpleBehaviour

{
MobileAgent myAgent;
AgentBehaviour(Agent a)

{
   super(a);
   myAgent = (MobileAgent )a;

}

public boolean done()

{
    return true;

}

public void action ()

{
    Location t = new locationtest();
    String newname = null;
    newname= "c. "+myAgent.getLocalName();
    myAgent.doClone(t,newname);
     myAgent.doMove(t);
    return;

}
```

```
package examples.mobile;
import jade.core.Location;

public class locationtest implements Location
 {
  public String getID ()
   {
      return "Main-Container@JADE-IMTP://ppc7" ;
   }
  public String getName ()
   {
  return "Main-Container" ;
   }
  public String getAddress ()
   {
   return "ppc7";
   }
  public String getProtocol ()
   {
  return "JADE-IMTP" ;
   }
}
```

References

[1] Brain Brewington, Daniela Rus, David Kotz, George Cybenko,Katsuhiro Moizumi, and Robert Gray," Mobile agents in distributed information retrieval", Thayer School of Engineering / Department of Computer Science, Dartmouth College, Hanover, New Hampshire, 1999.

[2] www.sei.cmu.edu/str/descriptions/**clientserver**.html

[3] http://www.cs.cf.ac.uk/Dave/C/node33.html

[4] www.the**mobileagent**.co.uk/

[5] Bjorn Hermans, "Intelligent Software Agents on the Internet: an inventory of currently offered functionality in the information society & a predication of (near-) future developments", Tilburg University, Tilburg, The Netherlands, 9th of July 1996/

[6] www.php.net/xml**rpc**

[7] en.wikipedia.org/wiki/Code_on_**demand**

[8] www.vdc-corp.com/autoid/press/05/pr05-38.html٣٦ -

[9]http://www-unix.mcs.anl.gov/mpi/

[10] www.llnl.gov/computing/tutorials/mpi/

[11] http://agent.cs.dartmouth.edu/software/agent2.1/

[12] www.worldscibooks.com/compsci/4283.html

[13] http://csrc.nist.gov/publications/nistpubs/800-19/sp800-19.pdf

[14] www.cetus-links.org/oo_**mobile_agents**.html

[15] http://www.jclark.com/jade/

[16] http://www.gemstone.org/gem-by-gem/english/jade.html

[17] http://www.fipa.org/specs/fipa00061/SC00061G.pdf

[18] www.etse.urv.es/recerca/ banzai/toni/MAS/FIPA2000/XC00061D.doc

[19] http://JADE.tilab.com

[20] Nicholas R. Jennings, Michael J. Wooldridge, "Agent Technology Foundations, Applications and Markets", Springer- Verlag Berlin Heidelberg New York, 199